CMU-CS-86-131

CONSENSUS : A Statistical Learning Procedure In a Connectionist Network

Gordon J. Goetsch
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pa. 15213

May, 1986

Abstract

We present a new scheme for the activity of neuron-like elements in a connectionist network. The CONSENSUS scheme is based on statistical inference. The guiding principle of CONSENSUS is that decisions should be deferred until sufficient evidence accumulates to make an informed choice. Consequently, large changes in network structure can be made with confidence. Nodes have an awareness of their role and utility in the network which allows them to increase their effectiveness. The reinforcement scheme utilizes the notion of confidence so that only nodes proven to contribute successfully issue reinforcements. Nodes are grouped into communities to exploit their collective knowledge which exceeds any individual member. The network was tested against several problems and was able to find suitable encodings to solve them.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, monitored by the Air Force Avionics Laboratory Under Contract F33615-84-K-1520.

Table of Contents

List of Figures

List of Tables

1. Introduction

Knowledge about the architecture of the brain has lead researchers to investigate the properties of "connectionist" systems. These systems are characterized by simple neuron-like processing elements which are interconnected and store their knowledge as the strengths of these connections. There are tasks for which these networks are eminently suited such as pattern recognition. With the proper connections and connection strengths, the processing elements can compute in parallel. For a specific task, determining the proper interconnections and strengths to utilize the computational power of the network is straightforward. For more general problems, we may not be able to determine the proper configuration of the network. Preferably, the network would be able to dynamically reconfigure itself to solve the task presented to it.

To reconfigure itself a program must gain an understanding of the task, in otherwords, it must learn. For a large domain rote memorization will be impractical since it requires exhaustively enumerating the domain. To complete the task without rote memorization, the network must understand some concepts about the domain. To learn autonomously, no outside agent may communicate to the network any concepts about the domain. Learning by example, involving a tutor presenting domain examples to the network and indicating the desired response, meets this criteria. This means that the network must be able to formulate its own concepts about the domain based upon the examples presented to it. The network must initially recognize features from the domain, then learn to utilize the features it has learned to recognize. Such learning would be evolutionary with improving performance as opposed to instant learning. A network utilizing domain feature recognition should be able to infer domain examples to which it has not been previously exposed. We will concentrate on learning-by-example.

Initial studies of simple networks learning to classify input vectors were encouraged by the perceptron convergence procedure. The perceptron convergence procedure detailed a method by which a one layer network could reach its full potential as a computing element. However, most interesting computations require more than one layer. In 1969, in *Perceptrons* [19], Minsky and Papert showed that no convergence procedure can exist for multi-layered networks. The lack of a proven convergence procedure does not mean that neural networks are incapable of interesting computations.

The search for effective convergence procedures continue, many of them with a mathematical bent. Recent work includes the Boltzmann machine [12], which relies on stochastic relaxation techniques to achieve convergence. Rather than attempt to prove convergence properties for a method, we will be satisfied by a convincing empirical demonstration of the ability of a method to achieve convergence.

This paper describes the CONSENSUS system, a connectionist network. CONSENSUS is an acronym for CONtext SENsitive NetworkS Using Statistics. The network discovers inherent environmental constraints by being presented with examples from the domain by a tutor. The network modifies its interconnections and programming to capture the underlying constraints from the domain. In this manner, the network learns the tutor's classification method from the examples. Experiments show that the network is able to learn in simple domains.

The downfall of many learning methods has been the "credit-assignment" problem. A learning method must be able to determine how to modify its parameters in order to improve its understanding of the task. Most previous work depends on making many small modifications with little confidence in each change or its effect on the network. To improve performance they rely on the net accumulated change of the many small modifications. Instead, we attempt to make a few large changes with good confidence. The guiding principle of CONSENSUS is that decisions should be deferred until sufficient evidence accumulates to make an informed choice. The CONSENSUS system attempts to do this by giving each node an understanding of the role of itself and its neighbors in the network. The decision method of CONSENSUS is based on probability theory and the statistics of group action.

The most distinguishing features of the CONSENSUS system are :

1. *The use of statistical inference for the classification method.* Changes are made when sufficient statistical evidence has accumulated to justify the change with a high degree of confidence.

2. *Large changes are made with confidence.* The network makes a few large changes with a high degree of confidence in contrast to more conventional systems which rely on the accumulation of many small changes each made with little confidence.

3. *Nodes have an awareness of their role and utility in the network.* It is advantageous for nodes to understand their role in the network so that they may perform that role more effectively.

4. *The use of the notion of confidence in the reinforcement scheme.* Only nodes that are proven to contribute to the success of the network may issue reinforcements, which aids in the translation of global reinforcements into local reinforcements.

5. *The grouping of nodes into communities to exploit their collective knowledge.* The collective knowledge of a group of nodes can exceed that of any of its members.

Section 2 describes the task domain, and Section 3 contains a description of the network components. Experimental results are presented in Section 4. Section 5 discusses several aspects of the network, and Section 6 contains concluding remarks.

2. The Domain

2.1 Overview

The learning task involves learning-by-example with immediate feedback by a tutor. The network is presented with a series of examples from the domain which it must classify. The proper classification is strictly a function of the current example, no other dependencies including temporal dependencies are allowed. The tutor gives immediate feedback to the network for each example, delayed feedback is not allowed. The task is a simple one, but has wide applicability. Many more complicated problems can be recast in the form of this domain.

2.2 The Learning Task

The task to be performed involves classifying binary vectors. The network must classify each example presented to it as either VALID or INVALID. The tutor, assumed to be infallible, categorizes each example as VALID, INVALID, or UNCERTAIN. If the tutor classifies an example as VALID the network must also classify it as VALID. If the tutor classifies an example as INVALID the network must also classify it as INVALID. If the tutor classifies an example as UNCERTAIN, then the network may classify it as VALID or INVALID. The tutor examines the network's classification and advises the network of the accuracy of its classification. The tutor rewards the network for a correct (matching) response and punishes it for an incorrect response when a VALID or INVALID was presented. The tutor neither rewards nor punishes the network when an UNCERTAIN example was presented. This illustrated in Table 2-1. The objective is for the network to learn to correctly recognize VALID and INVALID examples from the domain solely as a result of the tutor's advice.

Input Vector			Tutors	Networks Desired
X	Y	Z	Classification	Classification
0	0	0	Uncertain	Either
0	0	1	Invalid	Invalid
0	1	0	Invalid	Invalid
0	1	1	Invalid	Invalid
1	0	0	Invalid	Invalid
1	0	1	Valid	Valid
1	1	0	Valid	Valid
1	1	1	Valid	Valid

Table 2-1: Sample Problem

2.3 The Difficulty of the Task

The difficulty of any given problem is a function of the size of the binary input vector. A problem with binary input vectors *n* elements in length could present the network with up to 2^n distinct input vectors, since each of *n* inputs could assume 2 distinct states, ON or OFF. The network must be able to make an adequate classification of each potential input vector. If the number of potential vectors is *m* ($= 2^n$), then the network has 2^m possible ways to classify the set of possible input vectors since each of *m* input vectors could be classified in 2 ways, VALID or INVALID. A domain with input vectors of length 10, could generate up to 1024 distinct input vectors, and force the network to choose the correct classification function from among up to 2^{1024} different possible functions. The double exponential nature of the problem makes it extremely difficult to solve complex problems.

In practice, the problems are usually simpler. The network may not be presented with all of the potential input vectors. Some of the input vectors presented may be UNCERTAIN in which case any classification by the network is acceptable. The worst case assumes that there are no inherent regularities in the problem. Most interesting problems have some inherent regularities. Recognizing and representing these regularities is essential to solving problems quickly and efficiently.

A network capable of correctly classifying binary input vectors has the capability to classify arbitrary input vectors. Any input value, such as reals or integers, can be replaced by one or more binary input values, therefor the binary nature of the domain is not an inherent restriction on its capabilities.

3. The Network

3.1 An Overview

CONSENSUS is a connectionist network capable of learning from examples. The network consists of nodes which are organized into communities which are in turn organized hierarchically into layers. Nodes are interconnected by links which allow the nodes to communicate with each other. Communities are groups of nodes which are monitored by a distinguished node that reflects their collective judgement. A conceptual view of the network is shown in Figure 3-1. The following section gives an overview of the network components and subsequent sections describe the network in detail.

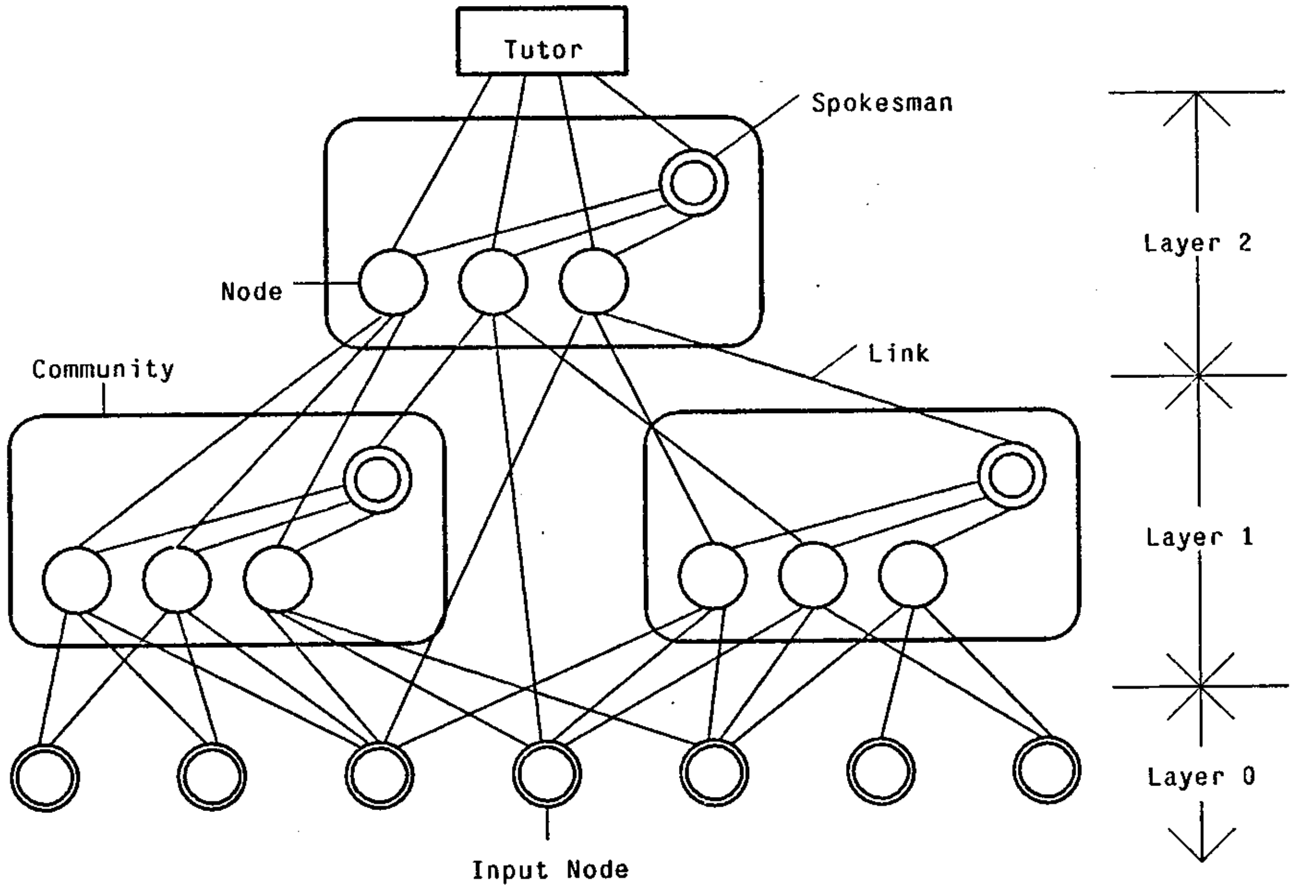

Figure 3-1: Conceptual View of a Network

3.2 The Network Components

3.2.1 Nodes

The nodes are the primitive computing elements in the network. Each node controls three links with which it can connect and communicate with other nodes. For convenience these controlled links will be referred to as the X link, the Y link, and the Z link. An arbitrary number of uncontrolled links may be linked to a node. A node is always in one of two states, ON or OFF, which is a deterministic pairwise boolean function of the states of two other nodes with which its controlled links are connected. Every node has a unique identifier and maintains a history summarizing the events which it has observed. These are conventional nodes, there are some specialized nodes in the network, but the term nodes will be use to refer to conventional nodes unless otherwise indicated. The operations of the network center on the conventional and specialized nodes and the following sections describe the operations in detail.

3.2.2 Links

Links allow communication between nodes. A link is a directed connection between two nodes. A link has a SUPERIOR end and a SUBORDINATE end which are distinguishable. The superior node resides in the same layer or a higher layer in the hierarchy than the subordinate node, and the superior node determines which node shall be the subordinate node. The link can transmit the following information from the subordinate node to the superior node :

- Subordinate node state.
- Subordinate node identifier.
- New-Function signal.

The link can transmit the following information from the superior node to the subordinate node :

- Reinforcement signal.
- Confidence-Level.

The superior end is permanently connected to a single node while the subordinate end may be connected to different nodes at different times.

3.2.3 Communities

A community is a collection of nodes that function together to share their understanding of the environment to which they are connected. The community consists of many conventional nodes and one spokesman node. Each community resides exclusively in one layer.

3.2.4 Spokesman

A spokesman is a distinguished node. It controls one link to every other node in its community which is fixed and may not be reconnected to other nodes. The state of the spokesman is a deterministic function of the states of the other nodes in the community. The state of the spokesman represents the collective judgement of the nodes of the community. Henceforth, the term spokesman is meant to be spokesman node.

3.2.5 Layers

Layers are composed of one or more communities. The network is organized hierarchically and there may be an arbitrary number of layers. The binary input vector is considered to be layer 0, the remaining layers are numbered from 1 to n. Communities in the uppermost layer are referred to as top communities, while any community that may connect to the inputs is referred to as a base community.

3.2.6 Environmental Inputs

The binary input vector from the environment is communicated to the network via input nodes. These are specialized nodes, one per element in the vector, and always assume the state of the corresponding element in the binary input vector. These nodes reside in layer 0 by definition.

3.2.7 Environmental Reinforcement

The tutor conducts the environmental reinforcement. The environmental reinforcements are sent to every node in the top layer. These nodes may generate internal reinforcements which can propagate downwards throughout the network.

3.3 Cycle Timing

Each cycle consists of the phases enumerated below. A central clock ensures that the nodes are properly synchronized.

1. Environmental Input Received. The input nodes assume the state of the corresponding element of the binary input vector.

2. Computation. Each node computes its new state after its subordinates have assumed their new states. Its new state is then available to its superiors so they may compute their new state. All nodes may compute in parallel subject to propagation delays.

3. Network Classification. Spokesman located in the top communities classify the binary input vector on behalf of the network.

4. Environmental Reinforcement. Nodes located in the top communities receive reinforcement from the tutor.

5. Internal Reinforcement. The nodes in the top communities issue reinforcement signals to their subordinates. The subordinates in turn issue reinforcements to their subordinates. Reinforcement may go on in parallel throughout the network subject to propagation delays.

6. Analysis. Each node analyzes the functions it could compute, and the usefulness of the links it controls. It determines if it should retain or change its current computed function, and which links, if any, should be replaced this cycle. All nodes may analyze in parallel.

7. Unlinking. Links designated for replacement are unlinked from their subordinate nodes.

8. Changing Functions. Nodes desiring to change their computed function now do so informing their superiors of their action.

9. Linking. Links without connections to a subordinate node are now reconnected to new nodes.

3.4 Receiving Inputs from the Environment

The environmental binary input vector is made available to the network in the Environmental Input Received Phase. Each input node assumes the state of a specified element, which remains the same from cycle to cycle, in the binary input vector. The states of these nodes are now available to the remainder of the network.

3.5 Computing the Output of a Node

A node outputs one of the sixteen pairwise boolean functions of two of the three links it controls. The node could compute any of the 38 unique pairwise boolean functions enumerated in Table 3-1 subject to the restriction that only top community nodes may compute a constant. It could compute a function of none of its links (a constant), a function of one of its links, or a function of two links. Only half of the functions are independent since every signal has exactly one inverse. The node remembers which function it is to output. During the Computation Phase the node assumes the state of its output.

3.6 Computing the Output of a Spokesman

The spokesman node of each community outputs the state of the majority of the other nodes in the community. The spokesman samples their states through the links it controls and determines which state is in the majority, breaking ties arbitrarily if needed. In this way, the node represents the collective judgement of the nodes in the community.

Functions of 0 inputs :

FALSE	TRUE

Functions of 1 input :

X	~X	Y	~Y	Z	~Z

Functions of 2 inputs :

X or Y	X nor Y	Y or Z	Y nor Z	Z or X	Z nor X
X or ~Y	~X and Y	Y or ~Z	~Y and Z	Z or ~X	~Z and X
~X or Y	X and ~Y	~Y or Z	Y and ~Z	~Z or X	Z and ~X
X eqv Y	X xor Y	Y eqv Z	Y xor Z	Z eqv X	Z xor X
X and Y	X nand Y	Y and Z	Y nand Z	Z and X	Z nand X

Table 3-1: Unique Pairwise Boolean Functions

3.7 Computing the Output of the Network

The output of the network is taken to be the state of the spokesmen of the top communities. If the state of the spokesman is ON, the network is said to have classified the example as ACCEPTABLE. If the state of the spokesman is OFF, the network is said to have classified the example as REJECTABLE.

3.8 The Tutor

The function of the tutor is to issue the appropriate environmental reinforcement to the network. The tutor can be thought of as a single node residing in the $n+1$ layer. The tutor has a fixed controlled link to every node of every top community through which it may reinforce these nodes.

3.9 Receiving Environmental Reinforcement

The tutor, presumed to be infallible, classifies the binary vector input as either ACCEPTABLE, REJECTABLE, or NEUTRAL. It issues a reinforcement signal to every node in the top communities. The state of each node in the top community is examined and compared to the desired classification. The tutor then issues reinforcement as described previously and illustrated in Table 2-1.

3.10 The Classification Method

3.10.1 Introduction

Consider the following problem. You are given a coin and asked to classify the coin as FAIR, a HEAD, or a TAIL. A HEAD coin comes up heads more often than tails, a TAIL coin comes up tails more often than heads, and a FAIR coin comes up heads and tails with equal frequency. How can you determine which category the coin belongs to if you are only allowed to test the coin by flipping it and recording the result?

Determining the correct classification with absolute certainty is impossible. Unless the coin always produces heads or tails, then any given flip could result in a head or a tail. A coin that produces both heads and tails could belong to any of the three categories. Any sequence of observed results *could* be produced by a coin from any of the three categories. This makes it impossible to rule out any of the three categories with absolute certainty.

Since the coin cannot be classified with certainty, we must settle with flipping the coin until a good guess as to which category it belongs in can be made. We could satisfy ourselves that the coin is a HEAD if a sequence was observed which was very likely for a HEAD coin but very unlikely for a FAIR or TAIL coin. The converse can be done to satisfy ourselves that the coin is a TAIL coin. Satisfying ourselves that a coin is a FAIR coin, is much more difficult. While a FAIR coin is equally likely to come up heads or tails, it is not assured in any particular number of trials. Indeed, a FAIR coin is far more likely to have an unequal number of heads and tails observed after a given set of trials. It would be expected that over a *very large* number of trials, the numbers of heads and tails would be approximately equal. We can categorize a coin as being as FAIR coin with confidence if we believe with confidence that it is not a HEAD coin and not a TAIL coin by process of elimination. This solution to the coin problem is based on statistical inference and can be placed on a more formal basis.

3.10.2 Statistical Inference

Statistical inference is the process of drawing conclusions about a population on the basis of a random sample. Alternative hypotheses are classified by *hypothesis testing*. In the solving the coin problem we may propose the hypothesis "the unknown coin is a HEAD coin." We generate a sample by flipping the coin and observing the result. Hypothesis testing is a general method for determining whether to accept or reject the proposed hypothesis about a random variable from information in the random sample [1].

3.10.3 Classifying the Coin

In the case of the coin problem, we are asked to categorize the coin as either FAIR, HEAD, or TAIL. If an insufficient number of trials to make a determination has been observed, we can claim that the proper classification is *unknown*.

To classify the coin as a HEAD it must be shown that it is unlikely to be a FAIR or a TAIL coin. A FAIR or a TAIL coin will have heads come up no more often than tails. In a sample of n trials, we will observe h heads and t tails such that $h + t = n$. If 100 trials of 1000 coin flips each were conducted, we would expect a FAIR coin to produce between 460 and 540 heads inclusive in 99 of the trials. This range from 460 to 540 constitutes the 99% *confidence interval* (two tailed) which is defined to be the range that is expected to encompass 99 out of 100 trials. If more than 540 heads are observed, we can claim that the coin is a HEAD. If it is actually a FAIR coin, the probability of our being wrong is .5%. If it is actually a TAIL coin, the probability of our being wrong is less than .5%. We accept a probability of error of .5% when we make a classify a coin as a HEAD coin.

Conversely, if the observed number of heads is less than 460, the coin can be classified as a TAIL coin with an error of .5% or less.

It should be emphasized that not classifying a coin as a HEAD or a TAIL is not equivalent to classifying it as a FAIR coin. A coin is classified as a HEAD or a TAIL when we have a preponderance of evidence in favor of that classification. We may or may not have sufficient evidence to classify a coin as a FAIR coin.

To prove that a coin is a FAIR coin by the above method would require an infinitely large number of samples. Since it is infeasible to prove that the coin is a FAIR coin, we will be content to show that the coin is fair to within an *equivalence factor*. If the equivalence factor is 5%, then we would consider a coin to be a FAIR coin if it can be shown that it will come up heads between 45% (50% - 5%) and 55% (50% + 5%) of the time. Showing that the coin will come up heads more than 45% of the time is analogous to categorizing the coin as a HEAD coin (comes up heads more than 50% of the time). The 99% confidence interval for a coin that comes up head 45% of the time is from 410 to 490 inclusive. So if more than 490 heads are observed, we are satisfied that the coin will come up heads more than 45% of the time. Analogously, the 99% confidence interval for a coin that comes up heads 55% of the time is from 510 to 590 inclusive, and if fewer than 510 heads are observed we are satisfied that the coin will come up heads less than 55% of the time. Therefor if between 491 and 509 heads inclusive are observed, we are satisfied that the coin will come up heads and tails with equal frequency to within a 5% equivalence factor and classify the coin as a FAIR coin.

If the number of observed heads is between 460 and 490 inclusive, or between 510 and 540 inclusive we have not classified it into any of the three categories. We judge its proper classification to be *unknown*, pending additional data.

3.10.4 Classifying a Function

Functions at a node can be classified in much the same way as coins can be classified. Coins were classified depending on whether they came up heads more than 50% (HEAD), less than 50% (TAIL), or equivalent to 50% (FAIR). The same can be done for functions at a node depending on whether they are correct more than the current function (SUPERIOR), less than the current function (INFERIOR), or the same as the current function (REDUNDANT). If there is insufficient information to make this determination, we classify it as UNKNOWN.

To better illustrate the previous discussion, consider the following example. In the ensuing discussion, I will refer to Table 3-2 and Figure 3-2. In this example, the present function that the node is computing is correct 70% of the time. Table 3-2 gives the classification criteria for differing number of trials. Figure 3-2 presents this information graphically, though not necessarily to scale. On the y-axis the fraction correct is plotted, on the x-axis the number of trials is plotted.

As we did before in the coin problem, we must determine what the 99% confidence interval is, which varies with the number of trials. The current fraction correct is shown by line Aa. The confidence interval about the current fraction correct is delineated by the lines Bb and Cc. If the fraction correct for the alternative function lies above the line Bb, we classify it as a SUPERIOR alternative function. If the fraction correct lies below the line Cc, we classify it as an INFERIOR alternative function. To classify the function as REDUNDANT we must show it falls within the equivalence factor of the fraction correct for the current function. This range is delineated by the lines Dd and Ee which are parallel to line Aa, the fraction correct for the current function. The 99% confidence interval about line Dd is delineated by lines Ff and Gg. The 99% confidence interval about line Ee is delineated by lines Hh and Ii. The classification of REDUNDANT can be made in the region bounded by lines Bb, Cc, Gg, and Hh. In the remainder of the figure, we make the classification of UNKNOWN since we do not have a sufficient number of samples to make a proper determination among the first three categories.

A few observations about the diagram are in order. In the leftmost part of the figure, the UNKNOWN region dominates. This reflects that only a few number of trials have been conducted and that insufficient information is available for a proper classification is available. In the rightmost part of the figure, the y-axis is divided exclusively into SUPERIOR, INFERIOR and REDUNDANT regions. This implies that with a sufficient number of trials, a proper classification can always be made. This is the case because the boundaries of the confidence interval asymptotically approach their center. So lines Bb and Cc asymptotically approach line Aa, lines Ff and Gg asymptotically approach line Dd, and lines Hh and Ii asymptotically approach line Ee. Therefor lines Bb and Cc will come closer to line Aa than any constant, namely the equivalence factor. This insures that the lower boundary to the SUPERIOR

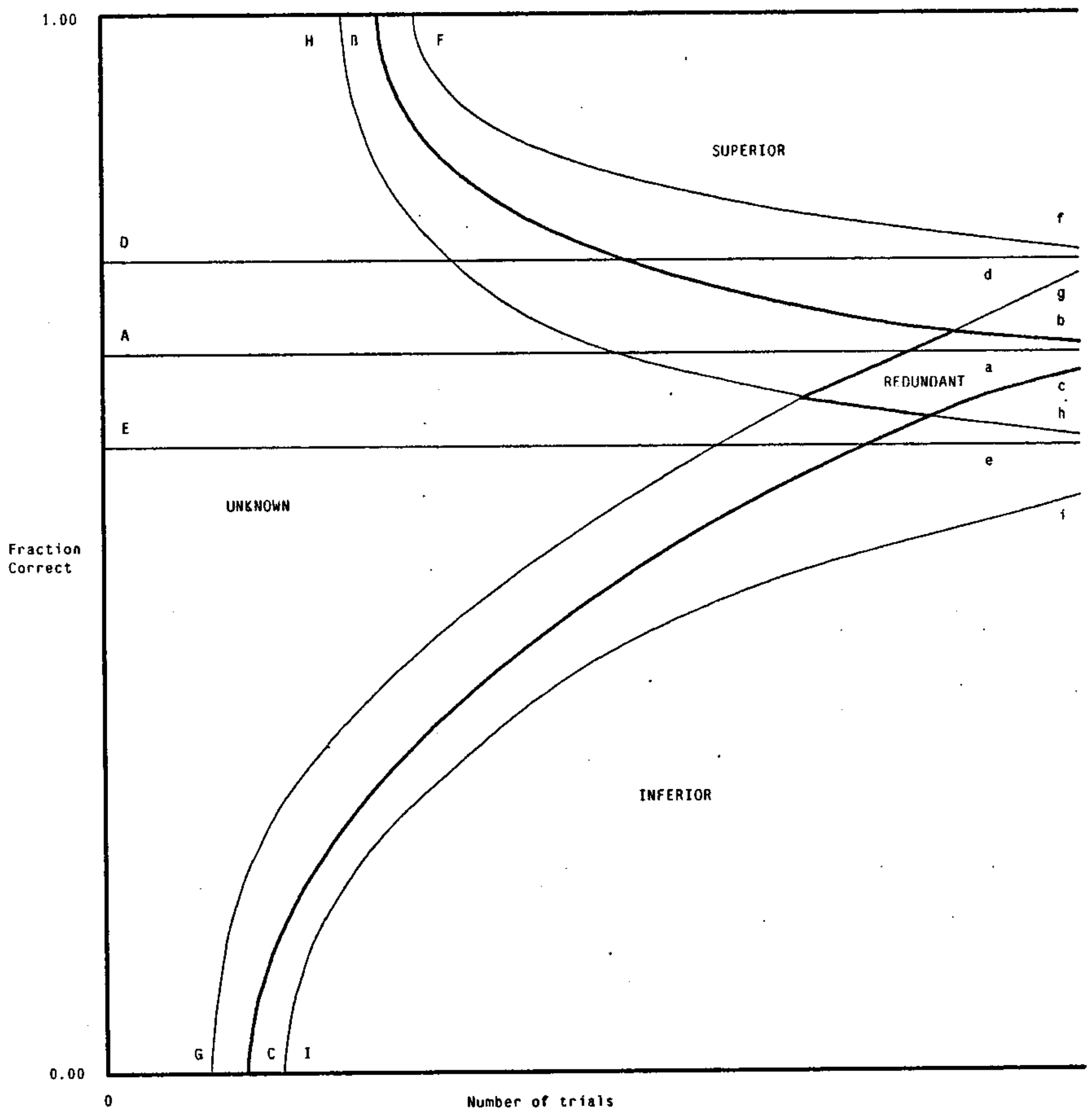

Figure 3-2: Classification Criteria Sketch

Size	10	20	50	100	200	500	1000	2000
Correct	7	14	35	70	140	350	700	1400
99% Lower	.327	.436	.533	.582	.617	.647	.663	.674
Upper	1.000	.964	.867	.818	.783	.753	.737	.726
5% Lower	1.000	.925	.824	.773	.737	.705	.689	.677
Upper	.397	.501	.592	.638	.671	.700	.715	.725
Reject <	4	9	27	59	124	324	663	1354
Redundant	-	-	-	-	-	-	689	1355
	-	-	-	-	-	-	715	1450
Accept >	10	19	43	81	156	376	737	1450

Table 3-2: Classification Criteria Example

region, line Bb, will intersect the upper boundary to the REDUNDANT region, line Gg. It also insures that the upper boundary to the INFERIOR region, line Cc, will intersect the lower boundary to the REDUNDANT region, line Hh.

The sketch in Figure 3-2 is asymmetrical. This is the general case, the figure will be symmetrical only when the current function, line Aa, has a fraction correct of 50%. In the network, we have used a chi-square test instead of explicitly calculating the binomial distribution to simplify the calculations of the confidence intervals.

An underlying assumption is that the each trial is independent. Were the trials to be dependent in some fashion, the assumptions about the probability distributions would be incorrect. Obtaining ten out of ten correct when all the trials are independent is far more significant than obtaining ten out of ten when the outcome of the trials are dependent. To employ this method requires assurance that the trails are independent and we will take measures discussed later to assure this.

By using this classification method, we can compare an alternate function to the current function and classify the alternate function as SUPERIOR, INFERIOR, or REDUNDANT, and if lacking sufficient information to place it in the first three categories, classify it as UNKNOWN. This gives us a means to determine which of two functions is better.

3.10.5 Correlating a Pair of Functions

We have need to determine when two functions are INDEPENDENT or DEPENDENT. This can be accomplished by using a method similar to that discussed above. We seek to determine when the pair of functions compute the same result to an unacceptable degree. The two functions are considered DEPENDENT if they compute the same functions or if one function is the inverse of the other. Two functions are considered to be equivalent if they obtain the same results within a *dependence factor* with 99% confidence. Two functions are considered to be inverses of one another if they obtain opposite results with a dependence factor with 99% confidence. If it cannot be determined that the two functions are DEPENDENT, we classify them as INDEPENDENT. By this classification of INDEPENDENT, we really mean that the two functions are not totally dependent.

To better illustrate this, consider the following example. In this example, a dependence factor of 10% will be used. Table 3-3 gives the classification criteria for differing number of trials. Figure 3-3 presents this information graphically, though not necessarily to scale. On the y-axis the fraction in agreement is plotted, on the x-axis the number of trials is plotted.

We must determine when we are 99% confident that the functions agree more often than the dependence factor allows. With a 10% dependence factor, we seek to show that the functions agree more than 90% of the time, line Jj, or less than 10% of the time, line Kk. The 99% confidence interval about the dependence factor must be determined, which varies with the number of trials. The limit of the confidence interval above the 90% function agreement is shown by line Ll. The limit of the confidence interval below the 10% function agreement is shown by line Mm. If the fraction in agreement for the specified number of trials is above line Ll or below line Mm, we classify the pair of functions as DEPENDENT, otherwise we classify them as INDEPENDENT.

It should be noted that the lines Ll and Mm approach lines Jj and Kk respectively asymptotically. This figure is symmetrical under all conditions.

3.11 Analyzing Functions at a Node

Each node computes one function of its inputs, while it has the potential to compute any of the 38 functions enumerated in Table 3-1 excluding the 2 constant functions for the nodes in non-top communities. The node maintains a history for the current function as well as for all of the potential alternative functions. During the Analysis Phase of each cycle, the node attempts to determine if any of the functions it is not currently computing could better satisfy its superiors. The node uses the classification method to compare each alternative function to the current function, and classifies each alternative among the set of SUPERIOR, INFERIOR, REDUNDANT, and UNKNOWN. The nodes use the following procedure to determine what function to compute in the next cycle :

1. If any of the alternate functions is classified as SUPERIOR then the alternate function with the greatest fraction correct is selected to be the new function the node will compute. In the case of a tie, the function requiring the fewest inputs is preferred with remaining ties broken arbitrarily. This new function is marked to be made into the current function in the Changing Function Phase. Links controlled by the node but unneeded by the new function are marked to be unlinked in the Unlinking Phase.

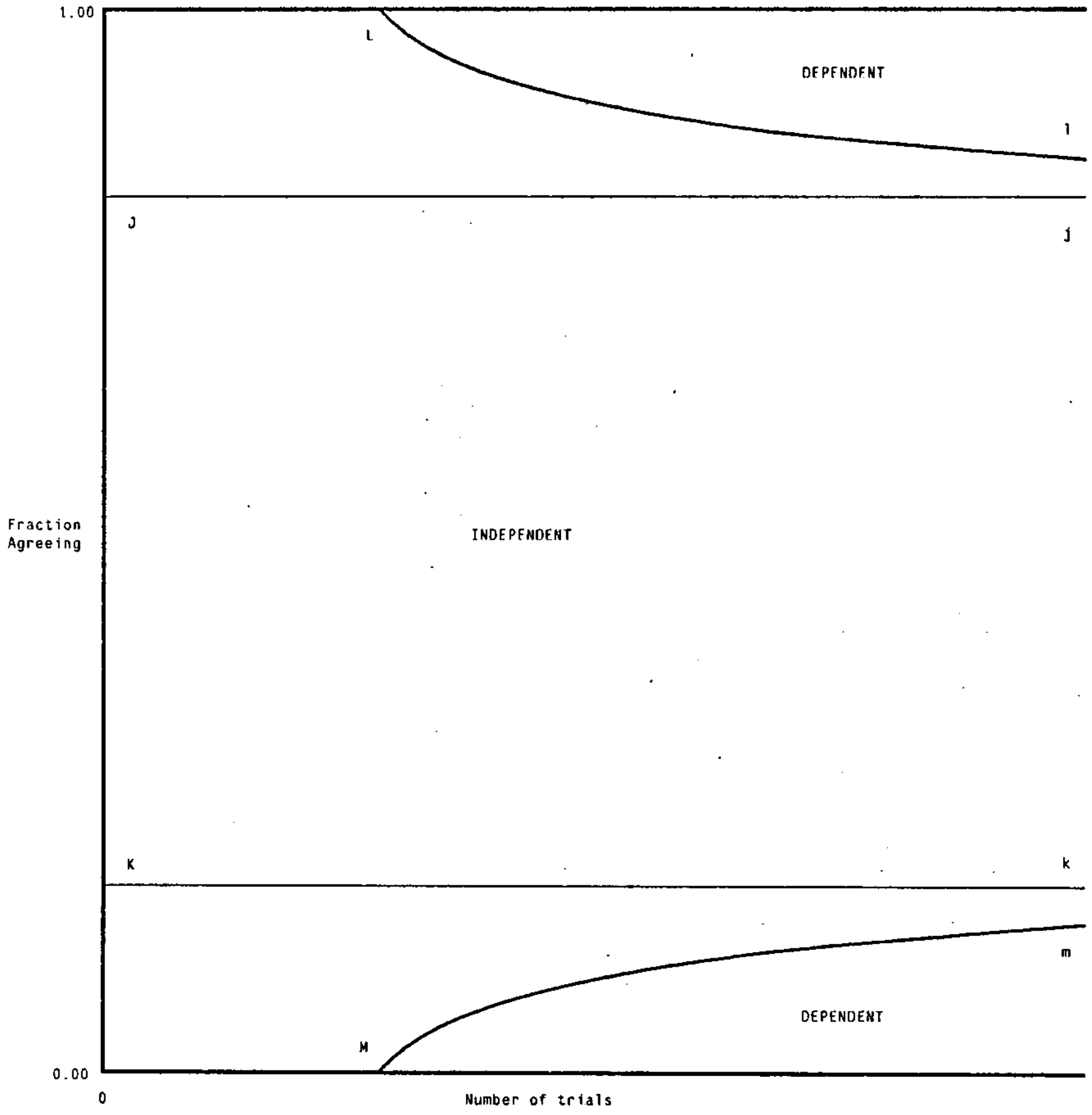

Figure 3-3: Dependence Criteria Sketch

Size	50	100	200	500	1000	2000	5000
Upper Dependence	45	90	180	450	900	1800	4500
99% Upper	1.000	.977	.955	.935	.924	.917	.911
Lower Dependence	5	10	20	50	100	200	500
99% Lower	.000	.023	.045	.065	.076	.083	.089
Dependent Upper	0	98	190	467	924	1834	4554
Dependent Lower	0	2	10	33	76	166	446

Table 3-3: Dependence Criteria Example

2. If none of the alternate functions is classified as SUPERIOR or UNKNOWN, then the alternate functions classified as REDUNDANT and the current function are compared to determine the function with the greatest fraction correct, with ties broken as described above. This function is marked as the new function, and the unneeded links are marked as discussed above.

3. If no alternate functions are classified as SUPERIOR, but at least one alternate function is classified as UNKNOWN, then the current function is retained. No function is marked for the Changing Function Phase, and no links are marked for the Unlinking Phase.

We desire to change to a SUPERIOR function as soon as we can determine which, if any, it is. When it is clear that no SUPERIOR functions exist because all the alternate functions are classified as INFERIOR or REDUNDANT, then we must settle for a function equivalent to the current one. When there are no SUPERIOR functions but still are UNKNOWN functions, we must wait to determine if any of the UNKNOWN functions may be classified as SUPERIOR.

When breaking ties between functions with equivalent fractions correct, we choose the function requiring the fewest inputs. By preferring the simplest function available, we make more controlled links available to be reconnected to other potentially useful nodes. This gives the node a preference for the simplest explanation for an observed phenomena (Occam's Razor).

When a new function is selected from one to three controlled links are marked for replacement in the Unlinking Phase. If the new function is to be a constant, then all three controlled links are unneeded and they are all marked for replacement. If a one input function is selected, then the necessary link is retained while the other two are marked for replacement. If a two input function is selected, then the two necessary links are retained while the third is designated for replacement. Once we have examined the possible functions and made our determination for the best function to compute, we need to examine as many new possible functions as possible. This is accomplished by replacing controlled links to nodes whose potential we have already examined.

3.12 Analyzing Links at a Node

Each node analyzes the signals received from its controlled links during the Analysis Phase. The controlled links may be transmitting signals that are of no value to the node; if so the node will seek to identify and replace the useless links.

A controlled link to an input node may be connected to a node which outputs a constant. Since a node can compute any pairwise boolean function, a link to a node computing a constant provides no useful information. As an example, assume the X link is connected to a node that always outputs TRUE. In Table 3-4, the 2 Input column shows the pairwise boolean function of X and Y and the 1 Input column shows the equivalent functions if X is always TRUE. Since the node computes all the functions in the 1 Input column with its Y link, there is no advantage to having the X link connected to a node that outputs TRUE. Analogous cases hold for the X link always signaling FALSE, and constants on the other links. Any links determined to be DEPENDENT with the functions TRUE or FALSE, are marked for replacement in the Unlinking Phase.

2 Input	1 Input	2 Input	1 Input
FALSE	FALSE	X and Y	Y
X nor Y	FALSE	X eqv Y	Y
~X and Y	FALSE	Y	Y
~X	FALSE	~X or Y	Y
X and ~Y	~Y	X	TRUE
~Y	~Y	X or ~Y	TRUE
X xor Y	~Y	X or Y	TRUE
X nand Y	~Y	TRUE	TRUE

Table 3-4: Simplifying Pairwise Functions with Constant Input

Two controlled links may be connected to nodes which output the same function or inverses of one another. In this case, only one of the links is useful since we can determine the second given the output of the first. Table 3-5 illustrates the case where the X and Y links output the same function. The 1 Input column shows the function of the Y link that is equivalent to the function of the X and Y links shown in the 2 Input column. Each Analysis Phase, all pairs of links are classified as either INDEPENDENT or DEPENDENT. If any pair of links is marked as DEPENDENT, then one link in the pair chosen arbitrarily is marked for replacement in the Unlinking Phase.

In either of the two cases above, the link being replaced may be part of the current function. For example, if it is determined that the X link is redundant and the function being computed is X OR Y, then the X link which are function depends on will be replaced. In cases like this, we mark the node to replace its two-input function by the one-input function equivalent.

2 Input	1 Input	2 Input	1 Input
FALSE	FALSE	X and Y	Y
X nor Y	~Y	X eqv Y	TRUE
~X and Y	FALSE	Y	Y
~X	~Y	~X or Y	TRUE
X and ~Y	FALSE	X	Y
~Y	~Y	X or ~Y	TRUE
X xor Y	TRUE	X or Y	Y
X nand Y	~Y	TRUE	TRUE

Table 3-5: Simplifying Pairwise Functions with Matching Inputs

3.13 Analyzing Output at a Node

A node that is not a member of a top community should not compute a constant. The tutor's classification of the input vectors may make a constant the best response from the network, therefor the nodes in the top communities may need to output a constant. From the previous section, we know that nodes cannot usefully employ a link to a node that outputs a constant. Unlinking from nodes which output a constant is inefficient since each node that links to the node that outputs a constant must independently determine to unlink from the node. It is more efficient to let the node detect when it outputs a constant. Each Analysis Phase, we compare the current function to a constant function and determine if they are DEPENDENT or INDEPENDENT. If they are determined to be DEPENDENT, all links are marked for replacement in the Unlinking Phase and a randomly selected function is marked to be the new function in the Function Changing Phase.

It should be emphasized that prohibiting the node from computing a constant function is not sufficient. Consider a node that outputs X OR Y, where X is ~A and Y is A OR B. The node will output TRUE since ~A OR A OR B will always evaluate to TRUE. Testing the X and Y links against a constant and each other will yield the INDEPENDENT classification. Though the inputs are not completely dependent, they may be partially dependent a case our classification scheme will not recognize. As a result, the test for a node that outputs a constant must be explicit.

3.14 Freeing Links

In the Unlinking Phase, the links designated to be unlinked are freed. Before being unlinked, the node identifier of the subordinate node is recorded so the node may avoid reconnecting to this same node. The connection of the link to the subordinate node is then severed. The link is now free to connect to another destination node in the Linking Phase.

3.15 Changing Functions

A node that is changing its function sends a New-Function signal to all of its superiors. If a node receives a New-Function signal from a link, all histories of functions that are dependent on that link are reinitialized. The node also determines if the current function is dependent on the signaling link, if so it sends a New-Function signal to its superiors. When a node changes its function, all of the nodes which have been tracking its usefulness must be informed of the change. It would be confusing to a node to merge statistics of a subordinate who was formerly computing X AND Y and is now computing X NAND Y, for example. The New-Function signal scheme performs this role.

3.16 Making Links

In the Linking Phase, any links without subordinate nodes are reconnected. The nodes in each community are allowed to connect their controlled links to any node from a set of other communities. This set of communities may vary for different communities. Among the set of communities which may be connected to, any node including spokesman are eligible. The only restriction is that a link may not connect to a node with which a link was unlinked from in the Unlinking Phase of the current cycle. This prevents linking to node that has just been shown to not be useful in the current cycle. This restriction is waived if the set of nodes that can be linked with would otherwise be null.

3.17 Calculating the Confidence Level of a Node

Every node maintains a measure of its confidence. Nodes with confidence have demonstrated that they are receiving reinforcements that are better than chance, meaning being more than 50% correct. Nodes without confidence have not demonstrated that the reinforcements they are receiving are better than chance. To determine if a node has confidence, we use the classification method discussed previously. We determine if the current function would be classified as SUPERIOR when compared against a function obtaining a fraction correct of 50%. If the current function would have been classified SUPERIOR then we classify it as being CONFIDENT, otherwise we classify it as being UNCONFIDENT. This distinction is important since CONFIDENT nodes may issue internal reinforcement signals while UNCONFIDENT nodes may not. CONFIDENT nodes may be ordered based on how confident they are, with nodes having the greater fraction correct being considered more confident than their fellows.

3.18 Recording History at a Node

In order to make informed decision, a node must have an understanding of its role in the network. To achieve this each node maintains a history of several of its functions. The node does not explicitly retain a memory of all past events, but rather a statistical summary of past events. Each node maintains all of the histories itemized below except for nodes in the top community which need not maintain an output history.

- Reinforcement history. Thirty-eight values, one per potential function the node could compute. Records the fraction of POSITIVE and NEGATIVE reinforcements received which were POSITIVE.

- Link history. Three values, one per controlled link, which records the fraction of cycles the link was in the ON state.

- Link pair history. Three values, one per pair of controlled links, which records the fraction of cycles both links were in the same state.

- Output history. Single value which records the fraction of cycles the node was in the ON state.

These histories have been weighted to give more recent events more importance. Unweighted histories inhibited effective learning because the nature of the classification method let functions with long histories dominate functions with short histories. Often a node found a good function and built up a long history to attest to its goodness. Later, when a better function was available the shortness of its history prevented its superiority from being recognized. As an example a node that received 90 POSITIVE reinforcements compared to 10 NEGATIVE reinforcements could not demonstrate a superiority over a function receiving 800 POSITIVE reinforcements and 200 NEGATIVE reinforcements. This leads to a close-mindedness along the lines of "it has always been good enough, why consider anything else?" The method of weighting histories alleviates this problem.

The histories are maintained as follows. Each history is saved as a value, x, between $-n$ and n where $2n$ can be interpreted as the effective memory length of the history. The history is initialized, usually to 0. Whenever a reinforcement is received, the value is scaled towards 0 by x/n. Additionally, if the reinforcement was POSITIVE, x is incremented by 1, and if it was NEGATIVE, x is decremented by 1. The impact of the current event is 1, and the effective impact of past events diminishes exponentially as they become more remote in the past. If a node receives POSITIVE and NEGATIVE reinforcements with equal frequency, the value will exponentially drift towards 0 indicating a balance between the two. If the reinforcements are always POSITIVE, the value will climb towards n, which can be reached but never exceeded, indicating pure POSITIVE reinforcement. The values from 0 to n indicate increasing degrees of net POSITIVE reinforcement. The value $((x + n)/2n)$ gives the fraction correct out of $2n$ events. When measuring states or agreement of states, we make the appropriate substitutions for the fraction of POSITIVE and NEGATIVE reinforcements. Histories are always initialized to 0, except for the Reinforcement history which is initialized to the value of the history of the current function.

The use of weighted histories gives a common length to all histories which facilitates comparing them. The mathematical comparisons are much simpler than with an exact history. The weighted history also solves the problem of good functions with long memories dominating better alternatives with shorter memories. Unfortunately, we must accept the converse, namely that it takes longer to refute poor functions. After initialization, the weighted history has an effective memory of $2n$ events. We choose 0 for all but the Reinforcement history for which we choose the value of the current function history. A reasonable initial value must be chosen since the during the first few cycles the assumed history shall dominate the early cycles activity. For example, if the history were initialized to 0, it would indicate a history of equal POSITIVE and NEGATIVE reinforcement, for a function that always receives NEGATIVE reinforcement it will take several cycles to overcome this initialization. The unweighted history does not have this drawback. Another advantage to the weighted history is its ability to quickly recognize when its reinforcement pattern has changed. Consider a node which has learned the optimal function demanded by its superior who reinforces it. If it has an unweighted history of 10,000 events and suddenly its superiors asked it to solve a different task, it would take thousands of events before the node realized that its superior were reinforcing it differently. With a weighted history a hundred events may suffice.

3.19 Receiving Internal Reinforcements

Nodes may receive reinforcements from any of their superiors. The node receiving reinforcement must determine which if any to utilize. To accomplish this the node measures the Confidence-Level of each node that is their superior. If no superior is CONFIDENT, then only NEUTRAL reinforcement signals can be received. If any superior is CONFIDENT, then only the reinforcement from the superior with the greatest confidence is utilized. In the event of more than one superior with equal confidence, the node whose reinforcement was utilized in the previous cycle will be utilized in this cycle, otherwise cycles are broken arbitrarily which helps to maintain continuity in the reinforcement signals between cycles. We prevent the node from utilizing more than one reinforcement signal per cycle to insure that the reinforcements received are independent. The independence assumption is vital to the classification method. If a node had several superiors who were computationally equivalent, then they would all produce the same reinforcement signals. The node would interpret these as independent reinforcement signals, thus giving them more significance they deserve. To prevent this possibility, only the superior with the greatest confidence is recognized.

3.20 Issuing Internal Reinforcements

Every node can receive and transmit reinforcement in the Internal Reinforcement Phase. These reinforcement signals are used to choose the current function as previously discussed. When a node is CONFIDENT is has the right to reinforce the nodes it has controlled links to. A node issues reinforcement through every controlled link.

The reinforcement scheme is outlined in Table 3-6 which illustrates reinforcement through the X link, the cases for the other links are analogous. If the node received a NEUTRAL reinforcement signal, the NEUTRAL reinforcement signal is issued. The node has received this signal because it has not played a significant role in the classification by the network, or the tutor issued an UNCERTAIN reinforcement signal. It should be remembered that the link is transmitting a boolean signal, and that the inverse signal could have been sent. Had the signal been inverted, the node may have output a different signal. If a different signal been sent and the node actually received a non-neutral reinforcement it can be shown that the new reinforcement would be the opposite of its current reinforcement. So the node can hypothesize accurately the reinforcement it would have received had the X link sent an inverted signal. If the node received a POSITIVE reinforcement and inverting the X link signal would also have resulted in a POSITIVE reinforcement, then the X link had no control over the reinforcement signal so we send through the X link a NEUTRAL reinforcement signal. The same would apply if both had been NEGATIVE reinforcement signals. If the node received a POSITIVE reinforcement and an inverted X link signal would have resulted in a NEGATIVE reinforcement, then we know the X link was crucial in the reinforcement received by the node. For the node to receive a POSITIVE reinforcement, the X link must continue to output the result it is now, so we issue a POSITIVE reinforcement through the X link. The opposite case occurs when the node receives a NEGATIVE reinforcement, and an inverted X link signal would have resulted in a POSITIVE reinforcement. We issue a NEGATIVE reinforcement through the X link to encourage the node connected to via the link to change the output it computes.

Reinforcement to X Link

Reinforcement Received By Node	Reinforcement Received By Node if X Link Signal Inverted	Reinforcement Issued to X Link	Reason
positive	positive	neutral	X Link - no effect
positive	negative	positive	X Link - correct
negative	positive	negative	X Link - wrong
negative	negative	neutral	X Link - no effect
neutral	--	neutral	desired signal unknown

Table 3-6: Reinforcement Scheme

Table 3-7 gives examples of reinforcement to the X link for two functions. In the first example, the node is computing the function X. The output of the node is identical to the state signaled by the X link. Observe that the reinforcement issued through the X link is identical to the reinforcement received by the node itself. The node is acting as an intermediary between its superior and its X link subordinate, and is merely passing information between these two nodes. In the second example, the OR function is being computed. When the Y link is signaling ON, the X link receives NEUTRAL reinforcement, since it cannot affect the output of the node. When the Y link is signaling OFF, the X link receives either a POSITIVE or NEGATIVE reinforcement since its output is crucial to the reinforcement received by the node.

Function	X	Y	Result	Reinforcement Node Received	Reinforcement Node Received if X Inverted	Reinforcement Issued through X Link
X	0	0	0	positive	negative	positive
	0	0	0	negative	positive	negative
	0	1	0	positive	negative	positive
	0	1	0	negative	positive	negative
	1	0	1	positive	negative	positive
	1	0	1	negative	positive	negative
	1	1	1	positive	negative	positive
	1	1	1	negative	positive	negative
X or Y	0	0	0	positive	negative	positive
	0	0	0	negative	positive	negative
	0	1	1	positive	positive	neutral
	0	1	1	negative	negative	neutral
	1	0	1	positive	negative	positive
	1	0	1	negative	positive	negative
	1	1	1	positive	positive	neutral
	1	1	1	negative	negative	neutral

Table 3-7: Reinforcement Examples

3.21 Initialization of the Network

The network can be initialized as follows. Mark every controlled link for connection, and every node to have its function changed to a randomly selected function from among those eligible functions. Execute the Changing Functions and Unlinking Phases successively, and the network is ready to begin.

3.22 Explanation of Cycle Timing

Conceptually each cycle is broken into several distinct phases. As implemented some of these phases are carried out simultaneously. The first three phases can be processed at the same time. Both reinforcement phases can be carried out simultaneously. No other overlap is possible.

The Unlinking, Changing, and Linking Phases are conducted separately to minimize disruption to the network during transitions. Nodes which are changing their function, signal a New-Function to their superiors. The New-Function signal causes its recipients to lose confidence and reinitialize their statistics. This loss of information should be avoided when it is unnecessary. If node P has designated the link to node Q to be replaced, and node Q is about to change its function, then there is no need for node P to receive the New-Function signal from node Q. Placing the Unlinking Phase before the Changing Function Phase allows their connecting link to be freed preventing the reception of the signal. Similarly if node R wishes to make a link to node Q, then there is no need for node R to receive the signal either. Therefor, the Linking Phase is placed after the Changing Function Phase so that the New-Function signals are transmitted before the new link connection is made. In both cases, nodes P and R had no reason to know that node Q was changing its function.

4. Results

4.1 Overview

To demonstrate the capabilities of the CONSENSUS system it was run against three instances of the learning task. The first instance requires classifying all possible functions of three inputs, which vary significantly in difficulty. The network was developed on this problem and later applied to the following problems. The second instance is the shifter problem, here the network must recognize an important regularity, but has more inputs to contend with. In the third instance, the network controls a simulated organism in a more complex environment.

4.2 A Simple Test

For a simple non-trivial test, the following task was posed to the network: Learn to recognize and correctly classify each of the possible unique boolean functions of 3 variables. The binary input vectors consists of 3 elements, allowing 8 (2^3) unique input vectors. We required the network to classify each vector as either VALID or INVALID, which allows 256 (2^8) possible functions. From among these 256 possible functions only 14 are unique, the remainder being equivalent if you can rename inputs and recognize inverses. Figure 4-1 enumerates the functions, gives the frequency with which they or an equivalent occurs, and diagrams a Karnough map of the function.

To solve this problem, the network was configured to have three communities organized into three layers with each community composed of 24 nodes. The nodes in each community are permitted to link to any node below them including the input nodes. The network was run against each unique function for 1024 cycles. The number of minterms correct (out of a maximum of 8) for the network is tabulated in Table 4-1.

The learning process was observed to occur in roughly three phases. The first phase occurs during the initial 20 to 30 cycles. The network begins with a random initialization of functions and connections of dubious value. During the first few cycles the top community nodes identify the aspects of the random initialization that were especially poor. These functions that perform badly are quickly replaced by better functions that generally compute slightly better than chance. In this problem, top community nodes would be expected to be initialized such that they would compute 4 of the 8 minterms correctly. Typically, after this first phase there are essentially no top community nodes getter fewer than 4 minterms correct and the majority are getting between 5 and 7 minterms correct. The spokesman in the top community usually have at least 6 minterms correct and often have all 8

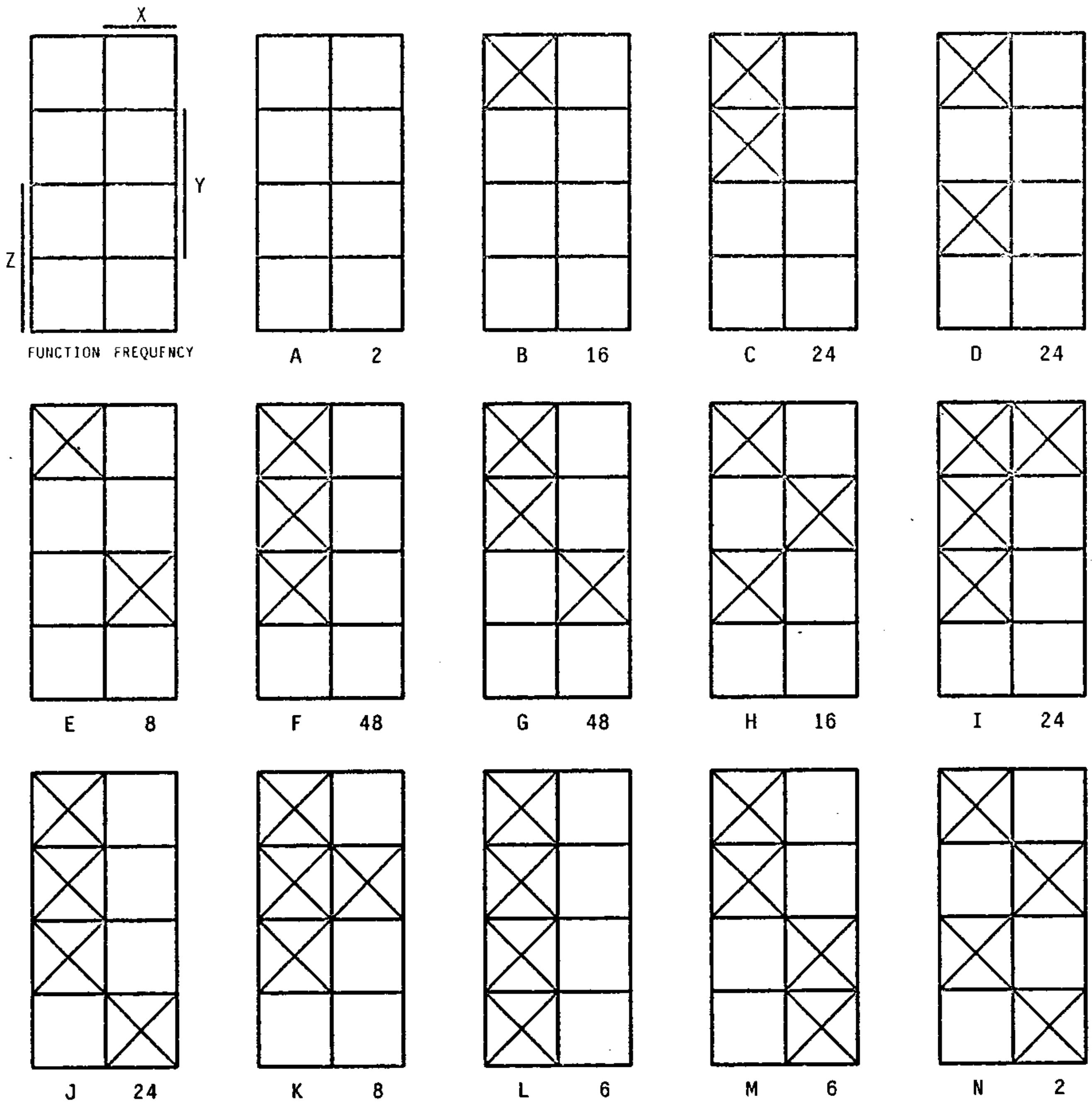

Figure 4-1: Unique Boolean Functions of 3 Inputs

minterms correct. Activity in this first phase is confined principally to the nodes in the top community because until the nodes in the top community gain confidence, no reinforcement is issued to the nodes to the lower levels. While performance after this phase is often surprisingly good, the knowledge is not robust. There are often many critical nodes in the network. These nodes are critical because a change in their function would drastically affect the performance of the network as a whole.

Slow progress is characteristic of the second phase. Most of the nodes in the top community now possess confidence and are reinforcing their subordinates. Several nodes in the lower levels are able to identify features that are useful to their superiors. The positive reinforcement from their superiors allows these nodes to "lock on" to the feature they have identified. Once a node has "locked on" to a feature, other nodes at higher levels are often able to determine how to use this feature to improve their performance. This phase ranges from 50 to 300 cycles in duration. At its conclusion, most top community nodes have either 7 or 8 minterms correct, though on difficult problem nodes with 6 minterms correct can be found. The spokesman has all of the minterms correct and many nodes in the lower level now possess confidence. The knowledge is relatively robust at this point.

The third phase consists of very slow learning. Nodes in the top community make slow progress while searching for a way to get their last minterm correct. When a node has all but one minterm correct, it must often explore a large number of possibilities before it finds a way to get all the minterms correct. Nodes in the lower level occasionly find improvements in their functions. The New-Function signal they generate during their transition to a new function can cause many nodes above them to lose confidence and reinitialize their histories. After losing confidence, these nodes are vulnerable to the temptation of changing their current function for one that is almost as good but is temporarily performing as well as or better than the current function. Nodes that yield to the temptation must then relearn the minterms they now have incorrect. The network generally makes very slow progress and eventually reaches an apparent equilibrium between the slow improvements of the top community nodes and the consequences of the waves of New-Function signals.

On these problems, the network obtains 8 out of 8 minterms correct in 512 cycles and maintains this through 1024 cycles. The network had the easiest times with the functions of two inputs or less (A, C, L, and M). These are the functions possessing the greatest regularity. The network had the greatest difficulty with functions G and H. These functions have several minterms which are dispersed from each other. Functions are difficult to learn if they have few underlying regularities. Adjacent minterms with the same value form regularities. Functions such as B or C with a preponderance of one minterm value have many adjacent minterms of the same value. Functions with the equal number of minterms of each value cannot help but possess some degree of regularity. The nodes with a slight preponderance of one minterm value offer the fewest underlying regularities.

	Function	Cycles 32	64	128	256	512
A	False	8	*	*	*	*
B	X ^ Y ^ Z	7	7	7	8	8
C	X ^ Y	8	8	8	8	8
D	X ^ (Y xor Z)	7	7	7	8	8
E	(X^Y^Z) v ~(X^Y^Z)	6	6	7	8	8
F	X ^ (Y v Z)	8	8	8	8	8
G	(X ^ Y) v (~X^~Y^Z)	7	7	7	7	8
H	(X ^ (Y xor Z)) v (~X^Y^Z)	8	8	7	7	8
I	(X ^ Y) v (~Y ^ Z)	8	8	8	8	8
J	X xor (Y ^ Z)	8	8	8	8	8
K	(X ^ Y) v (Z ^ (X xor Y))	7	7	7	8	8
L	X	8	8	8	8	8
M	X xor Y	7	7	8	8	8
N	X xor Y xor Z	6	7	8	8	8

* - every node in community has 8

Table 4-1: Results on Boolean Functions of 3 Variables

4.3 The Shifter Problem

As a more difficult test, the network was asked to solve the following problem : Learn to recognize shifts of the input vector. The binary input vector was divided into two components, S and V. The component S consists of a single element and designates the transformation on component V to be performed to produce O, the output vector. The component V consists of four elements numbered V_1 through V_4 and the vector O consists of four elements numbered O_1 through O_4. If S is ON, O should receive the corresponding elements of V, for example O_2 receives V_2. If S is OFF, O should receive the elements of V shifted one place, for example O_2 receives V_1. When a shift took place, O_1 should receive V_4 to produce a rotation of the last element. There are 32 (2^5) unique input vectors which were presented with equal probability. We required the network to produce O, the output vector. For each element of O, the network must determine which of 2^{32} functions implement the required function.

To solve the problem, the network was configured to have eight communities organized into two layers. Since a top community can produce only a single bit of output, one top community is needed for each element in O, the output vector. Essentially there are four networks operating in parallel.

Each network consists of one community in the first layer which connects only to input nodes, and one community in the second layer which is a top community and may connect only to nodes in its counterpart in the first layer. Each community was composed of 10 nodes, the maximum allowed by the current simulator given the earlier requirements. It should be noted that the network was given no advanced knowledge about the S and V components and must discover this on its own. The network was run for 1024 cycles and the number of minterms correct (out of a maximum of 32) is tabulated in Table 4-2.

In this problem, the first phase dominates the first 150 cycles. At the completion of the first phase the spokesman are ~90% correct utilizing nodes which ~75% correct. The second phase consists of roughly 450 cycles, as the nodes improve to ~95% accuracy and the spokesman become 100% accurate. During the third phase, the nodes produce slow progress in improving their accuracy.

Cycles	Nodes	Spokesman
32	18.65	22.75
64	21.65	22.75
96	22.78	22.75
128	24.30	27.75
160	24.75	28.75
192	25.20	28.50
224	25.20	27.00
256	25.65	27.75
288	26.85	30.00
320	27.05	30.00
352	27.10	30.00
384	27.20	30.00
416	27.60	30.00
448	28.15	31.00
480	28.28	31.00
512	29.12	32.00
544	29.00	32.00
576	29.50	"
608	29.70	"
640	29.85	"
672	29.85	"
704	29.85	"
736	29.85	"
768	29.85	"
800	30.05	"
832	30.05	"
864	30.05	"
896	29.85	"
928	30.05	"
960	30.20	"
992	30.40	"
1024	30.40	"

Table 4-2: Performance on Shifter Problem

4.4 The Tadpole Problem

As a final task, the network was asked to direct a simulated tadpole in a simple environment. The tadpole lives in a one-dimensional pond, it must come to surface for oxygen and dive to the bottom for food without waiting too long lest it die from starvation or asphyxiation. The pond has eight distinct depths, oxygen may only be obtained at the surface, depth 0, and food may only be obtained at the bottom, depth 7. The tadpole's lungs may hold up to 15 units of oxygen, and its stomach may hold up to 15 units of food. Each cycle, it consumes 1 units of oxygen and 1 unit of food. It dies when it has no oxygen or food. When at the surface, it fully replenishes its oxygen. When at the bottom, it fully replenishes his food. Each cycle it can either swim upwards or downwards. Swimming upwards decreases its depth by one except when at the surface, in which case it remains there. Swimming downwards increases its depth by one except when at the bottom, in which case it remains there. The tadpole is aware of how many units of oxygen and food it has. The tadpole was started at the surface with a full complement of oxygen and food, in the event it died the tadpole was reincarnated in this state.

To allow the network to control the tadpole, we must establish a correspondence between the tadpole environment and the network environment. To present the network with a binary input vector the following was done. The tadpole's food supply can be represented by a four bit number, as can the oxygen supply. We used the 3 high order bits from the food supply and the 2 high order bits from the oxygen supply to create a 5 bit binary input vector to the network. The VALID classification was taken to mean swim downwards, and the INVALID classification was taken to mean swim upwards. The tutor insists the tadpole swim downwards when it has less food than oxygen, and swim upwards when it has less oxygen than food. When it has an equal amount of food and oxygen, the tutor is indifferent and gives an UNCERTAIN classification. To confuse the network, the tutor may give an incorrect classification on any cycle. The tutor will classify the vector correctly 80% of the time, however each of the two alternatives will be designated 10% of the time. These classifications are used regardless to the current depth of the tadpole. The state of the network was not specially altered in the event the tadpole perished.

The nature of the problem demonstrates several capabilities of the network. With a binary input vector of 5 elements, up to 32 (2^5) unique input vectors may be presented. The network may need to consider up to 2^{32} different functions. Unlike the previous problems, the input vectors will not be presented with equal frequency. The frequency with which each input vector is presented may vary significantly during the course of the task. The tadpole has incomplete information receiving only 5 inputs when 8 are required to completely designate the state of his oxygen and food supplies. To the tadpole, the tutor is inconsistent. With fewer inputs the network has a resolution that is too coarse to completely distinguish all possible states of the tadpole. The tutor lies 20% of the time, so caution must be exercised in interpreting any specific reinforcement.

The network has configured to have three communities organized into three layers with each community composed of 24 nodes. The nodes in each community are permitted to link to any node below them including the input nodes. The network was run for 512 cycles. Figure 4-2 shows the tadpole under control of the network. The left-hand charts show the depth of the tadpole as a function of time. The right-hand charts show the networks knowledge at the end of each life or the end of the simulation. For each possible combination of inputs, the action the network would choose is displayed, a cross indicating swimming downwards otherwise swimming upwards.

The first life of the tadpole lasts 23 cycles. The tadpole meanders near the top, then swims to the bottom, lingers there, and then perishes while swimming to the surface. The network has no real understanding at this early stage. The apparently intelligent swimming sequences are the result of the random initialization of the network.

The second life of the tadpole lasts 17 cycles. The tadpole lingers near the top, then swims to the bottom and tries to quickly return to the surface. The network now has a general understanding that when food is less than oxygen swimming downwards is required. It understands the converse about the need to swim upwards when oxygen is the more critical commodity. It is overly concerned about lack of food. This is due to its starting at the surface, the tadpole typically finds oxygen early while lacking food which should lead it to be more concerned about food.

In its third life, the tadpole has learned the ropes. It has learned to methodically swim upwards and downwards replenishing its oxygen and food. Early in its life it pauses for one cycle at the bottom, later it avoids this pause. The tadpole continued this cyclical swimming for the last 472 cycles of the simulation. It recognizes that it lives in one of two states, needing oxygen or needing food. It recognizes its current state and swims up or down appropriately. Once the tadpole has commenced the swimming cycle, the network only receives 11 unique input vectors. The remaining input vectors are never encountered by the tadpole anymore. The tadpole has a perfect encoding for these input vectors since they always allow it to choose the correct action.

4.5 Review

The network has shown its capability to handle several different problems. The network has faced environments with inputs of varying frequencies, inconsistent tutors, and with and without important underlying regularities. In each case, the network has been able to find a suitable encoding to solve the task presented to it. These tasks show some variety in their interconnection schemes. We intend to explore the system's ability to produce its own interconnection scheme in future research.

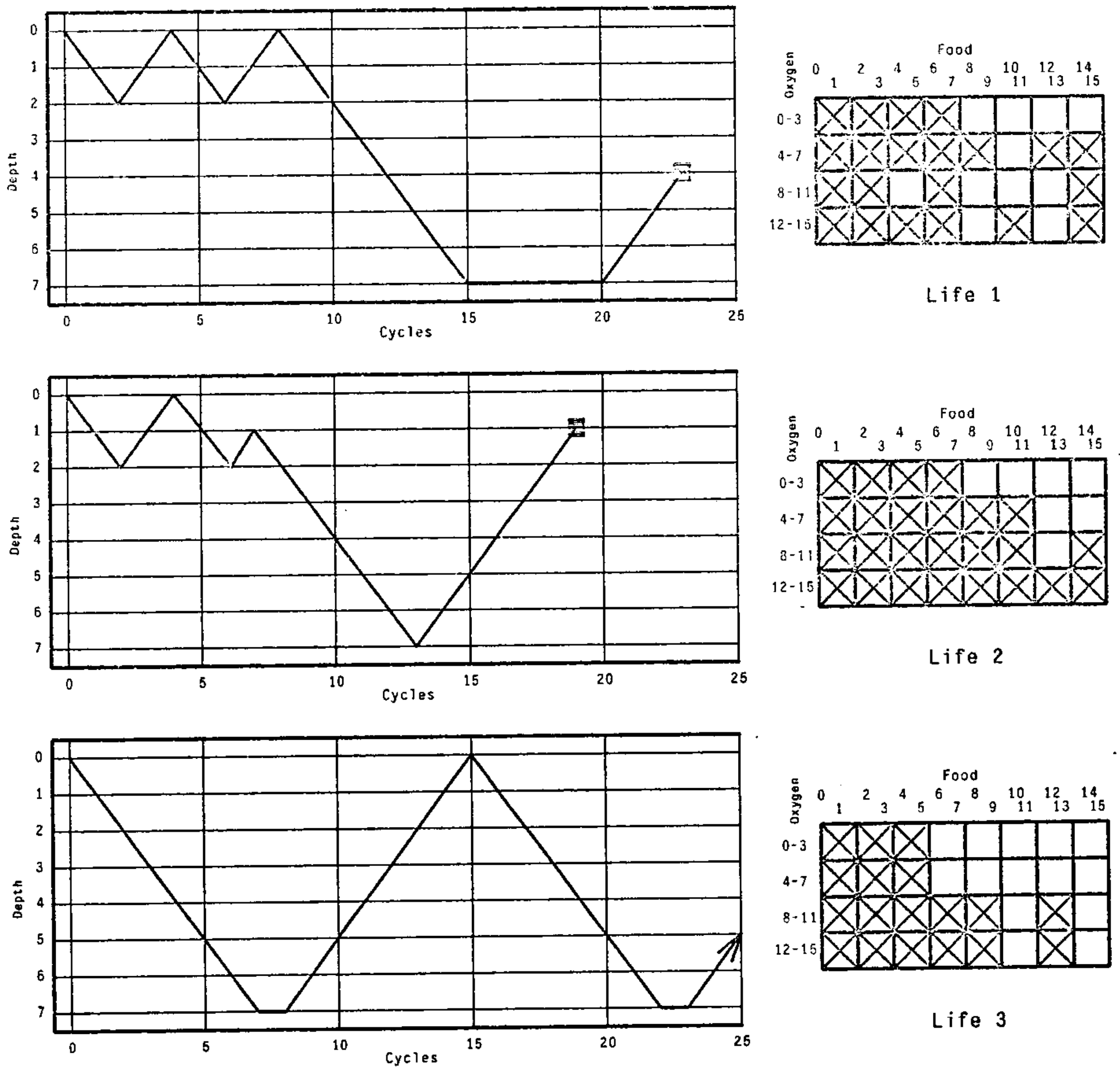

Figure 4-2: Tadpole Performance

5. Discussion

5.1 Introduction

In previous sections the structure of the network has been detailed together with its performance on several problems. In this section several aspects of the network shall be discussed including the notion of confidence and its role in the reinforcement scheme and the function of communities. The issue of convergence will be addresses along with some limitations on the network.

5.2 The Notion of Confidence

The notion of confidence is important to making the reinforcement scheme effective. This scheme deliberately attempts to suppress "noisy" reinforcement signals. Noisy reinforcement signals are of dubious value to a system based on statistical inference. These signals retard the rate of learning by generating additional trials that contain little information. Consider the coin-flipping experiment, before you need only study a single coin, if you must study several indistinguishable coins when you already know that all but one is fair, the randomness of the fair coins makes it more difficult to study the hypothetically "fair" coin. A node that is CONFIDENT has established that its performance is better than chance. A CONFIDENT node is aware that it serves an important function in the network. Nodes which are UNCONFIDENT would be expected to have a large noisy component in any reinforcement signals they might send since they have not established that they are performing better than chance. Since only nodes that are CONFIDENT may issue reinforcements other than NEUTRAL, the largest potential source of noisy reinforcements has been suppressed. The distinction between nodes with differing degrees of confidence, allows subordinates to respond to the superior with the greatest confidence and hence the least noisy reinforcements. This approach contrasts with that taken by most other researchers. They accept the presence of noisy reinforcements since their models rely on making numerous small changes. They expect that the errors will cancel each other out, leaving a meaningful change. We attempt to make a smaller number of more substantial modifications, but must not allow the presence of noisy reinforcements to slow down our determining which substantial modifications are best.

5.3 The Role of Spokesman and Communities

Each cluster has a distinguished node referred to as the *spokesman*. The spokesman samples the states of the other nodes in the cluster and adopts the most commonly found state. Essentially, he adopts the state of the majority of his brethren. His state represents the combined knowledge of the nodes in the cluster.

The basic notion is that collectively the nodes have more knowledge than any node has individually. Function H from the 3-variable tests provides a good example. A node in the first layer can do no better than getting 5 of the 8 minterms correct. Figure 5-1 shows the 19 functions that a first layer node could compute which have 5 minterms correct. The inverses of these functions have 3 minterms correct, and exhaust the alternatives available to the node. Clearly function H is beyond any individual first level node. It is also beyond any second level node with access only to first level nodes since the best any second level node can do is 7 minterms correct. A third level node is capable of calculating of function H.

However, function H is not beyond the capability of a community of nodes. If the first level community consisted of 19 nodes, each computing 1 of the 19 functions in Figure 5-1, the spokesman would have all the minterms correct. While the underlying regularities for this function are minimal each of the nodes with 5 minterms correct has captured some underlying regularity. While each node knows very little, each has some understanding of the problem. Collectively the nodes have sufficient understanding to solve the problem. Majority voting recognizes the 3 true minterms with a one vote margin, and rejects the minterm (~X) AND Y AND (~Z) by one vote. The remaining minterms are easily recognized to be false. While this example is unusual, it indicates the potential power of a collection of nodes each possessing minimal knowledge.

The community benefits from the independent pieces of knowledge each of the nodes possess. If each of the nodes were to compute the same function then the majority voting would yield no advantage. However, when each node has independently acquired a moderately good record, the group will have acquired an excellent record. As Table 5-1 demonstrates, a collection of 15 nodes each independently correct 80% of the time will have a majority correct over 99% of the time. Nodes with perfect knowledge (100% correct) or no knowledge (50% correct) derive no benefit from group action. Nodes with a moderate degree of knowledge (70% or more) do derive great benefit from group action. Groups of as few as 5 nodes can produce major improvements with groups as large as 15 producing even more. In practice, obtaining perfect knowledge in a single node is a difficult task, but obtaining a moderate degree of knowledge is not. By collecting a number of independent nodes with some knowledge, the community has good knowledge.

A crucial notion in the voting scheme is the independence of the nodes. We provide no explicit mechanism for ensuring independence of the nodes. Instead, we rely on the randomness of the connections and functions in the network. We select community sizes on the order of 20 or more nodes with the expectation that the nodes will have several independent methods of getting a large fraction correct. With several independent methods available, it can be expected that each of the

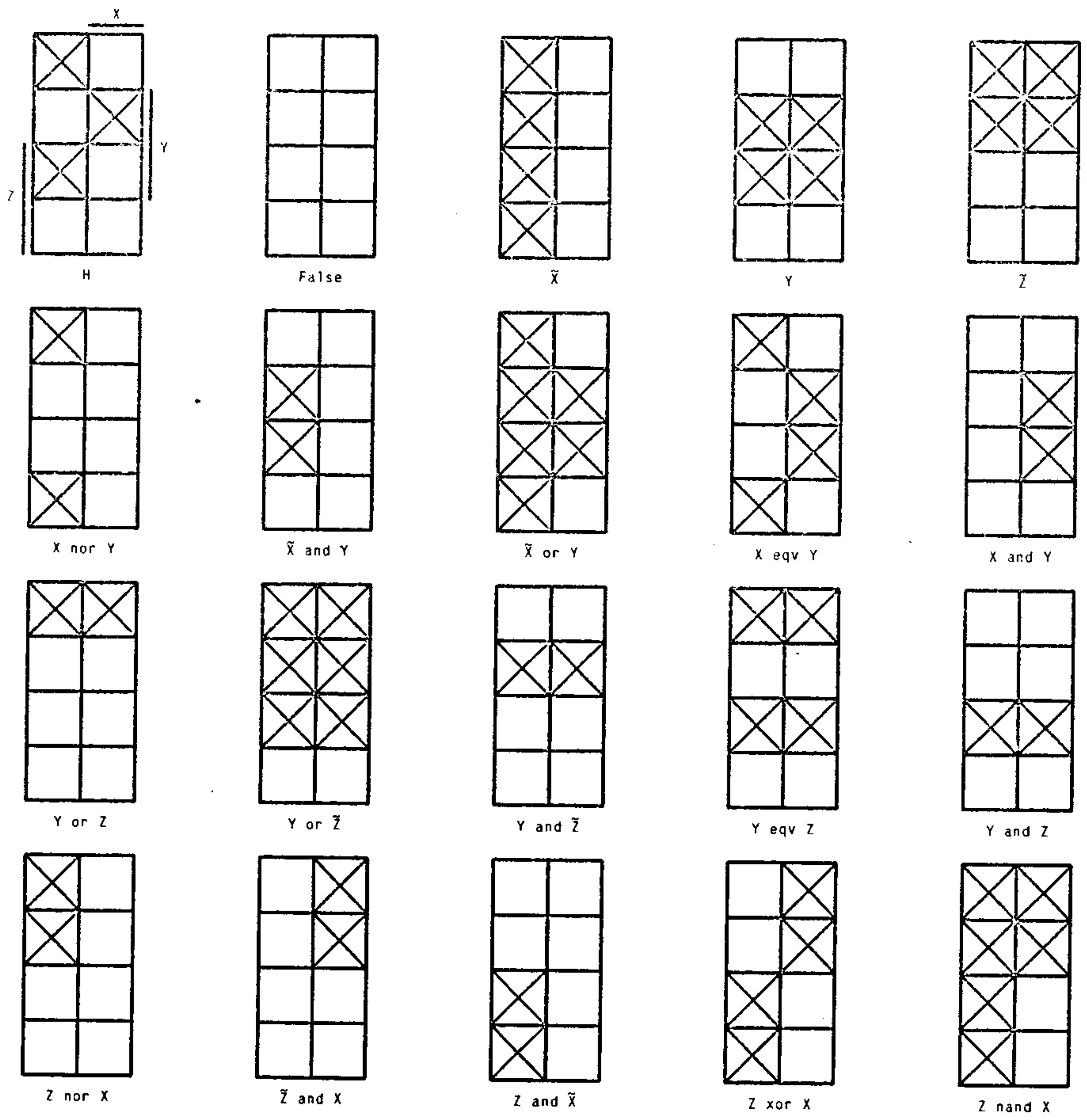

Figure 5-1: Community Example

Probability Right	Probability Majority Wrong For Size 1	3	5	7	9	15
.500	.5000	.5000	.5000	.5000	.5000	.5000
.600	.4000	.3520	.3174	.2898	.2666	.2131
.700	.3000	.2160	.1631	.1260	.0988	.0500
.800	.2000	.1040	.0579	.0333	.0196	.0042
.900	.1000	.0280	.0086	.0027	.0009	.0000
1.000	.0000	.0000	.0000	.0000	.0000	.0000

Table 5-1: Effects of Independent Majority Voting

methods will be utilized by approximately an equal number of nodes. While there is no guarantee of this, it has been observed to work well in practice. Communities are not large with the expectation that each node will find an equally good method, but rather so that each of the available methods is represented with rough equality. The spokesman nodes perform significantly better than conventional nodes when the conventional nodes encompass several independent methods.

5.4 The Credit-Assignment Problem

The downfall of many learning methods has been the "credit-assignment" problem. A learning method must be able to determine how to modify its parameters in order to improve its understanding of the designated task. In their paper [12], Hinton, Sejnowski, and Ackley state :

> The major technical stumbling block which prevented the generalization of simple learning algorithms to more complex networks was this : To be capable of interesting computations, a network must contain non-linear elements that are not directly constrained by the input, and when such a network does the wrong thing it appears impossible to decide which of the many connections strengths is at fault.

The CONSENSUS system attempts to do this by giving each node an understanding of the role of itself and its neighbors in the network. Each node in the network is a non-linear element that attempts to correlate two lower-order concepts into a higher-order concept for use by its superiors. The reinforcement scheme, including the notion of confidence, allows it to translate global reinforcement signals into meaningful local reinforcement signals to determine which part of the network must be improved.

The translation of global reinforcements to local reinforcements is carried out incrementally at each node. The top layers receive their reinforcement from the environment. While the top layer nodes are without confidence, they withhold reinforcement from the lower layers, allowing their immediate subordinates to experiment freely. When the nodes in the top community have confidence, they reinforce their subordinates. Each node refines the global reinforcement into a unique local reinforcement for each node it reinforces. The reinforcement scheme attempts to identify the nodes responsible for the global reinforcement and reward/punish that node accordingly. Nodes which cannot affect the global reinforcement receive neutral reinforcement. The subordinates of the top layer nodes reinforce their subordinates in kind.

The CONSENSUS system changes the functions at nodes and links in the network in order to develop an internal model that captures the structure of its environment. The decision method and reinforcement scheme give it the means to attempt to determine where faults lie and to correct them.

5.5 Interconnection Scheme

The organization of the nodes into communities imposes structure on the network. The simplest organization is to randomly interconnect nodes. This has the benefit of being simple, but is inefficient since the network must start from scratch for all knowledge. The grouping of nodes into communities allows the collective knowledge of the community members to be utilized. The spokesman serves this role and purpose by signaling the consensus of his compatriots. In addition to the introspective benefits of communities, the community provides a basis for external organization. Since the nodes of each community may be connected to the nodes of a specified subset of all communities we have a limited capability to "program" the network by specifying the interconnection scheme. An obvious application is to specify a scheme whereby nodes examine similar inputs together, in the case of a two-dimensional image, we can have nodes look only at pixels that are near one another. In the case of inputs of differing natures, we could organize the connections by input type, such as having some communities examine temperature, some pressure, and others force. These interconnection schemes can be specified in the lower layers to facilitate the learning on special problems before the higher-order concepts-reach the upper layers. This scheme allows us to specify the interconnection in the large, while retaining randomness in the small.

5.6 The Tradeoff Between Speed of Learning and Stability

The parameters affecting the rate of learning are the confidence-level, the equivalence factor, and dependence factor. To speed up the learning, in the short term, the confidence-level could be reduced, the equivalence factor increased, and the dependence factor increased. The drawback to speeding up the rate of learning is that the network becomes unstable. It is unstable because it is vulnerable to the chance presentation of a few unusual input vectors.

The rate of learning would be increased if the confidence-level were reduced from 99%, since each node would require fewer samples to make a decision. Increasing the equivalence factor would allow a greater number of functions to be classified as equivalent, and it would take less time for each function to be so classified. Increasing the dependence factor would make it easier to draw the conclusion that a pair of links was redundant. Increasing the equivalence and dependence factors would decrease the sensitivity of the nodes to fine gradations of performance, in more complex problems this loss would be of significance.

Since the network is constantly responding to the input vectors in proportion to their frequency in the immediate past, the presentation of a very small subset of the possible input vectors on many consecutive cycles would lead the network to believe that the frequency of presentation of the input vectors had changed. Reducing the tolerance of these parameters makes the network respond more quickly to these changes, since it requires fewer samples to make a decision. When the tutor presents such a subset of the input vectors by chance, the network may make an unwarranted reconfiguration thus losing its accumulated knowledge. Keeping the tolerances of the parameters high protects the network from short term sequences of examples by chance. The underlying philosophy of the network is to make a few well informed decisions rather than many less informed decisions, reducing the tolerance of the parameters would allow the network to make decisions on less information, contradicting our philosophy.

Learning schemes based on relaxation techniques face an analogous problem. The rate of learning can be increased by performing the relaxation faster, in the specific case of the Boltzmann machine by lower the temperature faster. The danger they face is that the faster the relaxation is performed the less likely the system will zero in on the global maxima. For CONSENSUS, the problem is not finding the global maxima, but staying there in the face of a chance presentation of an unrepresentable set of input vectors.

5.7 Tolerating the Environment and Tutor

The network does not require that the example input vectors span the space of all possible input vectors. The network responds to the input vectors in proportion to the frequency of their appearance; the more often a vector appears, the more the network will respond to that input. For input vectors that do not occur, the network may choose any classification. The network will choose the classification that is most consistent with the regularities inherent in the input vectors that have been presented to it. The frequency with which an input vector is presented may vary; the network will respond to that input vector in proportion to its frequency in the immediate past.

The network can tolerate non-deterministic and varying classifications by the tutor. The classification of an input vector by the tutor as VALID, INVALID, or UNCERTAIN need not be deterministic or consistent. For example, the tutor may classify a given input vector as VALID 90% of the time and INVALID 10% of the time. The network will classify the input vector as VALID to maximize its positive reinforcement. The network would also choose a VALID classification if the input vector were UNCERTAIN 90% of the time and VALID for 10% of the time. The ability to handle a non-deterministic tutor relieves the tutor of any requirement for internal consistency. The tutor's classification scheme may change with time; the network will respond to the classification used in the immediate past.

In an effort to maximize its positive reinforcement from the tutor, the network will concentrate on the input vectors that appear most frequently with the VALID or INVALID classification. Input vectors that are classified as UNCERTAIN or do not appear will be classified in the manner most convenient to the network in its pursuit of the more important input vectors. This is analogous to an electrical engineer freely utilizing do-not-cares in a Karnough map to produce the simplest circuit possible that fits his design requirements. In a pinch, the network may even intentionally choose the wrong classification for infrequent input vectors if it allows it to better classify the more frequent input vectors. When the tutor changes its classification of, or frequency of, some input vectors, the network will respond. After a short period of exposure to the new reinforcement criteria, the network will reconfigure itself in response.

5.8 Convergence

Until now, we have avoiding the issue of convergence. To be capable of achieving convergence, the network must be able to escape local maxima and settle into the global maxima. For complex systems this can be a very difficult task since it is infeasible to exhaustively search the entire space. The CONSENSUS network has the capability to escape many, but not all, local maxima and settle into a global maxima.

CONSENSUS constantly contemplates major changes in its functions, hence it is not limited to small local changes. A system confined to making small local changes has a limited horizon in searching for superior alternatives. A search conducted from the top of a small hill with a limited horizon will never "see" a mountain range in the distance. With each node at every level in the hierarchy constantly probing for alternatives with one controlled link, the network is not confined to local changes. In general, while the lower level nodes are contemplating minor changes, the upper level nodes will be contemplating major changes. This gives the network the opportunity to consider more than local changes. When confronted with two equivalent alternatives, the network chooses randomly between them. This gives it the capability to fully explore plateaus in the search space. While there may be no better alternatives available from the current position, there may be better alternatives available from a position equivalent to the current position.

The continuous probing for alternative connections gives the nodes the opportunity to explore all potential alternative connections subject to the restriction of maintaining the links for the current function. Unless the current function requires two links neither of which is used in the optimal function, the method of randomly assigning links will lead to the optimal link and function assignment in time. For the random interconnects to achieve perfection would be expected to take exponential time, by trial and error, so this has limited practicality.

Once a configuration is reached that is always correct, its components will always receive positive reinforcement and have no incentive to change their computations. The only method by which the configuration could be disrupted, is the transmission of the New-Function signal (see Section 3.15) as lower level nodes attempt to improve their performance. The momentary lack of confidence in the structure leaves it vulnerable to undesirable changes. Except for this weakness, the network will not leave a global maxima.

5.9 Parallelism

The major operations in the network can be conducted in parallel. The exception is the delay due to the propagation of signals through the network. The length of the propagation delay is proportional to the height of the network. The parallel processing at the nodes is possible because nodes require only local knowledge to function. The data on alternative computations is maintained locally at each node. Decisions regarding reinforcements can be made strictly on the basis of information from the local links. The only global control required is a clock to regulate the phases of each cycle.

5.10 Restrictions

A major restriction of the network is the restriction on the nodes to computing boolean functions. This poses no inherent restriction on the capability of the network, since any computable function has an equivalent boolean function. For this reason and simplicity, this investigation has only considered boolean computations at the nodes. In practice, the boolean function limitation restricts the fan-in of the network to two. Computing arbitrary boolean functions on more than two variables quickly becomes impractical. The low fan-in will require tall networks for more complicated problems. It is not clear how well the network will scale with increasing network height. In principle, there is no reason why the decision method cannot be applied to more conventional sum-of-weights-and-compare-to-threshold models. Such models are unable to compute the exclusive-or and equivalence functions, but this drawback is outweighed by allowing a greater fan-in which allows shorter networks to perform the same task.

The exploration of alternative hypotheses is greatly simplified from the knowledge that inverting a computation changes a reward to punishment and vice versa. However, this prevents the network from solving the two-arm bandit problem. The two-arm bandit problem is characterized by forcing the network to choose between two good alternatives (or two bad alternatives). While both alternatives are good, one alternative is better than the other. A network operating on the assumption that the alternative to a good choice is a bad choice will be unaware that the alternative is actually a better choice. The CONSENSUS network would be unable to solve such a problem. A solution to this problem would require that the network actually attempt the alternative computations, but this possibility has not been explored.

5.11 Issues Avoided

No attempt has been made to deal with sequences of input vectors. The model has no capability to memorize state, and its behavior must therefore be a function of the current input vector only. The issue of allowing cycles in the network has also been avoided. The choice of a hierarchical organization prevents the occurrence of cycles. The issue delayed reinforcement has been avoided since the network assumes all reinforcement is immediate. These are all important issues, but beyond the scope of the present work.

6. Conclusions

The guiding principle in CONSENSUS has been that decisions should be deferred until sufficient evidence accumulates to make an informed decision. It is the basis for the following features which most distinguish CONSENSUS from other neural networks.

The use of statistical inference for the classification method reflects this guiding principle by waiting for the accumulation of statistical evidence to support any proposed change. Changes are not made until the accumulated statistical evidence gives overwhelming support to any proposed change, thereby minimizing the chance of error. The classification method seeks to make a small number of major changes each with good confidence.

A few large changes are made each with good confidence. This is in contrast to more conventional systems which rely on the accumulation of many small changes each made with little confidence. These methods rely on the errors to cancel each other out. Since any given change may be in error, changes must be kept small. This restricts them to contemplating only very local changes. Since we have greater confidence in our changes we may contemplate more global changes.

Nodes have an awareness of their role in the network. They seek to maximize the flow of information through the network. This is accomplished by avoiding redundant connections and avoiding the computation of constants.

The notion of confidence in conjunction with the reinforcement scheme seeks to translate global reinforcements into meaningful local reinforcements without transmitting reinforcements of dubious value. Uninformed nodes do not issue reinforcement since they do not possess confidence. Nodes possessing confidence reinforce their subordinates based on their understanding of the role of themselves and their subordinates.

The grouping of nodes into communities exploits the fact that often the knowledge of the group exceeds that of any of its members. Obtaining perfect knowledge at a node is often extremely difficult, but obtaining good knowledge is much easier. The majority of a group of nodes, each possessing independently good knowledge, will have nearly perfect knowledge.

This combination of factors gives a new approach to the "credit-assignment" problem. Simulations of the CONSENSUS system has shown its ability to solve simple learn-by-example problems. Questions regarding the scalability and ultimate capability are currently unresolved.

Aknowledgement

I am particularly grateful to Hans Berliner for his assistance and encouragement.

I. Complexity of Boolean Networks

The worse-case lower and upper bounds on the number of layers required to compute an arbitrary boolean function of n inputs can be calculated if the presence of the spokesman are ignored.

To calculate a worst-case lower bound, recognize that an arbitrary function must be able to access all of the inputs. The restriction of a fixed fan-in, f, from the lower layers allows the calculation of a lower bound, since the uppermost node must have indirect access to every input. A first layer node can compute a function of up to f inputs, a second layer node can compute a function function of up to f^2 inputs, and the general case is that a nth layer node can compute a function of up to f^n inputs. From this it can determined that CEIL(LOG$_f$ n) layers are required to compute a function of n inputs. In the special case of f or fewer inputs, one layer is required. This allows the computation of constants which the input lines are not capable of individually. From this a worst-case lower bound of MAX (CEIL (LOG$_f$ n), 1) can be placed on the number of layers required.

To calculate a worst-case upper bound, imagine the construction of an AND-OR (Disjunctive Canonical Form) network. This requires two steps, the first to calculate up to 2^n minterms, and the second to OR the minterms together. Each minterm could be a function of all n inputs. Several layers of nodes are needed each computing the AND function, except in the first layer where some of the inputs may need to be negated. The minimal number of layers is a function of the fan-in of the nodes. From the lower bound calculations, CEIL(LOG$_f$ n) layers will be required. To do the second part, several layers of nodes computing OR are required to combine the minterms. CEIL(LOG$_f$ 2^n) layers are required to OR together 2^n minterms. This can be simplified to CEIL(n LOG$_f$ 2) layers. This could also be achieved by using an OR-AND (Conjunctive Canonical Form) network. By using the form requiring the fewest minterms or maxterms, at most $2^{(n-1)}$ minterms or maxterms are needed. Again a minimum of one layer is required to handle the case of f or fewer inputs. By summing the requirements for the two phases, an upper bound of MAX ((n - 1) * CEIL (LOG$_f$ 2)) + CEIL (LOG$_f$ n), 1) layers can be determined.

It can also be shown that as f increases to $2^{(n-1)}$, the upper bound decreases to two layers, one for the AND and one for the OR since a single layer is capable of performing each of these functions.

References

[1] Allen, Arnold O.
Probability, Statistics, and Queueing Theory with Computer Science Applications.
Academic Press, New York, NY, 1978.

[2] Ballard, D.H., Hinton, G.E. & Sejnowski, T.J.
Parallel Visual Computation.
Nature (306):21-26, 1983.

[3] Barto, A.G., Sutton, R.S. & Brouwer, P.S.
Associative Search Network: A Reinforcement Learning Associative Memory.
Biological Cybernetics (40):201-211, 1981.

[4] Barto, A.G., Sutton, R.S., & Anderson, C.W.
Neuronlike Elements that Solve Difficult Learning Control Problems.
IEEE Transactions on Systems, Man, & Cybernetics , 1983.

[5] Derthick, M.A.
Variations on the Boltzmann Machine Learning Algorithm.
Computer Science Department CMU-CS-84-120, Carnegie-Mellon University, Pittsburgh, PA, Aug, 1984.

[6] Fahlman, S.E.
Three Flavors of Parallelism.
In *Proceedings of the Fourth Conference of the Canadian Society for Computational Studies of Intelligence.* Canadian Society for Computational Studies of Intelligence, Saskatoon, Saskatchewan, May, 1982.

[7] Feldman, J.A., & Ballard, D.H.
Connectionist Models and Their Properties.
Cognitive Science (6):205-254, 1982.

[8] Feldman, J.A.
Dynamic Connections in Neural Networks.
Biological Cybernetics (46):27-39, 1982.

[9] Geman, S. & Geman, D.
Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images.
IEEE Transaction on Pattern Analysis and Machine Intelligence (6):721-742, 1984.

[10] Hinton, G.E. & Anderson, J.A. (Eds.).
Parallel Models of Associative Memory.
Erlbaum, Hillsdale, NJ, 1981.

[11] Hinton, G.E. & Sejnowski, T.J.
Optimal Perceptual Inference.
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition , June, 1983.

[12] Hinton, G.E., Sejnowski, T.J. & Ackley, D.H.
Boltzmann Machines: Constraint Satisfaction Networks that Learn.
Computer Science Department CMU-CS-84-119, Carnegie-Mellon University, Pittsburgh, PA, May, 1984.

[13] Hinton, G.E.
Distibuted Representations.
Computer Science Department CMU-CS-84-157, Carnegie-Mellon University, Pittsburgh, PA, 1984.

[14] Holland, J.H. & Reitman, J.S.
Cognitive Systems Based on Adaptive Algorithms.
Pattern-Directed Inference Systems.
Academic Press, New York, NY, 1978, pages 313-329.

[15] Hopfield, J.J.
Neural Networks and Physical Systems with Emergent Collective Computational Abilities.
Proceedings of the National Academy of Sciences USA (79):2554-2558, 1982.

[16] Hummel, R.A. & Zucker, S.W.
On the Foundations of Relaxation Labeling Processes.
IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI-5):267-287, 1983.

[17] Kirkpatrick, S., Gelatt, C.D. & Vecci, M.P.
Optimization by Simulated Annealing.
Science (220):671-680, 1983.

[18] Marr, D. & Poggio, T.
Cooperative Computation of Stereo Disparity.
Science (194):283-287, 1976.

[19] Minsky, M. & Papert, S.
Perceptrons.
MIT Press, Cambridge, Mass, 1969.

[20] Pfeiffer, P.E. & Schum, D.A.
Introduction to Applied Probability.
Academic Press, New York, NY, 1973.

[21] Rumelhart, D.E. & Zipser, D.
Competitive Learning.
Cognitive Science (9), 1985.