Learning in Artificial Neural Systems

Christopher J. Matheus and William E. Hohensee

Inductive Learning Group
Department of Computer Science
University of Illinois at Urbana–Champaign
1304 West Springfield Avenue
Urbana, Illinois 61801 USA

Arpanet: {matheus | hohensee}@a.cs.uiuc.edu
Phone: (217) 333–3965

December 1987

Keywords: artificial neural systems, neural networks, connectionism, learning

This report is a slightly modified version of a paper to appear in
Computational Intelligence Journal, Vol. 3, #4, 1987.

Learning in Artificial Neural Systems

Christopher J. Matheus and William E. Hohensee

Computer Science Department
University of Illinois at Urbana–Champaign

ABSTRACT

This paper presents an overview and analysis of learning in *Artificial Neural Systems* (ANS's). It begins with a general introduction to neural networks and connectionist approaches to information processing. The basis for learning in ANS's is then described, and compared with classical machine learning. While similar in some ways, ANS learning deviates from tradition in its dependence on the modification of individual *weights* to bring about changes in a knowledge representation distributed across connections in a network. This unique form of learning is analysed from two aspects: the selection of an appropriate network architecture for representing the problem, and the choice of a suitable learning rule capable of reproducing the desired function within the given network. The various network architectures are classified, and then identified with explicit restrictions on the types of functions they are capable of representing. The learning rules, i.e., algorithms that specify how the network weights are modified, are similarly taxonomised, and where possible, the limitations inherent to specific classes of rules are outlined.

1. Introduction

Neural networks, parallel distributed processing, and connectionist models — which we shall collectively refer to as *Artificial Neural Systems* or ANS's — represent some of the most active research areas in artificial intelligence (AI) and cognitive science today. The recent excitement is due to the promising qualities artificial neural systems exhibit in addressing challenging questions of intelligence. For instance, characteristic of an ANS is its distribution of knowledge across a network of units. Since these units act in some ways as evaluators of locally competing hypotheses, the computational strengths of ANS's can be exploited in the solution of a broad array of simultaneous constraint satisfaction problems. ANS's have shown promising results for a number of these problems, including content addressable memory, pattern recognition and association, category formation, speech production, and global optimisation (see Kohonen 1984; Rumelhart & McClelland 1986; Anderson 1977; Sejnowski & Rosenberg 1986; Hopfield & Tank 1986). In addition, the distribution of knowledge across a matrix of individual processing units leads to a natural kind of fault tolerance. Since the failure or loss of small portions of the network does not significantly alter overall system performance, ANS's tend to degrade gracefully. The inherent parallelism of distributed representations make ANS's ideal candidates for implementation on new parallel hardware architectures. Moreover, as crude approximations to biological neural systems, ANS's contribute to a new paradigm for dialogue among many of the disciplines of cognitive science. But perhaps the most interesting quality, and the central topic of this paper, is the way in which learning is effectively accomplished in ANS's through simple, low–level

learning rules.

Before we examine the low–level details of learning in ANS's, consider first the problem of learning in general. From one perspective, the goal of a learning system is to acquire a function that maps input events to output responses. For a given learning problem, a set of input features defines the multi–dimensional input space called a "feature space" in which all possible events exist as vectors. A system's output responses may use this same space or possibly a separate "concept space" or "hypothesis space" (Rendell 1986). In either case, an ANS, or any learning system, can be viewed as a black box that learns to map input events to appropriate output responses.

How then does an ANS differ on this level from other learning approaches? When properly configured, ANS's are theoretically capable of acquiring any arbitrary mapping function (Volper & Hampson 1986), and therefore have no inherent representational or learning advantages or disadvantages compared to traditional machine learning systems. In terms of efficiency, ANS's can be computationally more expensive (at least when implemented on sequential machines), and their training sets are frequently larger than required for more traditional methods. In fact, in certain situations, standard data structures and algorithms can be shown to learn mapping functions with greater efficiency than a typical ANS (Omohundro 1987).

On the other hand, ANS's exhibit interesting and useful learning capabilities. Generalizations "spontaneously" emerge through an ANS's exposure to, and processing of, input patterns. In multi–layered ANS's, these generalizations result in the construction of "new terms" for internal representations, thereby suggesting solutions to the problem of constructive induction (Rendell 1986). The generalizations that occur in some ANS's are remarkably similar to the results of learning in humans, displaying many of the same strengths and weaknesses. Further study of the emergent properties of ANS's may therefore lead to a better understanding of the process of human learning. Finally, while ANS's have already demonstrated useful learning capabilities, the field is new and may yet hold even greater unexplored insights into computational intelligence and learning.

This paper presents a general overview of learning within artificial neural systems. It does not assume familiarity with the terminology or operation of ANS's. Instead, we begin with an introduction to a simple artificial neural unit (section 2.1) and describe the components of a basic network configuration (section 2.2). The unique representation of knowledge within ANS's has a direct bearing on the nature of learning, and we therefore introduce some representational concepts early, in section 2.2. With this background in mind, specific issues surrounding learning in ANS's are examined. First, we contrast learning in ANS's with learning as it is viewed in traditional AI domains (section 3.1). We then separately consider the questions of 1) the learnability of a function within a network architecture, and 2) learnability given a specific learning rule. Looking first at the representational capabilities, we demonstrate some constraints imposed by the selection of different network architectures (section 4). Three general classes of learning rules are then introduced, and the capabilities and limitations of specific rules are considered (section 5). This paper strives to provide an introduction and overview of issues important to practical learning within ANS's. For a more formal analysis of computability and complexity issues see (Volper & Hampson 1986; Hampson & Volper 1986; Egecioglu *et al.* 1986).

2. Artificial Neural Systems

The distinguishing feature of Artificial Neural Systems is the use of many simple but highly interconnected parallel processing units. These units are, to varying degrees, based on the

functional characteristics of biological neurons. The objective of these systems, however, is generally not neural modelling, but rather to utilize the computational characteristics they exhibit. Useful computational qualities emerge when these simple units are organized into networks. The nature and arrangement of the connections between units determines both what types of functions a network is able to compute, and what it is able to learn with respect to a given learning rule. In this section we will review the basic function of a simple neural unit and discuss how collections of them are organized into networks. In addition, the manner in which knowledge is represented within a network is described. (The statements made in this section are generalizations of typical ANS's and are not intended to be universally applicable to all network designs.)

2.1. An Artificial Neural Unit

The principal component of an ANS is its processing unit (see Fig. 1). Input signals come into a unit along weighted connections, or "synapses," from neighboring units, and excite (in the case of positive links) or inhibit (for negative links) the unit's "firing" activity. The magnitude of a weight determines how strongly the output of one unit will influence another unit's activity: the larger the absolute weight between two units the greater is the efficacy of the connection, and the stronger the influence. Typically, the weighted contributions from all inputs are summed to determine an *activation level* for that unit:

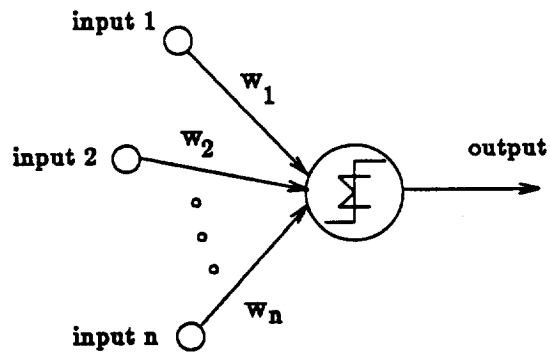


Figure 1: **A prototypic ANS unit.** Inputs from units 1 through n are summed together along weighted connections, w, to produce an activation value. That activation value is then typically thresholded to produce an output value for the unit.

$$a_i = \sum_j w_{ij} o_j ,$$

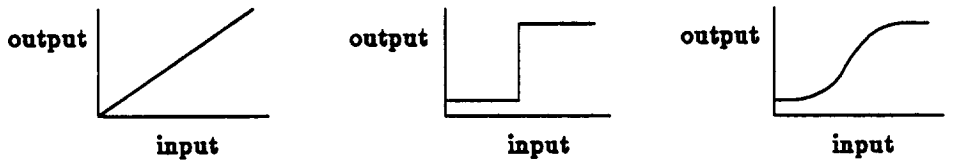
where a_i is the level of activation of the ith unit, w_{ij} is the weight from unit j to unit i, and o_j is the output signal from unit j. While this value represents the unit's "activation," it is not usually the same as the unit's output signal, o_i. Generally o_i is calculated by applying a non–linear output function, such as the simple step function:

$$o_i = \begin{cases} 1 & \text{if } a_i > \theta \\ 0 & \text{if } a_i \leq \theta \end{cases}$$

where θ is the threshold value at which the unit turns "on." After its calculation, this output value is transmitted along the weighted outgoing connections to serve as input to subsequent processing units.

Though this simple processing of input signals by an individual unit is basic to all ANS's, the details of implementation may vary between models. For example, a number of different output functions have been proposed. Figure 2 depicts the more common of these: a linear output function, the simple step–function, and a sigmoidal output function. The range of output values assumed by the units may also be varied, with different models choosing among binary, graded, or continuous values. Some networks further incorporate the effects of activation decay and internal resistance, modelling similar functions found in biological neurons. These and other details are, however, beyond the scope of this paper, and are more fully described in (Rumelhart & McClelland 1986). What is important to this overview is that the basic operation of an individual unit is extremely simple:

1) a level of activity, a_i, is calculated from the weighted input signals from other units
2) the unit's output signal, o_i, is calculated based on the unit's activation
3) o_i is transmitted along links to neighboring units for further processing.



A. linear function B. step–function C. sigmoidal function

Figure 2: **Output functions.**

2.2. Networks of Units

An ANS network is defined by specifying the connections between units. This specification usually takes the form of a *connectivity* or *weight matrix*. Connections are typically unidirectional although bidirectional links are assumed in some models (e.g., Hopfield 1982). In the most general case, every unit in the network is connected to every other unit. Such networks are said to be *totally connected*. In many ANS's, however, some connections are eliminated in order to improve performance and simplify learning. Figure 3 shows examples of a totally connected network and two networks with restricted architectures. The architecture of an ANS defines the structure used for knowledge representation and processing. Different architectures impose different biases on the functions that can be represented within the network. As will become evident, knowledge representation in an ANS is quite different from traditional AI approaches. Two forms of knowledge representation can be identified within neural systems: 1) the state vector of unit activations, and 2) the representation of the mapping function encoded in the weights. We will consider each of these in detail since they play an important role in our later discussions of learning.

The *state vector* consists of the vector of output values (or alternatively, activation values) taken across all units in the network. It represents the instantaneous chunk of knowledge currently being processed by the network. For a given architecture, portions of this vector may be used to represent the input and output of the problem. Thus, as shown in Figures 3b and 3c, the bottom–layer of the network may represent the input vector, while the top layer encodes the system's output; together they represent the knowledge structures present in the domain and range of the mapping relationship to be learned. In some networks, such as the multi–layered ANS shown in Figure 3c, units exist which serve neither as input nor output. These units are called *hidden units* since they are invisible to the external world. Hidden units are not initially assigned specific meaning, but rather provide a network with additional degrees of freedom in

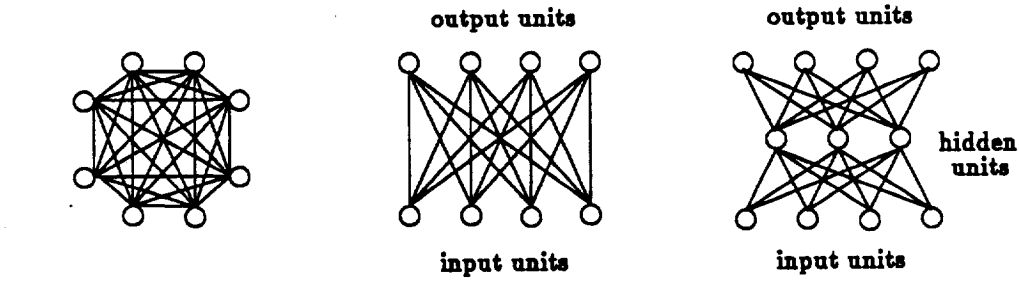


Figure 3: **Network architectures.**

which to construct useful internal knowledge representations. We will have more to say below (section 4.2) about the extended representational capability that hidden units provide, as well as the complications they bring to learning.

The second representational form is of longer duration, lasting beyond the short–term activation state vector. This persistent knowledge is encoded in the weights between the units, and it is through the modification of this distributed knowledge structure that learning occurs. Implicitly represented across these connections is the mapping function that defines how a given input state is translated into its associated output. Because all weights contribute to the representation, the acquired function is said to be "distributed" across the network, as opposed to being represented locally by any individual unit. The notion of a *distributed representation* is a matter of much debate and confusion. For a more thorough discussion of this topic the reader is referred to (Rumelhart *et al.* 1986; Hinton 1984).

One aspect of distributed representations relevant to our purposes is that all input/output pair mappings within the represented function are superimposed one on top of another across the network weights. In contrast, a symbolic representation will typically use a collection of locally represented function components. For example, imagine a production system where the mapping from a set of inputs to a unique output is encoded by a single rule such as [IF (*flies* ?X) AND (*has–feathers* ?X) THEN (*isa* ?X *bird*)]. The complete mapping function of the learning system is then represented by a collection of many such individually represented rules. In a network, *all* of the necessary rules are implicitly represented across a *single* set of weights. This characteristic of the representation becomes important from a learning perspective. Since learning involves the changing of weights, and any given weight may participate in the representation of multiple pieces of information, the learning of new knowledge will often depend upon and alter existing knowledge. It is as if adding the rule [IF (*flies* ?X) AND (*mammal* ?X) THEN (*isa* ?X *bat*)] could somehow affect (for better or worse) the knowledge contained in the *isa–bird* rule.

2.3. Information Processing

There are two different methods for processing information in ANS's. The first is called "settling" and the second "feedforward processing." Which of these methods is used for a specific ANS depends upon the chosen network model and the nature of the mapping function. Settling tends to be used primarily with auto–associative networks (discussed below), and usually in the context of pattern completion or optimization tasks. Processing in these systems begins by encoding the input vector across the entire state vector. New activation states for the individual units are then derived as the result of the propagation of signals between units. This iterative process of computing new states continues until the system "settles" into a stable state where no further changes are possible. At this point, the final state vector represents the output of the system. The manner in which the updating of unit states is conducted again depends upon the specifics of the model. Some models synchronously update all the units at each time step. Others use an asynchronous process and update units semi–randomly. The synchronous approach has the advantage of simplicity and will often settle more quickly. Asynchronous networks, on the other hand, are less likely to become trapped in local minima or oscillate perpetually between two states.

The second general form, feedforward processing, occurs predominately in feedforward, multi–layered networks. The knowledge structure to be processed is first represented as an activity pattern across the vector of input units. This information is then propagated through the weighted connections to the next level of units. The resulting pattern of activity at this layer

is, in turn, passed on to the next higher level. The feedforward processing of information continues in this step–wise, synchronous manner until a solution is represented as the activity across the final output layer. It is possible to combine this feedforward processing with "feedback" processing in which information from higher layers feeds back to lower levels. In this type of network (see Fukushima 1986), multiple iterations of the feedforward and feedback processes may be necessary before a stable final output is realized. The network's processing in this case becomes more like a settling process, terminating when activity stabilizes or diminishes significantly.

3. Learning Issues

We now turn to the issue of learning. As stated in the introduction, we have chosen to view the general problem of learning as the process of acquiring a function that maps input vectors (stimuli) to output vectors (responses). To learn this mapping function, a learning system receives a set of training events from which it constructs an internal representation of the function. Considered at this level, learning in ANS's is no different from learning in more traditional systems. In fact, ANS's can be effectively viewed within the context of traditional machine learning paradigms, as is done in section 3.1. The similarities between ANS's and traditional approaches, however, do not carry through to lower levels of analysis where the details of representation and learning become significant. Section 3.2 considers the importance of the relationship between representation and learning within ANS's.

3.1. Contrast to Traditional Machine Learning

Machine learning has traditionally been taxonomized into rote learning, learning from example (supervised learning), and unsupervised learning (Michalski *et al.* 1983). Learning in ANS's can be viewed from this same general perspective. One class of networks, for example, is designed to memorize specific patterns which are later recalled when either the same or a closely related input is presented to the system. *Content addressable memories*, such as those found in the Hopfield Network (Hopfield 1982) or some of the models by (Kohonen 1984), exemplify this form of rote–like learning.[1] In contrast, many networks learn from examples to map input patterns into a different set of associated output patterns. During learning, a network of this type is typically provided with an input pattern to which it responds by computing its best guess of the desired output. A teacher then provides the system with the correct output pattern, allowing the system to adjust the weights between units, if necessary, to bring its internal representation closer to the desired result. Rosenblatt's perceptron (1962), Widrow & Hoff's Adeline neuron (1960), and Sejnowski *et al.*'s NETtalk system (1986) each exhibit this form of supervised learning. Still other networks are designed to learn without direct guidance from a teacher, as is the case in competitive learning systems. In a typical example, exposure to a dynamic input stream causes individual units to become unique feature detectors of different input qualities, thereby effectively partitioning the space of events into useful clusters (Kohonen 1977; Rumelhart & Zipser 1985). The Adaptive Resonance Networks of Carpenter & Grossberg (1986) exhibit a similar type of unsupervised learning in the induction of category prototypes.

[1] The memorization of specific patterns in ANS's is different from pure rote learning. For instance, generalizations are made during learning that influence how unobserved events will be classified. Also, the addition of new memories is affected by and may alter existing memories.

3.2. Representation versus Learning

Although learning in ANS's can be abstractly viewed from this traditional perspective, there are significant differences as to how learning is performed when compared with other machine learning systems. The main distinctions concern two issues at the heart of any learning problem: the choice of the problem's representation, and the selection or development of a specific learning procedure. In traditional machine learning, symbolic representations such as first–order predicate calculus, decision trees, version spaces, schema, etc., are used almost universally to represent knowledge–level information (Michalski *et al.* 1983). These representations are the structures upon which the learning procedures directly operate, modifying them to reflect the results of classification. The learning algorithms employed by various systems are designed around and often critically depend upon the underlying representation chosen for the problem. This situation is the same for ANS's: the learning algorithms are specific to the chosen representation. The representations used in ANS's, however, are quite different from anything typically found in traditional machine learning. The distribution of knowledge across weighted connections, as described above, is a distinguishing quality of ANS's, and it necessitates a unique approach to learning.[2] More specifically, learning in ANS's is realized through the modification of the connection weights according to a chosen *learning rule*. Before reviewing some of the different learning rules, we will first look more closely at the relationship between representation and learning in ANS's.

The issue of representation versus learning in ANS's can be reduced to the following two considerations: 1) the representational capability of the network architecture being used, and 2) the capacity for learning within that architecture given a specific learning rule. The first of these concerns whether the architecture of a particular ANS design is capable of representing the function which is to be learned. Some architectures impose inherent representational limitations such that they are incapable of representing certain classes of functions. The classic example of this problem is the perceptron (Rosenblatt 1962), which is constrained to representing only linearly separable functions (discussed below). The second issue is, given a suitable network architecture, does the selected learning rule have the potential for acquiring the desired mapping function? Minsky and Papert (1969) demonstrated that any function can be represented by an appropriately chosen network architecture, but they conjectured there would not in general be a procedure for learning any arbitrary function from examples. The question thus becomes, for a given network and function, will a particular learning rule be capable of solving the learning problem? It is sometimes possible to show that a learning rule is guaranteed to converge to the correct representation if one exists (e.g., the perceptron convergence theorem (Minsky & Papert 1969)), though unfortunately this is not true in general.

4. Representational Capabilities

Representational capability and learnability are concerns central to the problem of learning in ANS's. In this section we will consider the issue of representation within ANS's. Section 5 will focus on the details of learning. For our analysis of representational capabilities it will be convenient to identify two basic classes of ANS architectures — auto–associative networks and hetero–associative networks. We will look at the representational capabilities of these classes

[2] The claim that the representations within ANS's are distributed across weights does not mean that they cannot represent more abstract symbolic structures such as trees and schema (see Touretzky 1986). The claim here is that the level at which the learning methods operate is fundamentally different between classical machine learning and ANS's.

individually.

4.1. Representation in Auto–Associative Networks

The *auto–associative* ANS's have a single layer of units and a single layer of weights with no hierarchical orientation to the network (see Fig. 3a). The major distinguishing quality of this class of networks is that the input, output, and state vectors are all one and the same, i.e., every unit of the network participates in the representation of both input and output data (Hinton 1986). This design characteristic means the input and output vectors must be of the same length, and are usually chosen from the same domain. Often in these cases the networks are totally connected and the weighted links between units may be bidirectional. Units may also be connected to themselves, providing direct "feedback" and thereby allowing a unit's current state to affect its next state.[3] Indirect feedback is prominent in auto–associative networks, occurring wherever a cycle exists in the network, as is trivially the case for bidirectional connections.

Computation or processing in an auto–associative network is initiated by instantiating an input pattern across the network units, and then, as described earlier, allowing the system to iteratively calculate unit activation levels, and propagate outputs through the net until it settles into a final output state. The output states are the memories stored in the system through learning. During settling, the input patterns are "attracted" to the closest memory state.[4] An analogy is often drawn between this settling process and the search for a minimum in an energy function. Imagine, for example, an energy surface superimposed over the feature space. Each event or point in the space is associated with a specific energy value. The memories represented in the auto–associative network correspond to minima in this energy surface (see Fig. 4). Due to the nature of the settling procedure, processing of the network causes the system to fall from its initial state into the closest energy "well."

What, if anything, can be said about the representational constraints of auto–associative networks? Using the analogy of the energy function, it can be shown that auto–associative networks are limited to representing mappings which can be defined in terms of the minima of a quadratic function depicting a continuous energy surface (Egecioglu *et al.* 1986). The complexity of this function is constrained by both the number of units and the range of outputs of the units (continuous outputs permitting finer resolution than binary–valued units). Because of this quality of associating an input with the nearest memory image, auto–associative networks have been used successfully (and almost exclusively) for pattern recognition/completion (Kohonen 1984; Hopfield 1982), though such networks have also been used to find good approximations to certain global optimization problems (Hopfield & Tank 1986).

4.2. Representation in Hetero–associative Networks

Architectures that explicitly distinguish between input and output units (e.g., Fig.'s 3b & 3c) constitute our second class of networks. They are sometimes called *hetero–associative* networks because they associate an input pattern presented across a designated layer of input units, with a specific output pattern across a separate output layer. This differs from the auto–

[3] In some models this capability for direct feedback is prohibited in order to guarantee stable, non–oscillating final states (see Hopfield 1982).

[4] Closeness can be measured by the Hamming distance between two state vectors in feature space.

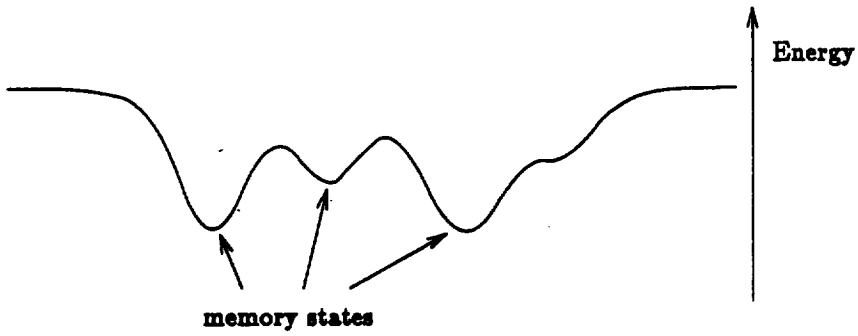


Figure 4: **Energy surface representing learned memory states.** Each well of the energy surface represents a final memory state into which the network may settle.

associative networks in that in addition to making a semantic distinction between input and output units, hetero–associative networks impose restrictions on the arrangement of connections that are permitted. A typical constraint, for example, is that units in one layer project only to units in subsequent layers. By constraining the network connections the overall number of weights can be reduced, decreasing the time needed for learning and computation. When these connections are unidirectional without feeding back to previous layers, computation can be efficiently accomplished using single–pass, feedforward processing. Such constraints on the connectivity of the network, however, can also restrict the class of functions that can be represented.

If we limit our consideration to networks having strictly acyclic, feedforward connections, some specific representational constraints can be identified for the classes of Boolean functions computable under different hetero–associative architectures. Consider first the simple, hetero–associative ANS shown in Figure 3b. A single layer of weighted connections serves to link two distinct input and output layers of the network. The perceptron model (Rosenblatt 1962) was a single–layered hetero–associative network of this type (with only one output unit), and was one of the earliest models to demonstrate interesting results. Unfortunately, as a representational system the perceptron, and other similar models, are severely limited by their architecture. Single–layered networks are inherently constrained to representing functions whose domain is linearly separable. This limitation results because a single perceptron of multiple input units and one binary–valued output unit is simply a linear discriminator that defines a hyperplane splitting the multi–dimensional feature space into two regions (Minsky & Papert 1969). This limitation is depicted graphically in Figure 5a.

To achieve greater representational and functional capability requires the addition of *hidden units*. As described above, hidden units are isolated from the external world in that they do not directly participate in the system's input or output. An ANS with hidden units usually have at least two layers of connecting weights: one layer of weights connecting the input to the hidden units, the other from the hidden to the output units (see Fig. 3c). ANS's in this class of network

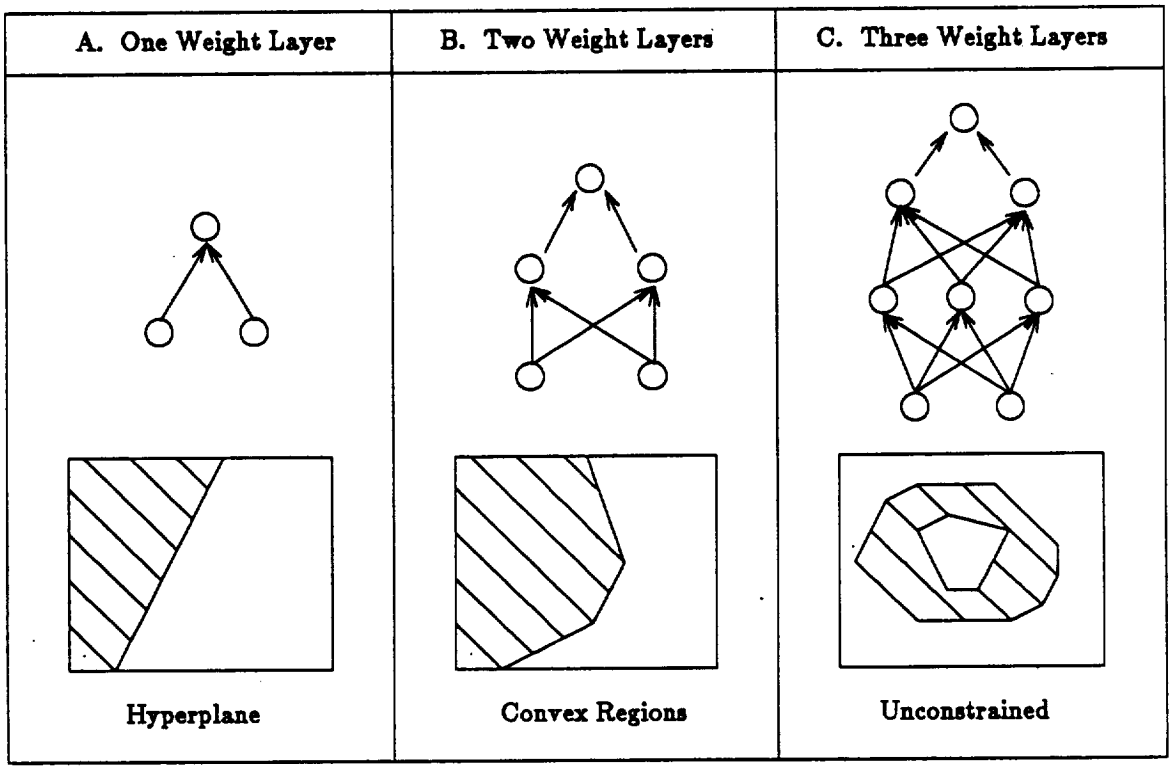


Figure 5: **Computable output regions for three different network architectures (after Lippman 1987).** *5a*: A network composed of one layer of weights is restricted to dividing the output space into two distinct hyperplanes. *5b*: A two–layered network may represent functions which map into convex open or closed regions. *5c*: A three–layered network given enough hidden units, is unconstrained and has the capacity to represent any arbitrary function.

architectures are called multi–layered, hetero–associative networks. Multi–layered networks are the focus of most current research, both because they have extended computational capabilities, and because the addition of hidden units considerably complicates the learning process. The learning problem that arises from the introduction of hidden units is the *credit assignment problem* (Minsky & Papert 1969). Since there is no knowledge of what state a hidden unit "should" be in, it is difficult to determine whether a given hidden unit is contributing positively or negatively to the overall performance of the system. The implications of this problem will become apparent later when we discuss specific learning rules.

A feedforward network with two layers of weights (one layer of hidden units), unlike a single–layered network, is able to represent any convex–open or closed region in feature space (see Fig. 5b). For Boolean categories described by Boolean features, a two–layered network can represent any arbitrary mapping function. This result follows from the fact that 1) any logical function can be implemented as a circuit with a layer of AND gates feeding into a single OR gate (Muroga 1971; Volper & Hampson 1986), and 2) with the appropriate choice of thresholds, linear threshold units can mimic the functions of AND and OR gates. Alternatively, this result is arrived at by observing that a convex region can be defined by a single unit which AND's the outputs from a set of perceptron–like units, forming the intersection of multiple hyperplanes (see Fig. 5b).

When the input features are multi–valued or continuous, a two–layered network may no longer be able to adequately represent the desired function. In this case, a three–layered network may become necessary. A three–layered, feedforward network can be shown to be adequate for representing any arbitrary function, given enough units at each internal layer (Hanson & Burr 1987; Lippmann 1987). An informal proof of this comes from observing that any geometric region in a multi–dimensional space can be approximated to an arbitrary degree of accuracy by ORing the outputs of a collection of two–layered networks which individually describe an appropriate set of convex–closed regions in the function domain (see Fig. 5c) (Lippmann 1987).

4.3. Summary of Representational Capabilities

What the above analysis shows is that the choice of a particular ANS architecture may result in inherent biases or constraints that restrict the class of representable functions. In the most general neural network, where every unit is connected to every other unit, there are no inherent representational biases, other than the obvious limits imposed by the finite number of units in the network. But while a completely connected ANS can mimic the architecture of any partially connected network having the same number of units (i.e., by selectively assigning connective links to zero weights), efficiency and learnability considerations often disfavor this most general approach, especially in large networks. Consequently, the network architectures chosen for practical systems frequently restrict the connections between units. By eliminating potential connections, however, the resulting architecture may be unable to represent certain function classes. Provided the network design permits the representation of the problem's mapping function, a restricted architecture can in many cases improve performance and learning.

5. Learning Rules

Even if a particular ANS architecture has the theoretic capability to model the desired function, there remains the question of whether the learning rule that is used will be able to produce the appropriate representation. Since long–term representations are distributed across the

weights of the interunit connections, learning new information is accomplished by modifying these weights in such a way as to improve the desired mapping between input and output vectors. This mapping function is learned by adjusting the weights according to some well defined learning rule. It will be convenient to identify three general classes of rules. The first, which we will call *correlational rules*, determine individual weight changes solely on the basis of levels of activity between connected units. The second class is the set of *error–correcting rules*, which rely on external feedback about the desired values of the output units. The third class comprises the *unsupervised learning rules*, in which learning occurs as self–adaptation to detected regularities in the input space, without direct feedback from a teacher.

It is interesting to note the analogic connection that can be drawn between these classes of ANS learning rules and traditional learning paradigms. The correlational rules are frequently found in auto–associative networks that perform rote–like learning of specific memory states (Hopfield 1982; Kohonen 1984). The error–correcting rules are predominately employed for supervised learning from examples since they use knowledge about how the system deviates from the teacher's examples to modify the connective weights (Rosenblatt 1962; Rumelhart *et al.* 1986; Sejnowski & Rosenberg 1986). The unsupervised learning rules are used in situations where the network must learn to self–adapt to the environment without receiving specific feedback from a teacher (Kohonen 1984; Rumelhart & Zipser 1985; Klopf 1986). These generalized relationships between learning rules and learning paradigms are only first approximations, and there are, of course, exceptions to each of these claims. Viewing the learning rules from this perspective, however, can help to illustrate their general usage.

In the following sections we will examine each class of learning rules in turn. Where possible, limitations and biases imposed by specific rules will be identified. Unlike some of the precise representational statements made in the preceding sections, similar constraints imposed by the learning rules have not been identified in all cases. In place of strict learnability claims, we will often instead use examples of specific learning rules to convey the general capabilities of a class of rules. The examples used below are chosen to be representative of the classes of rules in demonstrating their characteristic limitations. Space unfortunately does not permit an exhaustive review of all the significant learning rules.

5.1. Correlational Rules

The inspiration for many ANS learning rules can be traced to the early work of Donald Hebb (1949). Hebb's research on learning in biological systems resulted in the postulate that the efficacy of synapses between two neural units increases when the firing activity between them is correlated. The implication is that the connection from unit A to unit B is strengthened whenever the firing of unit A contributes to the firing of unit B. This rule can be defined formally as:

$$\Delta w_{ij} = o_i o_j ,$$

where Δw_{ij} is the change in the connective weight from unit j to unit i, and o_i and o_j are the output levels of the respective units. Notice that the only information affecting the weight change is "local," — it is derived solely from the levels of activity of the connected units, and not from any knowledge of the global performance of the system. Most ANS learning rules use information about the correlated (or anti–correlated) activity of connected units, but what makes a rule fit into the class of correlational rules is that these activities are the *only* basis for weight

changes.

In addition to Hebb's postulate, other correlational learning rules have been proposed for learning in ANS's. One such rule, sometimes termed anti–Hebbian, states that if a unit's firing is followed by a lack of activity at a second unit to which the first unit's signal is being projected, then the connection strength between those two units is diminished (Levy & Desmond 1985). There are only a small number of possible correlational rules of this sort. Since two connected units may assume two states of activation (assuming binary–valued units), and their connection weight can be increased, decreased, or left unchanged, there are 3^4 or 81 possible correlational learning rules of this type. Many of these are uninteresting, such as the rules in which no modifications ever take place, or where they always occur. Three of the more useful correlational rules are depicted in Figure 6.[5]

5.1.1. The Hopfield Network

Auto–associative networks typically employ correlational rules, with the classic example being the Hopfield Network which uses a combined Hebbian/anti–Hebbian learning rule. A Hopfield Network is a totally connected auto–associative network in which learning is accomplished by storing "memory states" into the network's weight matrix. To store a particular state, the following rule is applied: if two units are both "on" or both "off" then increase the weight of their connections; if one unit is "on" and the other is "off" then decrease their weights. Again assuming binary output values, the formal equation for this rule can be stated as:

Hebbian			Anti–Hebbian			Hopfield		
a_i	a_j	Δw_{ij}	a_i	a_j	Δw_{ij}	a_i	a_j	Δw_{ij}
0	0	0	0	0	0	0	0	+
1	0	0	1	0	–	1	0	–
0	1	0	0	1	–	0	1	–
1	1	+	1	1	0	1	1	+

Figure 6: **Table of different correlational learning rules.** For each of the possible activation correlations from unit j to unit i, the corresponding change in weight, Δw_{ij}, from unit j to i is given. A '+' indicates the weight is increased (the connection is strengthened), a '–' indicates the weight is decreased (the connection strength is diminished), and '0' indicates no change is made in the connection weight between the units.

[5] In the design of some learning rules, a temporal element is sometimes added to the basic Hebbian premise. Rather than taking into account the activities of the units at the same instant, the activities are sampled at some displaced time interval (Klopf 1986; Kosko 1986).

$$\Delta w_{ij} = (2o_i - 1)(2o_j - 1)$$

where o_i and o_j are the activation values of individual units i and j.[6]

Using this rule, several memory states can be stored at the same time in the same network. During recall the network is allowed to settle from its initial state into a final resting state which will be the memory closest to the initial state, as described in section 2.3. The number of memory states allowed in a Hopfield Network before severe degradation of recall occurs is less than .15N, where N is the total number of units in the network (Hopfield 1982). If the memory states represented by vectors are orthogonal, then each individual memory will be perfectly recallable, provided the memory capacity is not exceeded. In practical situations, however, the desired states to be memorized are almost never orthogonal, resulting in a significant decrease in storage capacity.

This limitation on the number and types of memories which can be stored is a result of using a correlational learning rule. Since learning in this case is an additive process, the storage of more than one memory creates additional, unintentional memory states resulting from the linear superpositioning of the stored states. When the memory capacity is exceeded these "spurious states" dominate, and the recall of valid memories degrades significantly. But even when the absolute memory capacity is not exceeded, recall can be significantly influenced by spurious states when memories are very similar or nearly linearly dependent. In general, the greater the number of states learned and the less linearly independent they are, the less accurate will be their recall.

5.2. Error–Correcting Rules

The second class of learning rules are those that employ *error–correcting* techniques. This class includes some of the most popular rules currently being used in actual implementations (Hinton *et al.* 1984; Rumelhart *et al.* 1986; Sejnowski & Rosenberg 1986). The general approach in these rules is to let the network produce its own output in response to some input stimulus, after which an external teacher presents the system with the correct or desired result. If the network's response is correct, no weights need be changed (although some systems might use this information to further strengthen the correct result). If there is an error in the network's output, then the difference between the desired and the achieved output can be used to guide the modification of weights appropriately. Since these methods strive to reach a global solution to the problem of representing the function by taking small steps in the direction of greatest local improvement, they are equivalent to a *gradient descent* search through the space of possible representations. The problems with learning rules that perform a gradient descent search will be considered at the end of this section, after we first describe some specific examples.

5.2.1. The Perceptron

A simple example of an error–correcting rule is the *perceptron convergence procedure* (Rosenblatt 1962). This learning scheme is as following: 1) if the output of a unit is correct, nothing changes; 2) if its output is "on" when it should have been "off," then the weights from

[6] More complicated learning rules have been proposed which improve the storage and recall capacities of Hopfield–like networks. These rules, however, typically deviate from the correlational class of learning rules and often require that the memory states be presented all at once in batch mode (Denker 1986).

"on" input units are decreased by a fixed amount; 3) if the output was "off" when it should have been "on," then the weights from "on" input units are increased by a fixed amount. According to the perceptron convergence theorem (Minsky & Papert 1969), this learning rule is guaranteed to converge to a correct representation, if and only if the desired function is linearly separable. Recall that an ANS with a single layer of weights is restricted to linearly separable functions, so this constraint is inherent to the architecture of the network and not the learning rule itself.

5.2.2. The Delta Learning Rule

The *delta learning rule* modifies the perceptron learning procedure slightly (Widrow & Hoff 1960; Rumelhart *et al.* 1986). The general technique is the same as with perceptrons: the weight of a connection is increased or decreased according to which change will bring the unit's response closer to the desired value. In the delta rule, however, the magnitude of this change varies according to the degree of discrepancy between the desired and achieved states. Formally, the rule defines the change in weight between two units as follows:

$$\Delta w_{ij} = \eta \delta_i o_j ,$$

where η is a global parameter controlling the rate of learning, and δ_i represents the difference between the desired value of the ith unit and its actual value:

$$\delta_i = (o_i^{desired} - o_i^{actual}).$$

This approach has an advantage over the perceptron learning rule in that the magnitude of the change is proportional to the degree of error.[7] Thus, for small errors that occur when the representation is almost correct, only minor changes are made; for larger errors, which presumably indicate significant misrepresentations, the weight changes are correspondingly greater. This rule, like the perceptron convergence procedure, is also guaranteed to converge to a solution if one exists.[8] Unfortunately, the delta rule only works for networks without hidden units, and therefore is also restricted to mappings representable by linearly separable functions. A desirable solution would be to extend this technique to multi-layered networks. To accomplish this, however, requires the addition of hidden units, and therefore raises the question of what are the "correct" activation values for the hidden units used in the calculation of δ_{hidden} (the difference between the desired value of a hidden unit and its actual value)? This dilemma is the classic credit assignment problem: the external world does not specify what the correct state of a hidden unit should be, and consequently the learning rule has no direct way to determine whether such units are positively or negatively contributing to the results.

[7] This proportional modification to weights is relevant only to systems with continuous or multi-valued activations. In the binary case δ_i is always one or zero.

[8] Hampson & Volper (1987) describe three similar rules that are guaranteed to converge.

5.2.3. Generalised Delta Rule

The *generalized delta rule* is an extension to the delta learning rule which overcomes the credit assignment problem and enables effective learning in multi–layered networks. This rule, devised by Rumelhart, Hinton, and Williams (Rumelhart *et al.* 1986), works by "propagating" errors back from the output units through to the internal hidden units. The credit assignment problem is solved by calculating the error of an individual hidden unit as the proportional summation (relative to the weights) of the errors of the units it feeds into. Thus, by working backwards from the observed errors in the output units, the errors of the underlying layers of hidden units can be calculated recursively.

The updating of weights follows according to the standard delta rule:

$$\Delta w_{ij} = \eta \delta_i o_j .$$

In the generalized case, however, the value of δ_i differs depending upon the type of unit. For output units, δ_i is similar to the δ_i of the standard rule, differing only by a factor of the derivative of the activation function. (For this reason the generalized delta rule requires that the system use an activation function that has a continuous finite derivative.) Hidden units, however, are handled in a more complicated manner involving both the derivative of the activation function and the summation of the δ's from the next higher level units, multiplied by the respective weights. Formally:

$$\delta_i = f'(net_i) \sum \delta_k w_{ki},$$

where $f'(net_i)$ is the derivative of the activation function with respect to its total input, net_i, and the δ_k's are the errors of the units receiving input from the ith unit. This recursive function of δ's is implemented in practice by first calculating activation values as described above, computing the changes to output units, and then using the output units' δ's to calculate the error values for the next lower level of hidden units, and so on for all units.

The generalized delta learning rule is a significant achievement because it provides an effective method for learning in multi–layered networks. This rule has been successfully used to learn non–linearly separable functions such as XOR, parity, and symmetry. Impressive results have also been realized in the NETtalk system (Sejnowski & Rosenberg 1986) which learns to read English text aloud. NETtalk begins with no prior knowledge of word pronunciation and gradually learns to read words of text verbally through exposure to free speech and words in a dictionary. One disadvantage with this rule (as with other gradient descent models) is that it does not scale well: what works well for a relatively small number of units, becomes exponentially expensive as the number of connections increases beyond a few hundred. Furthermore, the generalized delta rule requires additional space to record the states of all units during the settling of the network, as well as storage for all of the δ's used during the back propagation of error. We will return to these issues in section 5.2.5.

5.2.4. The Boltzmann Machine

An alternative to the generalized delta learning rule is found in the Boltzmann Machine (Hinton *et al.* 1984), a multi–layered network with a learning rule based on the statistical

mechanics of simulated annealing (Kirkpatrick *et al.* 1983). The purpose of this system is to construct an internal representation of the external world as a probability distribution of events. It learns relationships between inputs and outputs which reflect their relative occurrence in the environment.

The *Boltzmann learning rule* is rather complicated, involving the collection of statistical information on the system's "clamped" and "unclamped" equilibrium states. An event, represented as an input/output vector pair, is presented to the system and the external units are "clamped" (i.e., held in their initial states). The hidden units of the network are allowed to settle to equilibrium through simulated annealing, and then statistics are collected on how often pairs of units are both "on" over a given number of cycles. The system is then allowed to run freely, "unclamped," and again statistics of correlated activity are recorded at equilibrium. The two statistical values for each pair of units are subtracted and the resulting sign of this number is used to either increment or decrement the pair's (symmetric) weights by a fixed amount.[9] This process is repeated several times on a representative set of events. Following training, when the system is later presented with an input vector, the Boltzmann Machine reproduces the most likely output response based on its generalized internal representation of the world.

The Boltzmann Machine learning rule provides an effective method for learning the weights of hidden units and as such, is capable of learning complex, nonlinearly separable mapping functions. A major criticism of the Boltzmann learning rule (and of simulated annealing processes in general) is that it is extremely slow. An inherent requirement in the simulated annealing process is that the "cooling" of the system as it settles into equilibrium must take place slowly, otherwise it is liable to become caught in local minima. In addition, the Boltzmann learning procedure is basically a gradient descent method and thus, like other systems employing gradient descent, requires exposure to a considerable number of examples in order for it to learn the desired relationships. For even a relatively simple problem such as a 4–unit to 4–unit decoder (8 units total), the learning time can exceed 1800 learning cycles, with each cycle requiring 40 activation state evaluations (Hinton *et al.* 1984).

5.2.5. Summary of Error–Correcting Rules

After viewing several specific error–correcting rules, what can we say about them in general with regard to learning limitations or biases? First, error correcting rules are inherently gradient descent algorithms. Gradient descent (or hill climbing) techniques are not guaranteed to find global solutions, and may instead encounter and become stuck in local minima (maxima), ravines (ridges), etc. An ANS learning rule based on these methods might therefore render it impossible for the system to learn a function which it is fully capable of representing within its network architecture. In general, error–correcting rules work well only on problems in which the representation space is relatively smooth and free of local minima. Interestingly, however, Rumelhart *et al.* (1986) report from empirical results that "the apparently fatal problem of local minima is irrelevant in a wide variety of learning tasks."

Even if it is the case that most learning problems occur in well behaved representational spaces, error–correcting rules have other drawbacks (Sutton 1987). For incremental hill climbing to be effective in these rules, relatively small steps need to be taken. The implication is that the

[9] Although this learning rule is Hebbian–like in its use of correlated activities, it is classified as error–correcting because it makes use of the the difference between expected and actual correlated activity to determine in which direction to update the weights.

network will likely require repeated exposure to multiple training events. In practice this tends to be the norm, making for slow learning with long convergence times. Furthermore, error–correcting rules are sensitive to variations in the order of presentation of examples. If a network initially encounters a "bad" set of training examples, it could be led off into some undesirable part of the representational space from which it might never return. Together, these negative qualities have the potential for making an error–correcting learning rule ineffective or impractical for some problems. Whether a given learning problem can be solved with one of these rules is an issue often resolved only through experimentation.

5.3. Unsupervised Learning

The third form of learning in ANS's involves self–organization in a completely unsupervised environment. In this category of learning rules, the focus is not on how actual unit outputs match against externally determined desired outputs, but rather on adapting weights to reflect the distribution of observed events. The "competitive learning" scheme proposed by Rumelhart and Zipser (1986) offers one approach toward this end. In this model, units of the network are arranged in predetermined "pools," in which the response by the units to input patterns is initially random. As patterns are presented, units within the pool are allowed to compete for the right to respond. That unit which responds strongest to the pattern is designated the winner. The weights of the network are then adjusted such that the response by the winner is reinforced, making it even more likely to identify with that particular quality of the input in the future. The result of this kind of competitive learning is that over time, individual units evolve into distinct feature detectors which can be used to classify the set of input patterns.

5.3.1. Self–Organising Feature Maps

The "self–organizing feature maps" of Kohonen (1986) demonstrate a similar form of unsupervised learning. Kohonen's goal was to provide some insight into how sensory pathways, terminating in the cerebral cortex of the brain, often topologically map aspects of physical stimuli. Although this can be explained in part by the predetermined axonal growth patterns that are genetically encoded, it appears that some of this organization arises dynamically from learning itself. Kohonen provided one possible explanation by proposing learning rules which promote just this type of topologically–directed self–organization. In his model, weights of the network gradually come to represent qualities or "features" of the input stream, in such a way that topologically close units within the network respond similarly to similar input examples. In addition, response by the network as a whole is spread out in a way which most effectively covers the probability distribution of the feature space. These "feature maps" can best be explained by an example.

In the Kohonen model, a layer of input units is totally connected to a higher layer of output units. Each output unit is assigned a weight vector, W, that corresponds to the weights from the input vector to that unit. This vector represents the point in the feature space to which the unit is best "tuned," i.e., most sensitive to input. An input vector that is identical or close to a unit's weight vector will excite that unit in proportion to its similarity. As in competitive learning, Kohonen's learning rule first requires identifying that unit whose response best matches the input. When an input vector O of N dimensions is presented during learning, a distance measure, d_j, is calculated for each unit indicating how close its weight is to the input. The equation for this measure is:

$$d_j = \sum_{i=0}^{N-1} (o_i - w_{ij})^2.$$

The unit with minimum distance is selected as the winner, and its weight vector is modified to make the unit more sensitive to the input vector O. Associated with each output unit is a "neighborhood" of nearby units defined by a radius that slowly decreases in size over time. When a unit wins the competition and has its weights modified, the weight vectors of units in the immediate neighborhood are similarly modified, drawing them closer to the input pattern as well.[10] More formally, the weight modification rule is as follows:

$$\Delta w_{ij} = \begin{cases} \alpha (o_i - w_{ij}) & \text{for } i \in N_c \\ 0 & \text{for } i \notin N_c \end{cases}$$

where N_c is the winning unit's neighborhood having radius c.

The result of this learning rule is that over time, specific output units within the network learn to respond to particular qualities of the input stream. In addition, neighboring units are also pulled toward a similar response, causing units of the network to becoming topologically ordered with respect to features in the input stream. Thus, unsupervised learning systems of this type do not learn in the sense of finding a representation of a particular mapping function. Instead, the intention of these systems is to cluster the event space into regions of high input activity, and assign units to respond selectively to these regions. Promise for this approach appears greatest in the development of hierarchical systems consisting of several layers of competitive clusters (Rumelhart *et al.* 1986).

5.3.2. The Hedonistic Neuron

The unsupervised learning schemes can be implemented using only local information about the activities of individual units. This is accomplished by using a combination of excitatory and inhibitory connections between output units (Kohonen 1984). In a sense, each unit can be viewed as seeking to maximize its own activity. This localization of purpose is the basis for the notion of the "hedonistic" neuron (Klopf 1982). Klopf suggests the best approach to learning may be through the use of "self–interested" units seeking to satisfy themselves. In recent experiments, Klopf (1986) has shown how a "drive–reinforcement" model of learning for individual units can reproduce the classic S–shaped learning curve. Pursuing related ideas, Barto and Sutton (1985) have shown that practical short– and long–term learning can be achieved through the use of simple "goal–seeking" units. Their systems demonstrate learning remarkably similar to classical conditioning observed in animals.

[10] Some formulations augment the primary input with lateral feedback between units. In this case, *lateral inhibition* is brought to bear in an inhibitory penumbra surrounding the immediate neighborhood of mutual excitation. In areas beyond the inhibitory region, slight amounts of excitation may again be found, forming what is sometimes referred to as the "Mexican–hat function."

5.3.3. Summary of Unsupervised Learning Rules

It is difficult to make general comments on the types of problems that can be solved with unsupervised learning rules, such as the ones made for the other two classes of rules. One observation is that for the system to produce useful clusters it is necessary that the features describing the input space be appropriately selected; otherwise useful clusters may not exist in the event space. But this is a problem that holds for any learning rule, and the intention of unsupervised learning schemes is merely to identify regularity wherever it exists. Learning rules that use a measure of the difference between input and weight vectors, such as competitive learning, behave similarly to gradient descent search; they are thus plagued by many of the same problems as hill climbing. In particular, the changes made to the weights in these cases must be small, otherwise the units may jump around sporadically in the feature space. Small steps imply the need for large training sets, which in practice may amount to thousands of input examples. Kohonen and others attempt to solve this problem by beginning with large readjustment increments and gradually decreasing them with experience. The problem of local minima does not appear be as severe in these systems. At least part of the reason can be attributed to there being no "correct" solution to which these systems are expected to converge.

6. Summary

This paper reviewed several issues confronting learning in artificial neural systems. In particular, we examined two central concerns, the nature of learnable representations within ANS's designs, and the different forms of learning rules used to encode representations within the connective weights of a network. The choice of a network's architecture imposes certain inherent biases which bear directly on the kinds of functions that can ultimately be represented within the network. The most general form of architecture — a completely connected network — was shown to be able to represent any arbitrary input/output function (Minsky & Papert 1969). The expense of such generality, however, was the inefficiency and often unnecessary complexity of computation. Consequently, most network architectures impose limitations on the patterns of connectivity allowed between units of the network. Limiting connectivity, however, can result in reduced representational capabilities. For example, feedforward networks composed of a single layer of weights are constrained to represent only linearly–separable functions. Two–layered networks increase the representational capacity of the network by allowing for internal representations to be formed within hidden units. Such networks are theoretically capable of representing functions mapping input patterns to any convex region in feature space. For Boolean functions this class is complete (Volper & Hampson 1986). Networks composed of three–layers of intervening weights are unconstrained and therefore able to represent any arbitrary mapping function (Hanson & Burr 1987; Lippmann 1987).

A learning rule is an algorithm used to modify weights in an ANS for the purpose of acquiring an appropriate mapping function. Hebb's correlational postulate of synaptic modification is the basis for many learning rules, in particular the correlational rules. Hopfield Networks and Kohonen's feature maps exhibit the useful pattern recognition and completion capabilities of these rules. The error correcting rules differ from correlational rules in the use of gradient descent techniques, and have proven to be effective on many problems. They are subject, however, to the inherent limitations of hill climbing methods: becoming caught in local minima, sensitivity to the input presentation order, and characteristically slow learning caused by the need to make small incremental steps over a large number of input presentations. Unsupervised learning rules provide a recent alternative. These approaches employ adaptive learning techniques which cause

the units to become selective receptors of distinct attributes on the space of inputs. Unsupervised learning methods break away from the more common ANS notion of learning a specific mapping function, providing instead recharacterizations of the feature space based on detected regularity of activity. These rules are also interesting for the similarity they bear to different types of learning observed in animals.

New architectures and learning rules continue to be proposed. With the sudden surge of interest in Artificial Neural Systems, new developments will undoubtedly continue for some time. What is now perhaps needed more than novel approaches are further theoretical results concerning the inherent constraints and biases of these systems. A clearer understanding of the general capabilities and limitations of ANS's would provide valuable guidance for research in this field.

7. Acknowledgements

The authors would like to thank Larry Rendell, John Collins, Raymond Board, Bartlett Mel, Peter Haddawy, and Betty Cheng for their careful critiques and thoughtful suggestions.

8. References

Anderson, J. A. 1977 "Neural models with cognitive implications." In: *Basic Processes in Reading*, LaBerge & Samuels, ed. Lawrence Erlbaum Associates, Hillsdale, New Jersey, pp. 27–90.

Barto, A. G., and Sutton, R. S. 1985 "Neural problem solving." In: *Synaptic modification, neuron selectivity, and nervous system organization*, Anderson Levy Lehmkuhle, ed., Hillsdale, New Jersey, pp. 123–152.

Carpenter, G. A., and Grossberg, S. 1986 "Neural dynamics of category learning and recognition: attention, memory consolidation, and amnesia." In: *Brain Structure, Learning, and Memory*, R. Newburgh J. Davis and E. Wegman, ed. AAAS Symposium Series.

Denker, J. S. 1986 "Neural network models of learning and adaptation." *Physica 22D*, pp. 216–232.

Egecioglu, O., Smith, T. R., and Moody, J. 1986 "Computable functions and complexity in neural networks." *University of California Santa Barbara, pre-print.*

Fukushima, K. 1986 "A neural network model for selective attention in visual pattern recognition." *Biological Cybernetics*, Vol. 55, pp. 5–15.

Hampson, S. E., and Volper, D. J. 1986 "Linear function neurons: structure and training." *Biological Cybernetics*, Vol. 53, pp. 203–217.

Hampson, S. E., and Volper, D. J. 1987 "Disjunctive models of Boolean category learning.ρq *Biological Cybernetics*, Vol. 56, pp. 121–137.

Hanson, S. J., and Burr, D. J. 1987 "Knowledge representation in connectionist networks." *Bell Communications Research pre-print.*

Hebb, D. O. 1949 *The Organization of Behavior.* Wiley, New York.

Hinton, G. E. 1984 "Distributed representations." *Carnegie-Mellon University Technical Report CMU-CS-84-157.*

Hinton, G. 1986 "Learning in massively parallel nets." *Proceedings of International Conference on Artificial Intelligence*, p. 1149.

Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. 1984 "Boltzmann machines: constraint satisfaction networks that learn." *Carnegie-Mellon University Technical Report CMU-CS-84-119.*

Hopfield, J. J. 1982 "Neural networks and physical systems with emergent collective computational properties." *Proceedings of the National Academy of Science USA* (April), Vol. 79, pp. 2554–2558.

Hopfield, J. J., and Tank, D. W. 1986 "Computing with neural circuits: a model." *Science* (August), Vol. 233, pp. 625–632.

Kirkpatrick, S., Gelatt, C. D. Jr., and Vecchi, M. P. 1983 "Optimization by simulated annealing." *Science* (May), Vol. 220, pp. 671–680.

Klopf, A. H. 1982 *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence.* Hemisphere Publishing.

Klopf, A. H. 1986 "A drive-reinforcement model of single neuron function: an alternative to the Hebbian neuronal model." *AIP Conference Proceedings 151, Neural Networks for Computing* (April).

Kohonen, T. 1977 *Associative Memory: A System Theoretical Approach.* Springer, New York.

Kohonen, T. 1984 *Self-Organizing and Associative Memory.* Springer-Verlag, Berlin.

Kosko, B. 1986 "Differential Hebbian learning." *AIP Conference Proceedings 151, Neural Networks for Computing*, pp. 277–282.

Levy, W. B., and Desmond, N. L. 1985 "The rules of elemental synaptic plasticity." In: *Synaptic modification, neuron selectivity, and nervous system organization*, Anderson Levy Lehmkuhle, ed. Lawrence Erlbaum Associates, pp. 105–121.

Lippmann, R. P. 1987 "An Introduction to computing with neural nets." *IEEE ASSP Magazine* (April), pp. 4–22.

McCulloch, W. S., and Pitts, W. 1943 "A logical calculus of the ideas immanent in nervous activity." *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115–133.

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

Minsky, M., and Papert, S. 1969 *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA.

Muroga, S. 1971 *Threshold Logic and Its Applications.* Wiley, New York.

Omohundro, S. M. 1987 "Efficient algorithms with neural network behavior." *Complex Systems Journal*, Vol. 1, Issue 2, pp. 27–348.

Rendell, L. A. 1985 "Substantial constructive induction using layered information compression: tractable feature formation in search." *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pp. 650–658.

Rendell, L. A. 1986 "A general framework for induction and a study of selective induction." *Machine Learning*, Vol 1, Issue 2, pp. 177–226.

Rosenblatt, F. 1962 *Principles of Neurodynamics.* Spartan.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986 "Learning internal representations by error propagation." In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol 1*, D. E. Rumelhart & J. L. McClelland, ed. MIT Press, Cambridge, pp. 318–362.

Rumelhart, D. E., and McClelland, J. L. 1986 *Parallel Distributed Processing: Explorations in the Microstructures of Cognition Vol. 1.* MIT Press, Cambridge, MA.

Rumelhart, D. E., and Zipser, D. 1985 "Feature discovery by competitive learning." *Cognitive Science*, Vol. 9, pp. 75–112.

Sejnowski, T. J., and Rosenberg, C. R. 1986 "NETtalk: a parallel network that learns to read aloud." *Johns Hopkins University Electrical Engineering and Computer Science Technical Report JHU/EECS–86/01.*

Sutton, R. S. 1986 "Two problems with backpropagation and other steepest–descent learning procedures for networks." *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, pp. 823–831.

Touretzky, D. S. 1986 "Representing and transforming recursive objects in a neural network, or 'Trees do grow on Boltzmann Machines.'" *Proceedings of the 1986 Conference on Systems, Man, and Cybernetics* (October).

Volper, D. J., and Hampson, S. E. 1986 "Connectionistic models of boolean category representation." *Biological Cybernetics*, Vol. 54, pp. 393–406.

Widrow, B., and Hoff, M. E. 1960 "Adaptive switching circuits." *1960 IRE WESCON Convention Record, Part 4*, (August), pp. 94–104.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-87-1394	2.	3. Recipient's Accession No.
4. Title and Subtitle LEARNING IN ARTIFICIAL NEURAL SYSTEMS			5. Report Date December 1987 6.
7. Author(s) Christopher J. Matheus and William E. Hohensee			8. Performing Organization Rept. No. R-87-1394
9. Performing Organization Name and Address University of Illinois Dept. of Computer Science 1304 W. Springfield Avenue Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address			13. Type of Report & Period Covered Technical 14.

15. Supplementary Notes

16. Abstracts

This paper presents an overview and analysis of learning in *Artificial Neural Systems* (ANS's). It begins with a general introduction to neural networks and connectionist approaches to information processing. The basis for learning in ANS's is then described, and compared with classical machine learning. While similar in some ways, ANS learning deviates from tradition in its dependence on the modification of individual *weights* to bring about changes in a knowledge representation distributed across connections in a network. This unique form of learning is analysed from two aspects: the selection of an appropriate network architecture for representing the problem, and the choice of a suitable learning rule capable of reproducing the desired function within the given network. The various network architectures are classified, and then identified with explicit restrictions on the types of functions they are capable of representing. The learning rules, i.e., algorithms that specify how the network weights are modified, are similarly taxonomised, and where possible, the limitations inherent to specific classes of rules are outlined.

17. Key Words and Document Analysis. 17a. Descriptors

artificial neural systems
neural networks
connectionism
learning

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
unlimited	UNCLASSIFIED	27
	20. Security Class (This Page) UNCLASSIFIED	22. Price