Generative Learning Structures and Processes for Generalized Connectionist Networks

Vasant Honavar
Department of Computer Science
Iowa State University

Leonard Uhr
Computer Sciences Department
University of Wisconsin-Madison

Abstract

Massively parallel networks of relatively simple computing elements offer an attractive and versatile framework for exploring a variety of learning structures and processes for intelligent systems. This paper briefly summarizes some popular learning structures and processes used in such networks. It outlines a range of potentially more powerful alternatives for pattern-directed inductive learning in such systems. It motivates and develops a class of new learning algorithms for massively parallel networks of simple computing elements. We call this class of learning processes *generative* for they offer a set of mechanisms for constructive and adaptive determination of the network architecture - the number of processing elements and the connectivity among them - as a function of experience. Generative learning algorithms attempt to overcome some of the limitations of some approaches to learning in networks that rely on modification of *weights* on the links within an otherwise fixed network topology e.g., rather slow learning and the need for an a-priori choice of a network architecture. Several alternative designs as well as a range of control structures and processes which can be used to regulate the form and content of internal representations learned by such networks are examined. Empirical results from the study of some generative learning algorithms are briefly summarized and several extensions and refinements of such algorithms, and directions for future research are outlined.

1. Introduction

Pattern recognition, and learning to recognize patterns, are among the most central attributes of an intelligent entity. *Learning*, defined informally, is the process that enables a system to absorb *information* from its environment. Central to the more powerful types of learning is the ability to construct appropriate internal representations of the environment in which the learner operates. Learning must build the internal representations that perception and cognition utilize.

This research was partially supported by the Air Force Office of Scientific Research (grant AFOSR-89-0178), the National Science Foundation (grant CCR-8720060), the University of Wisconsin Graduate School, and the Iowa State University College of Liberal Arts and Sciences. A preliminary draft of this paper was published as technical report 91-2, Iowa State University Department of Computer Science. The current version appears in Information Sciences (Special issue on Artificial Intelligence and Neural Networks). The authors are grateful to the anonymous referees and Professor Kak for their suggestions on an earlier draft.

Pattern-directed inductive learning (see below) in massively parallel networks of relatively simple computing elements is the focus of this paper which is organized as follows:

[1] Section 1 enumerates some of the desiderata of learning systems to motivate the structures and processes developed in the following sections; and defines some essential terminology used in the discussion to follow.

[2] Section 2 summarizes briefly the two dominant research paradigms in computational approaches to learning - *symbol processing* (SP) systems (Newell, 1980) and *connectionist network* (CN) models (Feldman & Ballard, 1982; Rumelhart, Hinton, & Williams, 1986) and points out some potentially interesting extensions to the currently popular CN models, leading to what may be characterized, for lack of a better term, as *generalized connectionist networks* (Honavar & Uhr, 1990c).

[3] Section 3 motivates a new class of learning algorithms - *generative learning algorithms* (Honavar & Uhr, 1988; 1989a; 1989b; 1990b) which attempt to overcome some of the limitations of CN approaches that rely on the modification of weights within an a-priori chosen, fixed, network topology.

[4] Section 4 discusses a range of generative learning algorithms, briefly pointing out as appropriate, particular designs that have been empirically investigated.

[5] Section 5 briefly summarizes some of the results from experimental investigation of generative learning algorithms. (Detailed discussion of simulation of particular designs and the results of such simulation is beyond the scope of this paper and is readily available in the literature (Honavar & Uhr, 1988; 1989a; 1989b; 1990b; Fahlman & Lebiere, 1990; Yang & Honavar, 1991)).

[6] Section 6 discusses generative learning algorithms as constrained search through a space of internal representations of the environment, presents a summary of key ideas introduced in the paper, and outlines some promising directions for future research in this area.

1.1. Desiderata of Learning Systems

The learning structures and processes developed in this paper are motivated by considerations such as the following:

[1] Rapid learning and the ability to adapt to changes in the environment.

[2] Robustness in the presence of noisy or misleading data (e.g., by using a large number of observations or samples, and/or by being able to undo mistakes resulting from poor data).

[3] Ability to construct efficient internal representations of the environment.

[4] Resolution of the stability-plasticity dilemma (Grossberg, 1980), so as to be able to modify internal representations in response to changes in its environment and the tasks to be performed with minimal interference with performance on previously learned

tasks.

1.2. Pattern-Directed Inductive Learning: Some Definitions and Issues

Most connectionist learning networks that have been developed to date can be broadly characterized as inductive learners. Inductive learning typically involves learning to assign patterns to appropriate classes or categories. It is worth defining some of the terminology associated with inductive learning since pattern-directed inductive learning in massively parallel networks of relatively simple computing elements forms the focus of this paper.

Patterns - 0-dimensional, 1-dimensional, .., ϕ-dimensional

Patterns are typically specified in terms of a list of attribute-values. For reasons which will become clear later, we call these patterns 0-dimensional and denote them by vectors of attribute-values. Thus, a 0-dimensional pattern $\mathbf{X}^0$ may be specified by

$$\mathbf{X}^0 = [x_{k\,;}\ \ 1 \le k \le \nu\ ]$$

where ν is the number of attributes that specify $\mathbf{X}^0$.

Often, patterns embody some inherent spatial, temporal, or spatio-temporal structure. The scheme used for the specification of such patterns should be rich enough to preserve such structure (e.g., temporal ordering that may be implicit in a sequence of measurements of a set of attributes over time, spatial relationships implicit in a visual image). Thus, a 1-dimensional pattern $\mathbf{X}^1$ is a linearly-ordered sequence of 0-dimensional patterns:

$$\mathbf{X}^1 = [\mathbf{X}^0(i\,);\ \ 1 \le i\ \le R_I\ ]$$

where R_I is the number of measurements in the sequence (e.g., temporal resolution if the sequence is over time) and

$$\mathbf{X}^0(i\,) = [x_k\,(i\,);\ \ 1 \le k\ \le \nu\ ]$$

By analogy, a 2-dimensional pattern $\mathbf{X}^2$ is a 2-dimensional array of 0-dimensional patterns:

$$\mathbf{X}^2 = [\mathbf{X}^0(i\,,j\,);\ \ 1 \le i\ \le R_{I\,;}\ \ 1 \le j\ \le R_J\ ]$$

where R_I and R_J are the spatial resolutions along the dimensions indexed by i and j respectively. More explicitly,

$$\mathbf{X}^2 = \begin{bmatrix} \mathbf{X}^0(1,1) & \cdot\ \cdot & \mathbf{X}^0(1,R_J\,) \\ . & .\ . & . \\ . & .\ . & . \\ \mathbf{X}^0(R_I\,,1) & \cdot\ \cdot & \mathbf{X}^0(R_I\,,R_J\,) \end{bmatrix}$$

It is a simple matter to extend this scheme for specification of patterns $\mathbf{X}^\phi$ of any arbitrary dimension ϕ.

It is worth pointing out that this scheme for specification of arbitrary spatial, temporal, or spatio-temporal patterns is far more expressive than that typically used (i.e., a feature vector $\mathbf{X}^0$) in both connectionist as well as symbol-processing approaches to machine learning: it implicitly encodes spatial, temporal, or spatio-temporal distribution of attribute-value measurements into *iconic* or picture-like representations. This representational commitment can have important consequences to the design and performance of learning structures and processes.

Classes, Categories, or Responses Partition the Universe of Patterns

Let **U** denote the set of patterns (the *universe*) of patterns associated with the environment in which the perceiver-learner operates.

In practice, the environment in which a perceiver-learner operates may present it with a subset of **U**, $\mathbf{U}_a$. In many practical situations (e.g., perceptual learning, categorization) it is impossible to know the exact value for $|\mathbf{U}_a|$ for obvious reasons ($|\mathbf{U}|$ denotes the *cardinality* of the set **U**). Thus, the learner must have the ability to accomodate new categories whenever needed (e.g., when it encounters a new class of objects in its environment).

Classes or *categories* are subsets of the universe **U**. Let **C** denote the set of all such *classes*. Then

$$|\mathbf{C}| \leq \sum_{r=0}^{r=|\mathbf{U}|} \binom{|\mathbf{U}|}{r}$$

In practice, only a small subset of **C**, $\mathbf{C}_a$ may be relevant to the task that a perceiver-learner has to perform.

Pattern Recognition Involves Making an Appropriate Response or Class-Assignment to a Sampled Pattern

We define *pattern recognition* as the task of producing a particular response or class assignment C (where $C \in \mathbf{C}_a$) to each pattern **X** (where $\mathbf{X} \in \mathbf{U}_a$) that the perceiver-learner samples from its environment.

The universe of 0-dimensional patterns defined by ν attributes, each of which can take on one of β possible values, contains β^ν patterns. More generally, the universe of ϕ-dimensional patterns - $\mathbf{U}^\phi$ - defined by ν attributes each capable of taking one of β values and with resolution R_i along the ith dimension ($1 \leq i \leq \phi$) contains $|\mathbf{U}^\phi|$ patterns where

$$|\mathbf{U}^\phi| = \beta^{\nu(\prod_{i=1}^{i=\phi} R_i)}$$

Thus, for a universe of 2-dimensional visual patterns encoded by an $R_0 \times R_0$ binary input array, the upper bound on the number of patterns in **U** is $2^{R_0^2}$. That is,

$$|\mathbf{U}^2| \leq 2^{R_0^2}$$

The worst-case complexity of recognition is clearly NP-complete (Garey & Johnson, 1979;

Tsotsos, 1990). But in practice, the patterns in the universe of interest $\mathbf{U}_a$ often exhibit regularities (e.g., connectedness, compactness etc.) which, if judiciously exploited by the recognizer, can potentially make the expected-case complexity of recognition of patterns in $\mathbf{U}_a$ computationally tractable (Tsotsos, 1990).

Inductive Learning Defined

The task of an inductive learner is to learn the set of *mappings* $\mathbf{\Psi}$ that can assign each pattern $\mathbf{X}$ drawn from the universe $\mathbf{U}_a$ into one or more appropriate class(es) drawn from $\mathbf{C}_a$.

Of particular interest is the special case (typically assumed in many machine learning tasks) in which the elements of C_a are mutually *disjoint* classes. Also of interest are particular structures that might be induced over $\mathbf{C}_a$: the classes may be organized in a tree-like *hierarchy* in which the classes at any given level are disjoint (as in conceptual clustering (Michalski & Stepp, 1983)); or they may form a more complex *heterarchy*.

The set of classes C_a may be determined by an external agent (say, the teacher as in perceptron (Rosenblatt, 1962)), or it may be adaptively developed by the learner using some internal criteria for grouping patterns into classes - e.g., some metric of similarity between patterns as in e.g., ART networks (Grossberg, 1976a; 1976b), ID3 algorithm (Quinlan, 1986) and concept learning systems e.g., CLS (Hunt, 1962), COBWEB (Fisher, 1987).

Inductive learning is often guided by *feedback* from the teacher (or from the environment). Such feedback is used to modify $\mathbf{\Psi}$ as necessary.

A Typical Inductive Learning Scenario

Typically, only a subset (and often, a relatively small subset) of $\mathbf{U}_a$, viz. a *training set* $\mathbf{U}_a^{Train}$ is sampled by the by the system during learning. The functions $\mathbf{\Psi}$ that map from the set of patterns $\mathbf{U}_a$ into the set of classes $\mathbf{C}_a$ are induced from the sampled training set $\mathbf{U}_a^{Train}$.

In cases where an external agent provides the *correct* class assignment, it is possible to define a performance measure $P_a^{Train}(\mathbf{\Psi})$ which is the subset $\mathbf{U}_{a,Correct}^{Train}$ of the training set $\mathbf{U}_a^{Train}$ that is given correct class assignments by the system using a particular $\mathbf{\Psi}$. Thus,

$$P_a^{Train}(\mathbf{\Psi}) = \frac{|\mathbf{U}_{a,Correct}^{Train}|}{|\mathbf{U}_a^{Train}|}$$

The performance of such a system on categorizing a subset of patterns in $\mathbf{U}_a$, that it has never encountered before, viz. a *test set* $\mathbf{U}_a^{Test}$ is a measure of its ability to *generalize*. We have:

$$\mathbf{U}_a^{Test} \subset \mathbf{U}_a - \mathbf{U}_a^{Train}$$

$$P_a^{Test}(\mathbf{\Psi}) = \frac{|\mathbf{U}_{a,Correct}^{Test}|}{|\mathbf{U}_a^{Test}|}$$

Note that there are situations in which $P_a^{Test}(\mathbf{\Psi})$ alone is insufficient to evaluate the merit of the system. (Poor performance on the test set might be a consequence of the choice of a poor or inadequate training set).

Several variations on the above scenario are possible e.g., the teacher may provide sample patterns of particular classes, in some desired order (e.g., starting with the *simpler* patterns or classes and gradually introducing patterns or classes of increasing complexity).

The combinatorial space which an inductive learner has to contend with is intractable in the worst case (Gold, 1967). The characterization of mappings that are *feasibly* learnable has become an important area of research in machine learning (e.g., Valiant, 1984). As noted earlier, the universe of ϕ-dimensional patterns defined by ν attributes each capable of taking one of β values and with resolution R_i along the ith dimension ($1 \le i \le \phi$) contains $|\mathbf{U}^{\phi}|$ patterns where

$$|\mathbf{U}^{\phi}| = \beta^{\nu(\prod_{i=1}^{i=\phi} R_i)}$$

Even in one of the simplest cases possible, i.e., where we have only 2 classes, the number of different ways of assigning $\mathbf{U}^{\phi}$ patterns into 2 classes is given by $\sum_{i=0}^{i=|\mathbf{U}^{\phi}|} \binom{|\mathbf{U}^{\phi}|}{i}$. The problem of discovering the *correct* $\mathbf{\Psi}$ that assigns each of the patterns to their respective classes by exhaustively examining all possible class assignments is clearly intractable.

The major lesson from these complexity results is that an inductive learner, in order to be effective, has to explore only the *useful* regions of the combinatorial space. In order to do so, it has to efficiently absorb the *information* in the *training set*. It also should be constrained, by feedback and its structures and processes, to learn a useful $\mathbf{\Psi}$ to map the patterns in its sub-universe $\mathbf{U}_a$ into the appropriate classes (or network responses), given its limited resources, and the tasks it has to perform. The learning structures and processes developed in this paper constitute steps in this direction.

2. Approaches to Pattern-Directed Inductive Learning

Two of the dominant research paradigms in computational approaches to learning are *symbol processing systems* (Newell, 1980) and *connectionist networks* (Rosenblatt, 1958; Feldman & Ballard, 1982; Rumelhart, Hinton, & McClelland, 1986).

Learning in SP systems (e.g., SOAR (Laird, Newell, & Rosenbloom, 1987)) typically involves modification of stored data structures, rules, or procedures. Space does not permit a review of current work in machine learning within the SP paradigm (Carbonell, Michalski, & Mitchell, 1983; Dietterich & Michalski, 1983; Michalski & Kodratoff, 1990 provide such reviews). Many learning techniques based on symbolic representations suffer from high sensitivity to noisy data (primarily because of the use of rigid and inflexible inference and categorization strategies). Recent years have seen some tentative moves toward the use of probabilistic or fuzzy inference mechanisms to alleviate the problem of noise-sensitivity and brittleness of SP approaches to machine learning (Michalski & Kodratoff, 1990; Aha &

Kibler, 1989).

Another source of difficulty for computational approaches to learning is the choice of inappropriate abstract knowledge representations (e.g., a feature vector representation of data available in a road map where an *iconic* (i.e., picture-like) representation would have been much more appropriate), in order to make use of the available learning algorithms. It is well-known that the knowledge representation chosen influences the ease and the *computational complexity* of learning: e.g., Boolean concepts expressed in k-term DNF (disjunctions of at most k pure conjuncts, each with an unlimited number of literals) are not feasibly learnable whereas concepts expressed in k-CNF (*conjunctive normal form* with at most k literals per conjunct) are (Valiant, 1984). Thus we need a range of learning algorithms and architectures that exploit the strengths of available representations and/or efficiently transform between representations as necessary.

Networks of relatively simple computing elements such as CN offer an attractive and versatile framework for exploring a variety of learning structures and processes for intelligent systems for a variety of reasons (e.g., massive parallelism of computation, potential for fault and noise-tolerance, etc.). The remainder of this section describes CN, and some extensions to CN leading to what may be called (for lack of a better term), *generalized connectionist networks* (GCN) (Honavar & Uhr, 1990c). GCN offer a number of potentially powerful alternative learning mechanisms including *generative learning* which is discussed in detail in this paper.

2.1. Connectionist Networks (CN)

A connectionist network (Rosenblatt, 1958; Feldman & Ballard, 1982; Rumelhart, Hinton, & McClelland, 1986) consists of a directed graph whose nodes compute relatively simple functions over the inputs they receive from other nodes in the network via their weighted input links (or from the external environment) and transmit the results to other nodes (or the environment) via their weighted output links. A CN is typically specified in terms of the function computed by the individual nodes, the topology of the graph linking the nodes, and (if the network is adaptive), the algorithm used to modify the weights on the links. The network structures that implement the learning algorithm, the control structures that are necessary to perform a variety of functions (e.g., synchronizing the nodes, to switch on the learning processes, etc.) are generally left unspecified.

Each node computes a single scalar output value s that is some simple function of its (numeric-valued) inputs. A node n_j with K_j inputs has a K-dimensional *weight vector*

$$\mathbf{W}_j = [w_{j,1}, w_{j,2}, \cdots, w_{j,K_j}]$$

One weight is associated with each input link. Let the input to the node n_j be

$$\mathbf{I}_j = [i_1, i_2, ..., i_{K_j}]$$

Each node n_j has associated with it, a *node function* $\mathbf{T}_j$ which represents one or more computational steps involved in the calculation of the output of the node s_j. Some examples of such calculations are shown in figures 1-A and 1-B.

Some form of nonlinearity is necessary for making decisions necessary for categorization (Nilsson, 1965). The threshold function is a discrete decision-making device. It classifies all its input patterns into two sharply distinguished sets. The logistic function is a continuous version of the threshold function. Its graded response is attractive for a variety of purposes - including noise tolerance, contrast enhancement, and the existence of a derivative over the entire range of the function (a property required by some learning algorithms e.g., *generalized delta rule* (Rumelhart, Hinton, & Williams, 1986) for error back-propagation).

Often it is necessary to compare an input pattern with a stored *template*. Various versions of the *match* function are occasionally used to accomplish this task. A particular class of such functions is shown in figure 1-B. Such functions are attractive for a variety of reasons: The output is maximum when there is a perfect match between the stored weight vector and the input and monotonically decreases with increasing mismatch. The rate of decrease in response is governed by the functional form of f_j and can be tuned by varying α_j. The explicit measure of mismatch between $\mathbf{I}_j$ and $\mathbf{W}_j$ can be used to design a variety of

---

Let $w_{j,0}$ be the *bias* associated with the node n_j. It is customary to treat the bias as if it is a weight associated with a constant input $i_0 = -1$.

$$u_j = \sum_{k=0}^{K_j} w_{j,k} \times i_k$$

The output of the node n_j is given by

$$s_j = f_j(u_j)$$

Some commonly used f_j define the linear, threshold and logistic node functions:

$$f_j(u_j) = u_j \qquad \textit{linear function}$$

$$f_j(u_j) = \begin{cases} 0 & u_j < 0 \\ 1 & \textit{otherwise} \end{cases} \qquad \textit{threshold function}$$

$$f_j(u_j) = \frac{1}{1 + e^{-u_j}} \qquad \textit{logistic function}$$

Figure 1-A: Linear, Threshold, and Logistic Functions

---

learning algorithms (see below).

Learning in CN Modifies the Weights on the Links

Most of the work on learning in feed-forward CN involves the induction of functions $\mathbf{\Psi}$ that map the CN's input patterns to desired outputs. This is accomplished by modifying the *weight vectors* associated with each of the nodes linked in an a-priori fixed topology (Hinton, 1989). Some weight-modification schemes are based on correlations in the activation values associated with connected nodes based on the mechanism proposed by Hebb (1949). Others use various estimates of error between the desired and actual network outputs. Error estimates may be based on extremely specific feedback e.g., the desired network output provided by the teacher for each input pattern which is used in the *perceptron* algorithm (Rosenblatt, 1958) and in the error backpropagation algorithm (Rumelhart, Hinton & Williams, 1986; Werbos, 1974) and some of its faster variants (Fahlman, 1988; Balakrishnan & Honavar, 1992). Alternatively, they may be obtained from not-so-specific feedback e.g., a reward/punishment signal used in reinforcement learning (Barto & Anandan, 1985), or they may be internally derived, e.g., based on an estimate of the output necessary from a node to

---

The output of the node n_j is given by

$$s_j = f_j(u_j)$$

Some typical choices for u_j are:

$$u_j = \frac{\left(\sum_{k=0}^{K_i} |(w_{j,k} - i_k)|^r\right)^{1/r}}{\alpha_j \times \sigma_j{}^r}$$

where r is a positive integer, $\sigma_j{}^r$ is a suitable normalization term, and α_j is a tunable parameter. It is straightforward to specialize this expression to yield *normalized euclidian distance* between the vectors $\mathbf{I}_j$ and $\mathbf{W}_j$ or the *Hamming distance* if the vectors are binary. Some typical choices for f_j are:

$$f_j(u_j) = e^{-u_j{}^q}$$

where q is a positive integer, or, alternatively,

$$f_j(u_j) = \frac{1}{1+u_j}$$

These functions belong to the family of *radial basis functions*.

Figure 1-B: Various Match Functions

---

produce the overall desired behavior from the network, e.g., in competitive learning (Grossberg, 1987).

Consider a CN node n_j which receives one of it's inputs i_k, from the node n_k. The feedback to the node is t_j, an estimate of the desired output of the node n_k. The current output of the node is s_j. A typical rule for modifying the weight $w_{j,k}$ takes the form:

$$\Delta w_{j,k} = \eta \times \lambda(i_k, w_{j,k}, s_j, t_j)$$

where η is a constant of proportionality called the learning rate, and λ is the function used to compute the weight modification as a function of the error between the feedback and current output. Some of the most popular weight modification schemes of this form are the Perceptron algorithm for a network with one layer of modifiable links (Rosenblatt, 1958) and its generalization to a network with multiple layers of modifiable links (Werbos, 1974; Rumelhart, Hinton, & Williams, 1986).

A major limitation of the single-layer perceptron is its inability to learn to correctly partition a universe of patterns that is not *linearly separable* (Nilsson, 1965; Minsky & Papert, 1969). This is primarily due to the limited *representational power* of single-layer perceptrons. Multi-layer architectures can potentially overcome this limitation given an appropriate set of learning processes (Nilsson, 1965) and indeed multi-layer networks that learn using the generalized delta rule (Rumelhart, Hinton, & Williams, 1986) demonstrate this fact. It has been shown that the problem of determining an appropriate set of weights (that map a given set of inputs into desired outputs) within an a-priori chosen network topology (assuming that such weights exist) - the so-called *loading problem* is NP-complete (Judd, 1990). It appears reasonable to conjecture that the weight-modification schemes used in CN suffer from a variant of the loading problem. This difficulty is compounded by problems that plague gradient descent strategies (in particular, *local minima*). Little is known about how to overcome such difficulties in multi-layer CN learning algorithms.

Recent results in computational learning theory (Valiant, 1984; Judd, 1990; Natarajan, 1991) suggest that in order to make learning computationally tractable, it is almost always necessary to judiciously restrict the input/output mappings by incorporating suitable *representational* and *inductive* biases (Mitchell, 1980) into the system. The empirical question as to whether *particular* CN architectures can *efficiently* learn to effectively perform *particular* tasks (e.g., pattern recognition) remains largely open. Generative learning algorithms that adaptively acquire new representational primitives as a function of experience (see below) offer a range of alternatives for overcoming the limitations of single-layer perceptrons.

2.2. Generalized Connectionist Networks (GCN)

The following rather general definition (Honavar & Uhr, 1990c; Uhr, 1990a) makes clear that today's CN architectures can be extended in many potentially useful ways.

A GCN is a graph (of linked nodes) with a particular topology $\mathbf{\Gamma}$. The total graph can be partitioned into three functional sub-graphs - $\mathbf{\Gamma_B}$ (the *behave/act* sub-graph), $\mathbf{\Gamma_\Lambda}$ (the *evolve/learn* sub-graph), and $\mathbf{\Gamma_K}$ (the *coordinate/control* sub-graph). The motivation for distinguishing among these three functions will become clear later. The nodes in a GCN compute one or more different type/s of functions: **B** (*behave/act,*); $\mathbf{\Lambda}$ (*evolve/learn,*); and **K** (*coordinate/control*).

$$\text{GCN} = \{\mathbf{\Gamma}, \mathbf{B}, \mathbf{\Lambda}, \mathbf{K}\}$$

Today's CN are specified (typically only partially) as follows:

* The overall graph structure, $\mathbf{\Gamma_B}$ of the sub-net that behaves (today much of the total graph, including the entire sub-graphs needed to handle learning and control - is usually left unspecified). A complete description of the entire graph $\mathbf{\Gamma}$ is necessary to completely specify CN realizations of such architectures.
* The node function $\mathbf{T}_k$ and the weight vector $\mathbf{W}_j$ associated with each node n_j that define the functions computed during the behave cycle. Typically, the same node function is used with each of the nodes in the network.
* The single function $\boldsymbol{\lambda}$ that is used to compute the changes to the weight vectors during the learning cycle - but not the actual sub-net structures that are needed to actually compute and make these changes.

A network's behavior is typically initiated by sending values to the input sensing nodes. Its resulting performance is the set of values sent by its output acting nodes. The net's behavior is completely determined by its topology, the values originally associated with its links, and the functions computed by its nodes and links - plus any modifications made to these by learning.

Several potentially powerful extensions to today's CN - including more powerful structures and processes for behaving, control, and learning - suggest themselves, leading up to systems that can be characterized as *generalized connectionist networks* (GCN) or *generalized neuromorphic systems* (GNS) (Honavar & Uhr, 1990c). Our focus here is on learning structures and processes.

Toward More Powerful Learning Structures and Processes

Learning in today's CN is almost always handled by processes that change the weight-vector $\mathbf{W}_k$ associated with each node n_k so as to reduce the error between the desired output from the node and its actual output. There is no compelling reason to restrict ourselves to this set of learning processes. There are a variety of potentially more powerful alternatives:

[1] Learning that modifies the node functions $\mathbf{T}_k$ associated with the processing elements in the GCN: Learning might alter the steepness of the sigmoid function (Tawel, 1989); equivalently, if a threshold element is used, learning might involve systematically ranging through a number of alternative node functions (e.g., from logical OR to logical AND) by changing that threshold.

[2] Learning that modifies the *weight matrices* or *local templates* associated with each processing element in the GCN: This might involve the use of one of the several weight-modification strategies currently used in CN (e.g., Widrow & Hoff, 1960) or other forms of weight modification that may be appropriate for the corresponding node functions used (e.g., if a node that matches its input with a stored weight matrix and produces a measure of match - e.g., a gaussian match node - is used, a suitable weight modification strategy might be to blur the weight matrix by adding a small fraction of a sufficiently well-matched input).

[3] Learning that modifies the connectivity of the network $\mathbf{\Gamma}$ (i.e., by the addition (and when indicated, or necessary to make space, deletion) of links and nodes) viz., *generative learning* (Honavar & Uhr, 1988; 1989a; 1989b) and related approaches (Ash, 1989; Diederich, 1988; Fahlman & Lebiere, 1990; Hanson, 1990; Gallant, 1990; Nadal, 1989; Rujan & Marchand, 1989). Given a suitable set of generative learning mechanisms, a GCN can adaptively search for and assume whatever connectivity is appropriate for particular tasks (possibly under certain predefined topological constraints - e.g., those imposed by locally-linked multi-layer converging-diverging networks (Honavar & Uhr, 1988; 1989a).

[4] Learning that modifies the control structures and processes $\mathbf{K}$ that are used in GCN: In particular, learning might alter the controls that regulate particular types of learning (e.g., parameters such as the learning rate used in weight modification); or the initiation and termination of *plasticity*, or the manner in which different sorts of learning (e.g., weight modification and generative learning) are coordinated. We will examine some examples of such mechanisms later.

[5] Learning that modifies the learning structures and processes ($\mathbf{\Lambda}$) themselves, e.g., changes in weight modification strategies, changes in node generation strategies, and so on.

3. Motivations for the Study of Generative Learning in GCN

This section motivates *generative learning* - which enables GCN to adaptively acquire the necessary network connectivity and the functions $\mathbf{\Psi}$ - that are necessary for effectively classifying the patterns in its universe - by recruiting nodes and growing links as needed.

Adaptive Determination of Network Connectivity

If a CN has to learn the necessary $\mathbf{\Psi}$ entirely through changing the weights of its links (which is by far the most commonly used learning mechanism), it must be initialized to contain *a sufficient number of appropriately linked nodes.* The only way to guarantee that this kind of network has enough nodes, each with the necessary links, is either to provide all of them in in advance, using a priori knowledge about $\mathbf{U}_a$, or to make some guess as to the necessary number - and use a substantially larger number of nodes and links than that to be on the safe side. The only way to be completely safe is to use worst-case estimates; but this is extremely impractical because many problems that such networks have to handle (e.g., real-time pattern recognition) are NP-complete (see Minsky & Papert 1969 for a discussion of the time versus memory complexity of pattern matching).

Generative learning, which enables a GCN to modify its own topology (nodes and connections), appears to offer a partial solution out of the difficult problem of choosing a fixed network connectivity. Given mechanisms to generate, the network can gradually grow toward a sufficient topology, under the implicit guidance of the input patterns and feedback from the environment, as well as the network's learning structures and processes, until it has the necessary number of nodes and links.

Rapid Learning and Generalization

A CN that learns the necessary $\boldsymbol{\Psi}$ by changing weights within a network of fixed size and connectivity has to solve the intractable *loading problem* (Judd, 1990). A learning algorithm that simply adds new units as necessary can rapidly build a network that is essentially a random-access memory or a look-up table to represent any arbitrary set of functions (Baum, 1989). But unfortunately, such a look-up table has to store each sample pattern and is thus inefficient in its use of memory (Minsky & Papert, 1969). Furthermore, it is incapable of generalizing correctly to sample patterns that are not stored in the look-up table. Thus, such networks have to be supplemented by mechanisms that enable the network to generalize correctly to future sample patterns.

There are a number of reasons for using CN with nodes that have limited fan-in (i.e., each node receives inputs from a small subset of the nodes in the layer below - see Honavar & Uhr, 1988; 1989a for an example of such networks and the motivations for such restricted connectivity). However, when the fan-in is limited in this manner, it becomes necessary to use multiple layers to enable the computation of complex functions over the entire input pattern. But as the networks get deeper, weight modification schemes such as error backpropagation perform poorly because the the error signal gets weaker as it reaches layers further from the output nodes (Sandon, 1987). Generative learning can potentially offer a way around this problem by constructively building up multi-layer networks.

CN that learn through weight modification can potentially support generalization (Rumelhart, Hinton, & Williams, 1986) - essentially because they are function approximators. However, it has been observed that CN that have far too many nodes in excess of the number actually needed for a learning task often exhibit poor generalization (in cases where they learn to *overfit* the training set) (Rumelhart, 1988).

These considerations suggest that generative learning algorithms that combine the ability to add nodes and links only when needed with the ability to change weights, might yield networks that learn rapidly without sacrificing the ability to generalize correctly to future sample patterns. (Contrast this with schemes that start with a large network and discard links to improve generalization (Le Cun, Denker, & Solla, 1990; Mozer & Smolensky, 1989; Hanson & Pratt, 1989)).

Constructive Estimation of Expected-case Complexity

If successful, generative learning can provide us with a constructive, empirical estimate of the expected-case complexity of the task e.g., perceiving and learning to perceive everyday objects - estimates which are extremely difficult to obtain by any other means (because such tasks are usually ill-defined).

Neurobiological Considerations

Animal learning appears to involve the growth of new synapses throughout life in addition to the tuning of synaptic strengths (Greenough & Bailey, 1988). GCN that combine generative learning with tuning of node functions (e.g., the weight matrices that encode synaptic strengths) therefore appear to provide a potentially useful framework for simplified models of brain development (Honavar, 1989).

Integration of Symbolic and Sub-Symbolic Approaches to Learning

Generative learning provides mechanisms for rapid construction of distributed look-up tables through extraction, abstraction, and generalization of potentially useful input patterns (see below). Such mechanisms have obvious commonalities with processes such as *chunking* (Laird, Newell, & Rosenbloom, 1987) and exemplar-based knowledge acquisition (Bareiss, 1988) used in SP systems. This, combined with the ability to fine-tune the acquired knowledge structures using weight modification strategies of the sort used in CN, provides a basis for integration of symbolic and sub-symbolic approaches to machine learning in GCN. Symbol structures thus constructed through generative learning are *grounded* in respective sensory representations.

4. Alternative Designs for Generative Learning

In its most general form, generative learning can be viewed as the process of acquisition of masks, partial templates, or weight-matrices - which in some form encode potentially useful information in the input patterns, generating (creating or recruiting from a pool of uncommitted nodes) a node to instantiate each function as it is acquired, and incorporating the generated node into the network.

In what follows, we assume that generation encodes information in the form of weight matrices. It is a simple matter to extend this to encodings in the form of non-numeric patterns (e.g., symbol-structures) either by encoding the non-numeric patterns into a numeric (e.g., binary code) or by using nodes that match non-numeric patterns (or pattern abstractions such as trees, graphs, etc.) using a variety of similarity metrics (Honavar, 1992a; 1992b).

The Basic Generative Learning Algorithm

A generative learning scheme is shown in a skeletal form in figure 2. Its definition suggests that generative learning requires the following sub-tasks to be performed by (preferably local) microcircuits in GCN:

[1] Deciding where to generate a new node (in multi-layer networks);

[2] Deciding when to generate a new node (as opposed to say, continued modification of existing nodes);

[3] Choosing a pattern $\mathbf{X}_k$ or sub-pattern to be encoded by the generated node n_k in its weight matrix $\mathbf{W}_k$ (this includes choosing the inputs to a generated node);

[4] Choosing the nodes that receive the output s_k from the node n_k (as one of their inputs) and the corresponding weights;

[5] Evaluating nodes as potential candidates for generation (and possibly, degeneration);

[6] Choosing a weight modification strategy that is appropriate for the node functions used, and for the particular choices of generation strategy.

Different design decisions on how each of these subtasks are accomplished in a GCN lead to different variants of the generative learning strategy.

4.1. Choosing a Pattern Vector to Encode by Generation

Several alternative strategies may be used to choose a pattern $\mathbf{X}_k$ to be associated with the generated node n_k.

Choosing a Pattern Vector at Random

A weight vector $\mathbf{W}_k$ may be chosen for the node n_k at random (thereby encoding a random pattern vector). $\mathbf{W}_k$ may then be modified using one of the available weight modification strategies (e.g., error back-propagation). An example of this approach is the *dynamic node creation* algorithm (Ash, 1989) which adds *hidden* nodes one at a time to a 3-layer network of sigmoid nodes.

Choosing a Pattern Vector by Mutation or Crossover

The network may be initialized with nodes generated to encode pattern vectors at random as described above, or to encode a small subset of prespecified patterns. Additional patterns to be encoded are then obtained by applying *mutation*, *crossover*, or *inversion* operations akin to the ones used in evolutionary learning algorithms (Fogel, Owens, & Walsh, 1966; Holland, 1975; Koza, 1990) to the pattern vectors already encoded by the network. This has clear parallels in biological evolution and in the genesis of the immune system in animals (Jerne, 1967; Farmer, Packard, & Perelson, 1986).

Choosing a Pattern Matrix by Extraction

The input patterns that the network sees during its learning experience carry potentially useful information. Thus, it is desirable to generate nodes that have their weight-matrices and response functions chosen such that a generated node is tuned to respond optimally to some sub-pattern of its input pattern (or some abstraction of such a pattern - represented by the outputs of some nodes in the network). This can be accomplished through *generation by extraction*, i.e., the process of detecting a potentially informative sub-pattern $\mathbf{X}_E$ in a

```
while some performance criterion P is not met do
    {
    sample a stimulus pattern X^0 from U_a^Train;
    compute the network output S;
    .
    .
    if there is reason to modify the network - e.g., error unacceptable
        {
        .
        .
        while the criteria for generation are satisfied do
            {
            Evaluate candidate patterns to be encoded;
            Choose one or more patterns to encode;
            Add the node(s) that encode the chosen pattern(s) to the network;
            Initialize the weight matrices associated with the generated nodes;
            Establish the desired output connectivity from the added nodes
            .
            }
        .
        .
        }
    .
    .
    }
```

Figure 2: A high-level outline of a generative learning algorithm The performance criterion P is evaluated by periodically testing the network on patterns sampled from $\mathbf{U}_a$. A number of essential details - e.g., criteria for generation; choosing a suitable pattern to be encoded by a generated node; initialization of the weight matrix associated with the generated node - are left unspecified.

sampled input pattern $\mathbf{X}$ (or some abstraction of it) and generating a GCN node n_k with a suitable weight matrix $\mathbf{W}_k$.

Generation by extraction can be illustrated by considering a network of *gaussian match* (or *radial basis function*) nodes. The output s_k of such a node with weight matrix $\mathbf{W}_k$ is given by

$$s_k = e^{-D(\mathbf{X},\mathbf{W}_k)}$$

where $D(\mathbf{X},\mathbf{W}_k)$ is a measure of mismatch between $\mathbf{X}$ and $\mathbf{W}_k$. Suppose the sampled input is a binary pattern $\mathbf{X}_S = [x_S(i);1 \le i \le \nu]$ where $x(i) \in [0,1]$; and the extracted sub-pattern is is $\mathbf{X}_E = [x_E(i);1 \le i \le \nu]$ where $x_E(i) = x_S(i)$ OR $x_E(i) = *$ where * denotes *don't care*. For instance, if $\mathbf{X} = [1,0,1,1,0,1,0,1]$, a possible extraction could be $\mathbf{X}_E = \mathbf{W}_k = [*,0,*,1,0,*,*,*]$. Then generation by extraction would recruit a node n_k and set its weight matrix $\mathbf{W}_k = \mathbf{X}_E$. A physical interpretation of * in this case is the absence of a link from the corresponding input node to the generated node n_k. The generated node n_k in this case responds with an output of 1 whenever the network receives an input pattern $\mathbf{X}$ such that x_2=0 AND x_4=1 AND x_5=0 and its output falls monotonically as the mismatch between $\mathbf{W}_k$ and its input $\mathbf{X}$ increases.

The next section shows how the information-content of an extraction can be increased by further refining the process of extraction.

Generation by Extraction of Novel Sub-patterns

An extracted sub-pattern is likely to provide useful information to the system if it is sufficiently *novel*. We can associate with each set of nodes $\mathbf{M}_i$ that receive their inputs from the same *receptive field* $\mathbf{Z}_i$, a *novelty detector-extractor* microcircuit d_i (The receptive field of a node n_k is defined by the set of nodes from which it receives its inputs). Let us call the set of nodes $\mathbf{M}_i$, the *field of influence* of the novelty-detector-extractor microcircuit d_i (see figure 3).

$$\mathbf{M}_i = [m_i(j);\ 1 \le j \le \mu_i]$$

where μ_i is the number of nodes that constitute the field of influence of the novelty-detector-extractor circuit d_i.

Let $\mathbf{X}$ denote the input pattern to the network. Let $\mathbf{X}_i$ denote the sub-pattern of the input (or some abstraction of it) seen by the receptive field $\mathbf{Z}_i$. Let s_j denote the output of m_j in response to $\mathbf{X}_i$. When activated, d_i produces a *generate signal* g_i if the sub-pattern $\mathbf{X}_i$ is sufficiently *novel* (see below). The conditions under which the novelty-detector-extractor circuits are activated are discussed later.

Thus, $g_i = 1$ if a novel sub-pattern is detected by d_i in the receptive field $\mathbf{Z}_i$; otherwise $g_i = 0$. Translating this into a functional specification for d_i we get:

$$g_i = 1 \textbf{ if and only if } \forall j\ s_j \le \tau_i;$$
$$g_i = 0 \textbf{ otherwise}$$

where τ_i is a predefined (or tunable) *novelty threshold*. That is, novelty-detection involves identifying sub-patterns in the current input pattern (or some abstraction of it) for which no node in the network has a sufficiently strong response (i.e., greater than the novelty threshold). Note that this notion of novelty is functionally related to similar ideas used in the ART family of architectures (Grossberg, 1976a; 1976b; Carpenter & Grossberg, 1987).

When a novelty-detector-extractor microcircuit d_i has found a novel sub-pattern in its receptive field $\mathbf{Z}_i$ (i.e., $g_i = 1$), it:

[1] Recruits an uncommitted GCN node n_k from the pool of such nodes available to it locally in the network;

[2] Initializes the receptive field of the node to $\mathbf{Z}_i$ by growing links from each of the nodes in $\mathbf{Z}_i$ to the recruited node n_k.

[3] Adds n_k to the set of nodes $\mathbf{M}_i$;

[4] Initializes its weight matrix $\mathbf{W}_k$ such that it is tuned to respond to the (novel) sub-pattern $\mathbf{X}_i$ (see above) that it has just detected; and

[5] Resets the *generate signal* g_i to 0.

The operation of a novelty detector-extractor microcircuit is illustrated in figure 3.

A network that generates by novelty-driven extraction has to be initialized with novelty-detector extractor microcircuits each with a distinct receptive field that determines its field of influence. Alternatively, the first node to be generated from an as yet un-represented receptive field (no need to look for novel sub-pattern here) should recruit a novelty detector-extractor microcircuit (from an uncommitted pool of such circuits) to perform extraction of novel subpatterns that might appear in its receptive field in the future.

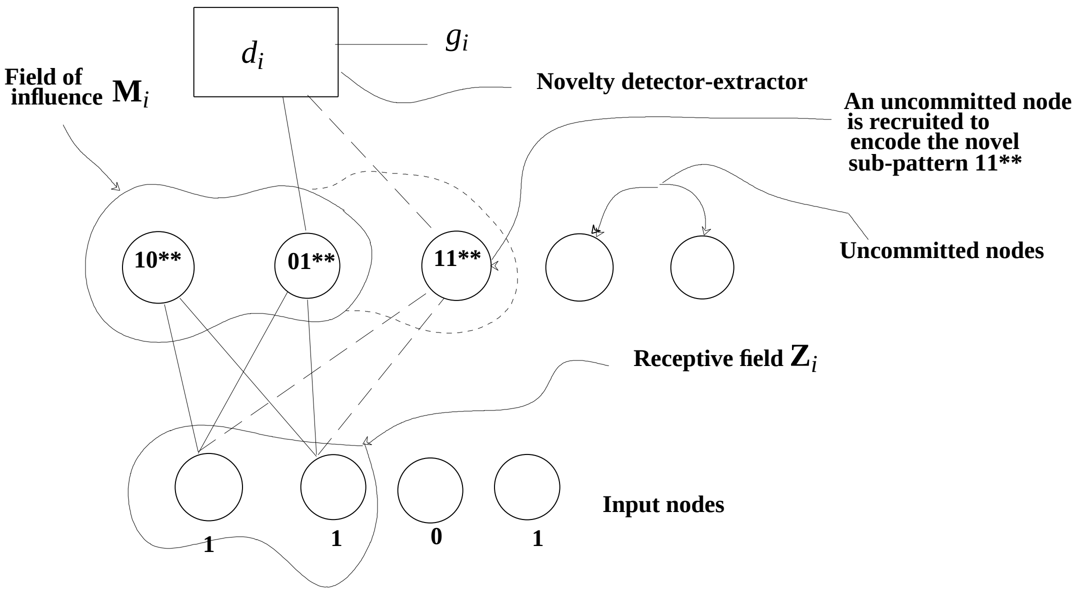

Figure 3: The operation of a novelty detector-extractor microcircuit.

Increasing the Information Content of an Extraction

Generation by extraction of novel subpatterns increases the likelihood that the generated node encodes potentially useful information. A strategy that can be used (either by itself or in conjunction with novelty detection) to increase the information content of an extraction is discussed below.

Instead of choosing one or more novel extraction(s) at random, the system can recruit a pool of *candidates for generation* each encoding a different extraction from the input pattern. (Depending on the locus of generation in the network, the entire pool might be situated at a single network layer or may be constituted by nodes at different layers). The presentation of sample patterns continues as before, and each of the candidates for generation in the pool get trained and evaluated in parallel. The candidates in the pool compete with each other to be added to the network. This procedure allows a phase of evaluation of the information-content (or usefulness) of a potential extraction before it is used in the network. One simple strategy for evaluating the information-content of a potential extraction is to correlate the output of the corresponding node with the residual error which it is intended to reduce; high correlation is indicative of high information-content. Similar versions of this strategy are used by (Honavar & Uhr, 1989; Fahlman & Lebiere, 1990) and empirical studies suggest that learning performance generally improves with increase in size of the candidate pool (Yang & Honavar, 1991).

Alternatively, the evaluation of the information-content of a potential extraction can be based on a measure of *mutual information* (Shannon, 1963) between the output of a candidate for generation and the nodes already in the network; candidate extractions that are most *informative* of the desired network output are the ones encoded by the generated nodes. Other alternatives for selection of a sub-set of candidate nodes for addition to the network are suggested by the techniques for optimal feature sub-set selection developed for statistical pattern classification (see Fukunaga, 1990 for a discussion of such selection algorithms).

Generalizing Extracted Sub-patterns

Generation by Extraction of novel sub-patterns as described above essentially fixes the weight matrix $\mathbf{W}_k$ associated with the generated node n_k using a single extracted sub-pattern. Often, the universe of patterns contains many patterns that contain more or less similar sub-patterns. When such sub-patterns are sufficiently similar (and hence not *novel*) to the ones already encoded by the network, novelty-detector-extractor microcircuits ensure that no new nodes are generated. However, the similarity among sub-patterns can in itself be potentially meaningful. Such similarity can be used to *generalize* (in a limited sense) a previously encoded extraction is outlined below (for discussion of some related work, see Hampson, 1990; Reilly & Cooper, 1990; Rumelhart & Zipser, 1986):

Suppose an activated novelty-detector-extractor microcircuit d_i fails to find a *novel* sub-pattern in its receptive field $\mathbf{Z}_i$ (i.e., $g_i = 0$). In this case, there exist one or more nodes in the set $\mathbf{M}_i$ which have their output above the novelty threshold τ_i. This means that the sub-pattern $\mathbf{X}_i$ appearing in the receptive field $\mathbf{Z}_i$ is sufficiently similar to the sub-pattern encoded by at least one of the nodes - $m_i(j)$ whose outputs s_j are greater than τ_i. A similarity-driven generalizer microcircuit then generalizes the weight matrix associated with such nodes as follows:

$$\mathbf{W}_j = \mathbf{W}_j + \eta \times \mathbf{X}_i$$

where η is a small positive constant. This has the effect of moving the sub-pattern for obtaining optimal response from node $m_i(j)$ a little bit closer to the current input to the node. This is the strategy that has been used for generalization in most of our simulations to date.

Additional schemes for generalization e.g., those based on *surface-fitting* or *interpolation* (Hampson, 1990; Wolpert, 1989a; 1989b) are also worth investigating in this context.

4.2. Establishing the Output Connectivity of a Generated Node

Several alternative designs for choosing the set of nodes that should receive outputs from a generated node are possible. Different design choices for establishing the output connectivity of a node have different implications in terms of how the weights on the output links get initialized and get changed through weight-modification. Some alternative designs are discussed below.

A generated node is linked to each of the nodes in the output layer of the network and initialized with random (or heuristically chosen) weights. This is the design used in recognition cones that combine generation and weight-modification (Honavar & Uhr, 1988, 1989a), the DNC algorithm (Ash, 1989), and the *cascade correlation architecture* (Fahlman & Lebiere, 1990). Alternatively, a generated node is linked selectively to the network output node(s) which are indicated by the feedback provided with the training sample that was used for generation. Several variations on these basic designs are possible.

4.3. Weight-Modification Strategies

The mechanisms for generalizing extracted sub-patterns described above constitute one form of weight modification. A different form of weight-modification strategy is suggested for the links that connect the generated nodes (or, if desired, input nodes), with the output nodes. We have used variants of the simple perceptron rule (Rosenblatt, 1958) or the Widrow-Hoff rule (Widrow & Hoff, 1960) for the modification of weights between the output nodes and other nodes in the network. Note that this does not prevent the network from learning non-linearly seperable class partitions because the input representation is augmented with additional representational primitives that capture non-linear interactions among the inputs through generative learning thereby circumventing the need for back-propagation of error signals through multiple layers.

4.4. Deciding When to Generate

Like the mechanisms that choose the pattern to be encoded by a generated node, the mechanisms that control the timing of generation of nodes are central to the design of generative learning structures. Several alternative designs for micro-circuits that control the decision to generate new nodes in GCN are possible:

[1] Generate nodes at a certain predetermined or tunable rate all the time (*perpetual generation*) along with corresponding de-generation;

[2] Generate only when the feedback indicates that the network produced an incorrect output (*error-driven generation*);

[3] Generate when the feedback differs from the network's output and the performance of the network has been unsatisfactory over a sufficiently long sequence of training samples (*error-driven minimal generation*); and

Of these, we will examine in some detail, *error-driven minimal generation* which has been used in some of the generative learning algorithms that have been examined empirically (Honavar & Uhr, 1988; 1989a; 1990b; Fahlman & Lebiere, 1990).

Error-Driven Minimal Generation

Error-driven minimal generation is motivated by the need to strike a suitable balance between tuning of nodes already present in the network (e.g., by limited generalization of extractions discussed above, or other forms of weight modification) and the generation of new nodes when the network determines that its current topology is incapable of attaining the desired performance. If a network with an inadequate number of nodes is restricted to learning by tuning its weight matrices alone, it may never be able to attain the desired performance. On the other hand, if new nodes are generated indiscriminately, it can result in an unduly large and inefficient network. Thus the network must have effective processes that modify weights and tune the nodes, evaluate nodes for their usefulness, and discard nodes and links that are found useless.

The general strategy used in error-driven minimal generation is as follows: Continue to tune the existing nodes (by weight modification) as long as the network continues to make progress toward the desired performance criterion P; Initiate generation when the improvement in performance has leveled off (see figure 4 for details). This requires microcircuits that keep track of the desired performance measure over a sliding window in time (or a sequence of presentations of a training sample). A particular example of this strategy that is used in recognition cone networks which combine generation with weight modification learning (Honavar & Uhr, 1988; 1989a) is specified in detail in figure 4. In this case, one or more sufficiently novel (see above) extraction(s) is (are) obtained from a misclassified training pattern (or a sequence of such patterns) whenever generation is initiated. Somewhat different versions of this strategy are used in the DNC algorithm (Ash, 1989), and the cascade-correlation algorithm (Fahlman & Lebiere, 1990).

4.5. Deciding Where to Generate

Although the discussion of various aspects of generation did not explicitly address generation in multi-layer networks, all the the details carry over to that case with the exception that extractions at higher layers occur from abstractions of the input pattern instead

t_0 = time (measured in number of samples of class C_i seen by the network) when the last node was extracted from a sample pattern from class C_i.

t = current time.

$P_i(t)$ = the classification accuracy on sample patterns of class C_i at time t.

Δ = time interval over which the performance improvement is measured.

$P_i(T)$ = the desired classification accuracy at the end of learning.

$p_{generate}$ = a parameter that controls the degree of leveling off of the learning curve sufficient to initiate generation.

A generation is initiated from the current sample pattern **X** IF AND ONLY IF:

[1]. $\frac{P_i(t+\Delta) - P_i(t)}{1-P_i(t_0)} < p_{generate}$ AND

[2]. $t > t_0$ AND

[3]. $P_i(t+\Delta) < P_i(T)$

Figure 5: An example of error-driven minimal generation

of the sampled input pattern itself. However, it is not obvious which layers (if any) should take precedence over others as potential loci for generation. Any such precedence built into the generative process influences the structure that the network assumes as a result of learning. Some alternative designs are considered below:

The simplest strategy initiates a generation at any layer in the network whenever the criteria for generation (see above) are satisfied. This strategy is employed in recognition cone networks that learn by combined generation and weight modification (Honavar & Uhr, 1988, 1989a). A different strategy is suggested by the following assumption: Complex features of the input should be encoded based on *informative* simple features. This suggests biasing the system such that it tends to fill in the lower layers before generation can start filling in the higher layers of the network. This suggests biasing the system such that it tends to fill in the lower layers before generation can start filling in the higher layers of the network (This has clear parallels in the development of the visual system in mammalian brains

(Honavar, 1989).

Additional alternatives exist, e.g., networks that employ encodings of the input at multiple spatio-temporal resolutions (Honavar & Uhr, 1990b) may be biased so that generation at a lower resolution is preferred to one at a higher resolution for reasons of representational parsimony. Networks with regular topological structure (e.g., local receptive fields, multi-layer convergence) raise additional possibilities: e.g., a generated node might be replicated at each spatial location which is the implementation used in recognition cone networks for 2-dimensional pattern recognition (Honavar & Uhr, 1988; 1989a).

4.6. Node/link Evaluation, Network Pruning, and Network Reorganization

Often it is necessary to eliminate nodes that are evaluated to be useless (see below) - to make room for new (potentially more useful) extractions. Such an evaluation may be based on:

[1] The estimated information-content of a node - e.g., using the weights on the output links; or

[2] the *mutual information* estimates of redundancy of information encoded by the different nodes; or

[3] the results of competitive interaction among nodes to encode patterns from the environment.

The task of elimination of nodes from the is an instance of optimal feature sub-set selection problem in statistical pattern recognition (Fukunaga, 1990). A number of network pruning schemes have been proposed in the literature (Le Cun, Denker & Solla, 1990; Mozer & Smolensky, 1989; Hanson & Pratt, 1989) to reduce the size of networks (thereby potentially improving generalization as well because of the elimination of excess degrees of freedom used by the network for function approximation). Degeneration processes of this sort have not been integrated into generative learning designs that have been studied to date.

On a slower time-scale than that of generative learning, processes that reorganize previously generated nodes into structures (e.g., trees, groups, independent functional modules, etc.) may be used for a number of purposes:

[1] To make the network more compact (e.g., by replacing 2 or more nodes by a functionally equivalent single node);

[2] To eliminate excessive redundancy (e.g., by replacing 2 or more nodes that encode the same information by a single node); and

[3] To introduce fault-tolerance (e.g., by replacing a single node by a functionally equivalent distributed cluster of nodes).

5. Empirical Results

The range of alternative designs for generative learning discussed in this paper is obviously quite large. Only a small subset of such designs have been empirically investigated to date. A detailed discussion of their implementation and simulation is beyond the scope of this paper. The following brief summary is meant to give an indication of the part of the design space for generative learning algorithms that has been at least partially explored through simulation so far:

[1] Feedback-guided minimal generation through extraction of novel sub-patterns in multi-layer converging-diverging networks for perceptual learning of simple (compact and connected) 2-dimensional visual patterns (Honavar & Uhr, 1988; 1989a): Results suggest that generative learning is substantially faster than learning by weight-modification alone in a network of a-priory fixed topology, and the networks learned are generally compact.

[2] Some simple extensions that facilitate efficient learning from multi-resolution spatial, temporal, and spatio-temporal patterns (Honavar & Uhr, 1990b): Results suggest that the use of multi-resolution representations with a suitably controlled generative learning mechanism result in relatively parsimonious representations.

[3] Feedback-guided minimal generation through extraction of novel sub-patterns for pattern classification using a range of real-world data (e.g., soybean diseases, chess end games, audiology, iris): Simulations indicate that generative learning yields results comparable to those obtained by weight modification alone in a smaller number of learning cycles and with the resulting networks being relatively compact.

[4] Variations of the cascade-correlation architecture (Fahlman & Lebiere, 1990) on a range of real-world data sets such as the ones listed in [3] above: Simulations suggest about an order of magnitude speedup in learning within a certain range of parameter choices; again, the resulting networks are much smaller than the ones needed to attain comparable classification accuracy with weight modification using the generalized delta rule (Yang & Honavar, 1991).

Extensive experimental evaluation of generative learning algorithms is necessary to identify the strengths and limitations of different designs. However, in the opinion of the authors, the results obtained so far are quite encouraging. A large variety of alternative generative learning approaches remain to be explored.

6. Discussion, Summary, and Future Directions This section summarizes the key ideas presented in this paper and outlines some directions for future research in this area.

Generative Learning as Search

Generative learning can be viewed as a process of search for useful internal representations of the envornment. CN that learn using weight modification alone search for useful internal representations of the environment in the space of weight matrices within a fixed network topology. GCN that combine generative learning with with weight modification extend the search for such representations to the space of network topologies in addition to the space of weights. This necessitates the incorporation of additional control structures and processes in such networks, that constrain the search to promising regions of this space so that wasteful exhaustive search is avoided. Several such techniques for guiding the search that have been investigated so far include:

[1] Built-in general topological constraints on the network connectivity that are suggested by the problem domain - e.g., local connectivity and converging-diverging multi-layer structures for vision (Honavar & Uhr, 1988; Honavar & Uhr, 1989a).

[2] Controls that regulate learning so that the network is forced to generate new nodes or links only when it fails to improve its performance (e.g., error-driven minimal generation discussed above); and

[3] Novelty detectors that help to choose potentially information-rich extractions.

Several additional control structures can be used to constrain the search for, and the development of, appropriate internal representations of the environment. These include:

[1] *Vigilance* (first introduced by Grossberg (1976a; 1976b) in the context of *adaptive resonance theory*) which refers to the sensitivity of a node to mismatch between the pattern it is tuned for and the input.

[2] *Representational parsimony*: Several controls that bias the network toward learning efficient internal representations have been incorporated into generative learning designs investigated so far (e.g., feedback-guided minimal generation through extraction of novel patterns, guided search through multi-resolution spatial, temporal, and spatio-temporal representational hierarchies (Honavar & Uhr, 1990b). Parsimony requirement in some situations may conflict with - hence need to be resolved with - the need for damage resistance (through redundance), or rapid learning.

[3] *Plasticity*: controls the number, the locus, and the frequency of generations initiated. Several possibilities for dynamic regulation of network plasticity are suggested by the development of the mammalian visual cortex (Honavar, 1989).

[4] *Representational Bias*: Constraints on generation (e.g., the topological constraints on network architecture) constitute potentially powerful representational and inductive biases. Such biases can influence the space-time complexity and hence feasibility of generative learning algorithms for real-world applications.

Summary and Future Directions

Generative learning structures and processes examined in this paper offer potentially powerful learning capabilities in GCN - by enabling such networks to adaptively and incrementally modify their connectivity to meet the needs of the tasks that they have to perform (This does not necessarily imply the use of hardware architectures that are dynamically reconfigured during learning. Once an adequate network topology for a given task is discovered through generative learning, it can be implemented using any hardware or software that might be appropriate). A variety of regulatory controls can potentially influence the form as well as content of the internal representations of the environment learned by such networks.

Learning in most of today's CN involves a search (usually by gradient-descent on an error surface that represents the error between the desired and actual performance of the network so as to minimize the error) in the space of weights within a network topology that is fixed a-priori. If the initial choice of the number of nodes and/or network connectivity is inappropriate, such a network fails to attain the desired performance. Generative learning can potentially find an adequate network connectivity constructively, through learning. In multi-layer networks, generative learning can discover successively higher order relationships among attributes in the input patterns. Such networks can therefore potentially learn arbitrarily complex mappings from the universe of input patterns to the desired categories or class descriptions.

In CN that learn by weight modification alone, there is often a risk of getting caught in a local minimum, or a shallow trough in the error surface. Generation (and discarding) of nodes can be thought of as providing a means of dynamically altering the terrain in which gradient-descent is being performed, which incidentally, offers a way of potentially climbing out of the local minimum.

The range of alternative designs for generative learning discussed in this paper is obviously quite large. Only a small subset of such designs have been investigated so far empirically with encouraging results. A wide range of designs and a number of important questions (e.g., the issues of scalability and computational efficiency of such designs) remain to be investigated.

We are in the process of developing a flexible simulation environment to facilitate a systematic investigation of a a broad range of generative learning algorithms. Some topics currently under study include: integration of unsupervised and supervised generative mechanisms generative learning; a design for incremental generation of multi-level representations for the recognition and description of 3-dimensional objects (Honavar, 1992a; 1992b); generative learning with structured representations (Honavar, 1992a); improved weight modification algorithms, extraction and generalization mechanisms for generation; construction of iconic representations using generative learning; and experimental comparison of generative learning designs with each other as well as with a range of other learning methods.

References

Aha, D. W., & Kibler, D. (1989). Noise-tolerant instance-based learning algorithms. In *Proceedings of the 1989 International Joint Conference on Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

Ash, T. (1989). Dynamic node creation in backpropagation networks. *Connection Science - Journal of Neural Computing, Artificial Intelligence and Cognitive Research* 1 365-375.

Balakrishnan, K., & Honavar, V. (1992). Faster learning in multi-layer networks by eliminating flat-spots. Submitted for publication.

Bareiss, E. R. (1988). Protos: A unified approach to concept representation, classification, and learning. Ph.D thesis, University of Texas, Austin.

Barto, A. G., & Anandan, P. (1985). Pattern recognizing stochastic learning automata. *IEEE*

Transactions on Systems, Man, and Cybernetics 15 360-375.

Baum, E. B. (1989). A proposal for more powerful learning algorithms. *Neural Computation* 1 201-207.

Carbonell, J. G., Michalski, R. S., & Mitchell, T. M. (1983). An overview of machine learning. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.) *Machine Learning - An Artificial Intelligence Approach.* Palo Alto, CA: Tioga.

Diederich, J. (1988). *Knowledge-intensive recruitment learning.* Technical report TR-88-010. Berkeley, CA: International Computer Science Institute.

Dietterich, T. G., & Michalski, R. S. (1983). A comparative review of selected methods for learning from examples. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.) *Machine Learning - An Artificial Intelligence Approach.* Palo Alto, CA: Tioga.

Fahlman, S. E. (1988). Faster-learning variations on back-propagation. In G. E. Hinton, T. J. Sejnowski, & D. S. Touretzky (Eds.) *Proceedings of the 1988 Connectionist Models Summer School.* San Mateo, CA: Morgan Kaufmann.

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

Farmer, D. J., Packard, N. H., & Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Physica* 22D 187-204.

Feldman, J. A., & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science* 6, 205-264.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning* 2 139-172.

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

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

Gallant, S. I. (1990). Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks.* 1 179-.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: Freeman.

Gold, E. (1967). Language identification in the limit. *Information and Control* 10 447-474.

Greenough, W. T., & Bailey, C. H. (1988). The anatomy of a memory: convergence of results across a diversity of tests *Trends in Neuroscience,* 11, 142-147.

Grossberg, S. (1976a). Adaptive pattern classification and universal recoding I: parallel development and coding of neural feature detectors. *Biological Cybernetics* 23 121-134.

Grossberg, S. (1976b). Adaptive pattern classification and universal recoding II: feedback, expectation, olfaction, and illusions. *Biological Cybernetics* 23 187-202.

Grossberg, S. (1980). How does the brain build a cognitive code? *Psychological Review* 1 1-51.

Grossberg, S. (1987). Competitive learning: from interactive activation to adaptive resonance. *Cognitive Science* 11 22-63.

Hanson, S. J., & Pratt, L. Y. (1989). Some comparisons of constraints for minimal network construction with back-propagation. In D. S. Touretzky (Ed.) *Neural Information Processing Systems (Vol. 1).* San Mateo, CA: Morgan Kaufmann.

Hanson, S. J. (1990). Meiosis Networks. In D. S. Touretzky (Ed.) Neural Information Processing Systems (Vol. 2). San Mateo, CA: Morgan Kaufmann.

Hebb, D. O. (1949). *The Organization of Behavior.* New York, NY: Wiley.

Hinton, G. E. (1989). *Connectionist learning procedures Artificial Intelligence* 40 185-234.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: 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. In G. E. Hinton, T. J. Sejnowski, & D. S. Touretzky (Eds.) *Proceedings of the 1988 Connectionist Models Summer School.* San Mateo, CA: Morgan Kaufmann.

Honavar, V., & Uhr, L. (1989a). Brain-Structured connectionist networks that perceive and learn. *Connection Science - Journal of Neural Computing, Artificial Intelligence and Cognitive Research* 1 139-160.

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, CA: Morgan Kaufmann.

Honavar, V. (1989). *Perceptual development and learning: From behavioral, neurophysiological, and morphological evidence to computational models.* Technical report 818. Madison, WI: University of Wisconsin, Computer Sciences Dept.

Honavar, V., & Uhr, L. (1990a). Coordination and control structures and processes: possibilities for connectionist networks. *Journal of Experimental and Theoretical Artificial Intelligence* 2 277-302.

Honavar, V., & Uhr, L. (1990b). Efficient learning using multi-resolution representations of spatial, temporal, and spatio-temporal patterns. In *Proceedings of the Conference on Neural Networks and Parallel-Distributed Processing.* Indiana University-Purdue University.

Honavar, V., & Uhr, L. (1990c). Symbol processing systems, connectionist networks, and generalized connectionist networks. Technical Report 90-24. Ames, Iowa: Iowa State University Department of Computer Science.

Honavar, V. (1992a). Learning algorithms based on generalized distance measures. In *Proceedings of the 1992 SPIE Conference on Adaptive and Learning Systems* (To appear).

Honavar, V. (1992b). Learning parsimonious representations of three-dimensional shapes. In *Proceedings of the NATO Advanced Research Workshop on Mathematical Descriptions of Shape* (To appear). Netherlands.

Hunt, E. B. (1962). *Concept Formation: An Information Processing Problem.* New York: Wiley.

Jerne, N. K. (1967). Antibodies and Learning: Selection Versus Instruction. In G. C.

Quarton, T. Melnechuk, & F. O. Schmitt (Eds.) *The Neurosciences: A Study Program.* New York, NY: Rockefeller University Press.

Judd (1990). *Neural Networks and the Complexity of Learning.* Cambridge, MA: MIT Press.

Koza, J. R. (1990). *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems.* Techincal report STAN-CS-90-1314. Stanford, CA: Stanford University Department of Computer Science.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence* 33.

Le Cun, Y., Denker, J. S., Solla, S. A. (1990). Optimal Brain Damage. In D. S. Touretzky (Ed.) *Neural Information Processing Systems (Vol. 2).* San Mateo, CA: Morgan Kaufmann.

Michalski, R. S., & Stepp, R. E. (1983). Learning from observation: conceptual clustering. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.) *Machine Learning - An Artificial Intelligence Approach.* Palo Alto, CA: Tioga.

Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.) (1983). *Machine Learning - An Artificial Intelligence Approach.* Palo Alto, CA: Tioga.

Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.) *Machine Learning - An Artificial Intelligence Approach. (Vol. 2).* San Mateo, CA: Morgan Kaufmann.

Michalski, R. S., & Kodratoff, Y. (1990). Research in machine learning: recent progress, classification of methods, and future directions. In Y. Kodratoff & R. S. Michalski (Eds.) *Machine Learning - An Artificial Intelligence Approach (Vol. 3).* San Mateo, CA: Morgan Kaufmann.

Minsky, M. & Papert, S. (1969). *Perceptrons: An Introduction To Computational Geometry.* Cambridge, MA: MIT Press.

Mozer, M. C., & Smolensky, P. (1989). Skeletonization: a technique for trimming the fat from a network via relevance assessment. In D. S. Touretzky (Ed.) *Neural Information Processing Systems (Vol. 1).* San Mateo, CA: Morgan Kaufmann.

Nadal, J. P. (1989). Study of a growth algorithm for a feedforward network. *International journal of neural systems.* 1 55-.

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

Newell, A. (1980). Physical symbol systems. *Cognitive Science* 4 135-183.

Nilsson, N. J. (1965). *The Mathematical Foundations of Learning Machines.* New York: McGraw-Hill.

Pagallo, G., & Haussler, D. (1989). Two algorithms that learn DNF by discovering relevant features. In *Proceedings of the 6th International Workshop on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning* 1 81-106.

Rosenblatt, F. (1958). The Perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* 65 386-408.

Rujan, P., & Marchand, M. (1989). Learning by activating neurons: a new approach to learning in neural networks. *Complex Systems* 3 229-.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation, In *Parallel Distributed Processing - Explorations into the Microstructure of Cognition (Vol. 1: Foundations).* Cambridge, MA: MIT Press.

Rumelhart, D. E., Hinton, G. E., & McClelland, J. L. (1986). A general framework for parallel distributed processing. *Parallel Distributed Processing - Explorations into the Microstructure of Cognition (Vol. 1: Foundations).* Cambridge, MA: MIT Press.

Rumelhart, D. E. (1988). *Parallel Distributed Processing.* Plenary lecture given at the IEEE International Conference on Neural Networks, San Diego, CA.

Sandon, P. A. (1987). *Learning Object-Centered Representations.* Ph.D Thesis, University of Wisconsin-Madison.

Shannon, C. E. (1963). The mathematical theory of communication. In C. E. Shannon & W. Weaver (Eds.) *The Mathematical Theory of Communication* Urbana: University of Illinois Press.

Tawel, R. Does the neuron "learn" like the synapse? In D. S. Touretzky (Ed.) Neural Information Processing Systems (Vol. 1) San Mateo, CA: Morgan Kaufmann.

Tsotsos, J. K. (1990). Analyzing vision at the complexity level. *Behavioral and Brain Sciences* 13 423-469.

Uhr, L. (1990a). Connectionist networks defined generally, to show where power can be increased. *Connection Science - Journal of Neural Computing, Artificial Intelligence and Cognitive Research* (In press).

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

Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences.* Ph.D Thesis, Harvard University.

Wolpert, D. H. (1989a). *A Mathematical Theory of Generalization: Part I.* Los Alamos, NM: Los Alamos National Laboratory.

Wolpert, D. H. (1989b). *A Mathematical Theory of Generalization: Part II.* Los Alamos, NM: Los Alamos National Laboratory.

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. In *IRE WESCON Convention Record (Part 4).*

Yang, J., & Honavar, V. (1991). Experiments with the cascade-correlation algorithm. In Proceedings of the Fourth UNB Artificial Intelligence Symposium. Fredericton, Canada.