FEATURE GENERATION AND SELECTION BY A LAYERED NETWORK OF REINFORCEMENT LEARNING ELEMENTS: SOME INITIAL EXPERIMENTS

Charles W. Anderson

Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01003

COINS Technical Report 82-12

Acknowledgement

Andrew G. Barto and Richard S. Sutton designed the learning rules, guided the development of the network architecture used in the experiments, and provided helpful comments concerning the issues and clarity of this report. Richard Sutton also contributed to the computer graphics used in the figures. This research was supported by the Air Force Office of Scientific Research and the Air Force Avionics Laboratory through Contract No. F33615-80-C-1088.

Abstract

The adaptive network approach to pattern classification was persued at an optimistic rate in the 1950's and 60's. As the complexity of the tasks increased, networks with layered architectures were formulated to allow the construction of nonlinear input-output mappings. However, progress with this approach was practically halted by the lack of general learning algorithms with which layered networks could acquire the appropriate mappings. Adaptive elements using error-correction techniques are not of sufficient complexity to allow the type of inter-element cooperativity that is necessary for efficient learning to occur in layered networks. We have found that the desired cooperativity can arise by constructing networks from elements that employ a more powerful form of reinforcement learning. This is illustrated in this report by presenting a layered network that learns to perform some simple control tasks by simultaneously 1) developing an input representation with the appropriate balance between discrete and continuous components, and 2) learning the appropriate control actions as a function of the input representation.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

The determination of appropriate representations for complex problem-solving systems remains one of the primary issues confronting investigators in the field of Artificial Intelligence (Barr and Fiegenbaum, 1981). Although most problem domains can accommodate many different types of representations, usually a particular representation will afford the most direct solution to a particular problem. One way of distinguishing the various representations is by classifying them as being either discrete, continuous, or a combination of the two (Berliner and Ackey, 1982). Semantic networks, frame-based systems, and production rule systems are examples of systems employing discrete state and control structure representations; classical control systems using real-valued state vectors and analytic functions are examples of systems employing continuous state and control structure representations. As Berliner and Ackley (1982) have stated, there are many problem domains for which a purely discrete or continuous representation will not be satisfactory. If it is impossible to determine a priori an appropriate representation, then it would be highly desirable to employ a problem-solving system that is capable of developing the best balance between the discrete and the continuous extremes. An attempt at designing such a system is Waterman's (1970) production-rule system which generates rules that partition a problem's state space.

Little effort in the field of AI has been made to duplicate or extend Waterman's results. So to begin a study of the issues and possible techniques involved in the development of representations with the proper 'discrete/continuous' balance, we should look for similar lines of research in other fields or domains. The control of dynamical physical systems is a specific problem-solving domain in which the representation problem has received some attention. In this domain, the representation problem is often referred to as part of a pattern classification problem or a feature selection problem. The application of pattern classification techniques has long been considered a necessary step in the design of "intelligent" control systems that are capable of performing tasks in complex environments (Fu, 1970; Mendel and McLaren, 1970; Mendel and Zapalac, 1968; Saridis, 1977; Widrow and Smith, 1964).

This report reviews some of the issues that arise in the control systems domain that are relevant to the representation problem, particularly the development of representations with a balance between continuous and discrete components. Following this review, a control system is presented that learns, through interaction with its environment, to encode its input into an appropriate set of features to which goal-directed actions are associated. The control system is a layered Associative Search Network, or ASN, defined by Barto, Sutton, and Brouwer (1981), with a structure similar to that used by Barto, Anderson, and Sutton (1982). This layered ASN structure represents one of our initial attempts to investigate the potential of this network

approach for problems involving adaptive representation development. Although we will be dealing in this report specifically with the control system domain and with representations that would not be considered sophisticated by many AI researchers, it is our belief that many of the issues addressed and solutions proposed are relevant to other problem-solving domains.

2.0 THE "DIVIDE AND CONQUER" APPROACH TO CONTROL SYSTEMS

The classical approach to the design of a control system is to analyze the plant to be controlled, referred to here as the environment, and to arrive at an analytic function defining the control surface, i.e., an expression that produces a control action when given the state of the environment. The potentially great complexity of the environment has led to the investigation of control systems with adaptive, or learning, capabilities. By including learning capabilities in a control system, less analysis of the environment is required. The simplest approach is to give a control system the ability to alter the parameters of its control surface function. The environment has to be analyzed to the point where a parameterized class of possible control surface functions can be defined. Saridis (1977,1979) reviews several methods by which this type of learning control system, which he terms "parameter-adaptive", can be designed.

However, in order to increase the applicability of a learning control system to more general (i.e., less analyzable) environments, the structure of the control surface function, in

addition to its parameters, must be alterable (Kaufman, 1967; Wiener, 1967). Many methods for accomplishing this lead to searches over very large spaces of possible function structures. This motivated a "divide and conquer" approach: The input space of the control system is divided into regions, and one structure of limited complexity is associated with each region or set of regions. Thus, the single search over the large space of complex structures can be replaced by many searches over smaller spaces of simpler, and sometimes linear, structures. A trade-off arises between the degree of complexity allowed for each structure and the number of regions into which the input space is divided, or between the complexity of the structures and their quantity.

3.0 STORAGE VS. COMPUTATION

"Storage vs. computation" refers to the general type of trade-off encountered with the "divide and conquer" approach. (Raibert (1977) labels this trade-off as "table look-up vs. analytical equations.") At the storage extreme, every possible input vector, or group of similar input vectors, is represented by a location in a storage device, and in each location is stored the specification of a control action. To choose an action for a given input vector, a table look-up scheme is employed simply to refer to the appropriate storage location. The only computation required by this approach is the determination of which location to reference. On the other hand, the computational extreme applies an analytic function to the current input vector to calculate the control action to be taken.

Both approaches have advantages and disadvantages. The storage method results in a control surface that responds with an action quickly, due to the limited amount of computation required. It also suggests a straightforward generalization procedure: changes in the action specification at one location can be extrapolated to neighboring locations. The storage method is directly extensible to more complex tasks by decreasing the number of input vectors that are represented by each location in storage, with the limit of one location, and thus one action, for each possible input vector. A potential problem with the storage method is the possibly large amount of experience required to learn a good control surface. This problem could be alleviated by grouping the input vectors in a manner that results in beneficial generalizations; that is, by representing in one storage location those situations that are likely to produce the same action. This implies a knowledge of the control surface before learning, which cannot be assumed for complex tasks.

For simple control surfaces, the computational approach is superior. If the structure of the analytic function is known a priori, then the generalization resulting from a few trials will greatly facilitate further learning. For example, if the desired control surface function is known to be linear, then training to a relatively small number of input vectors might be sufficient to learn the entire function. However, very few methods have appeared that are capable of learning the correct structure of the function, and that can, therefore, perform complex tasks. In general, the computational extreme will

require far less storage than the storage extreme, but will tend to yield slower response times.

The storage and computational approaches are actually the two extremes of a spectrum of methods, i.e., it is possible to devise methods that combine some aspects of both storage and computation. For example, storage locations can store function specifications that produce actions, rather than storing the action specifications themselves. Also, some input vectors could be represented by more than one location, in which case the resulting action could be a combination of the actions calculated within each storage location. Therefore, for a given task, a compromise between these two extremes might be found that would result in the most efficient performance.

The storage vs. computation issue presented itself to the designers of learning control systems when they had to decide on the quantity and placement of the regions in the input space. Michie and Chambers (1968) took the extreme storage approach by providing sufficiently small, disjoint regions so that one control action could be associated with or "stored" in each region. Others assumed a compromise between storage and computation by associating simple analytic expressions with each region (Raibert, 1977,1978) or by using overlapping regions and computing an action by combining the actions associated with each region containing the current input vector (Albus, 1979; Mendel, 1970; Mendel and Zapalac, 1968; Widrow and Smith, 1964).

In all of these references, the regions are defined before any learning occurs and remain fixed, although Michie and Chambers (1968) mention the possibility of allowing the regions to "split" apart and "lump" together as learning proceeds. It would certainly be advantageous to the designer if the role of specifying or altering the regions was given to the learning control system, resulting in still further generality in the possible environments in which the system could satisfactorily perform.

It is largely this objective that has persuaded many investigators of control theory to draw upon results in the field of pattern classification. The relation between these two fields is a close one, since a control problem can be recast as a pattern classification problem by defining the input vectors as patterns and the control actions as classes (Fu, 1970; Mendel and McLaren, 1970; Mendel and Zapalac, 1968; Saridis, 1977; Widrow and Smith, 1964). In fact, within the pattern classification field we find a process, that of feature extraction, that endows a classification system with many of the "divide and conquer" aspects discussed above. A feature extractor groups patterns into sets having common feature values, just as the input vectors to a control system can be grouped into regions. Thus, a rich source of information and techniques is available as a guide in the search for mechanisms with which a control system can learn to divide its input space. Some of the issues concerning feature selection and their relation to control theory will now be discussed.

4.0 OPEN-LOOP VS. CLOSED-LOOP LEARNING

The solutions to pattern classification problems and control problems can be categorized in several ways, one of which is open-loop vs. closed-loop learning. This dichotomy, as used here, refers to whether or not any knowledge about the attempted classification or action is fed back to the system as learning proceeds. A method employing open-loop learning, such as clustering, does not use this type of knowledge. In the learning control systems mentioned previously, the procedures used to divide the input space into regions were certainly open-loop, since they were fixed a priori and did not change as the systems learned.

The initial approach towards feature selection taken by investigators in pattern classification was to ignore the results of the classification task and attempt to find specific aspects that were present in a large number of the patterns (Block, Nilsson, and Duda, 1964; Fukushima, 1969; Nagy, 1969; Riseman, 1971; Sears, 1965; see Gose, 1969 for further references concerning this approach and others to be discussed below). Their motivation was to reduce the number of components in the pattern that the classifier must deal with. However, the features that resulted from these methods were often capable of reconstituting the original patterns, indicating that the features were possibly much greater in number and/or complexity than the classification tasks required.

Other investigators realized that a feature selection process should depend on the information that is available from a classification attempt, making it a closed-loop process. Minsky (1963) stressed this when he defined the "teleological requirements of classification" by stating that "useful classifications are those which match the goals and the methods of the machine." The same concern is seen in MacKay's (1969) suggestion that the "problem of pattern recognition is always ill-defined until the class of agents have been specified." Holland (1969) observed that "different environments or different goals will determine different sets of critical features." Arbib (1972) coined the phrase "action-oriented perception" to label a perceptual process that utilizes the information resulting from its use (by an action-selecting mechanism) to alter its representational analysis.

Thus for problems that are not completely analyzable, the pattern classification process must be sensitive to the results of its classifications and be able to alter the classifier and the feature extractor in order to perform well. For control theorists, this means that divisions of the input space must be regulated by the learning control system.

To develop pattern classification techniques that are useful to the control theorist, we can concentrate on only those methods that employ some knowledge of the results of a classification or action. The following section presents and contrasts the two forms that this knowledge-of-results can take.

5.0 ERROR-CORRECTION VS. REINFORCEMENT LEARNING

The specification of a pattern classification or a control problem determines whether error-correction or reinforcement learning is applicable. In particular, the amount of a priori knowledge about the correct classifications or actions makes this distinction. The typical pattern classification problem includes a set of patterns and their appropriate classifications, thus permitting an error between a system's classification attempt and the correct classification to be computed. In this case, the system can apply proven error-correction techniques to try to force the error to zero. However, for the typical control problem, the correct actions are unknown and the objective is to truly discover which actions will drive the environment into a goal state. Here only a reinforcement signal is available that might, for example, indicate how close the current state is to a goal state. Errors cannot be computed since the correct control actions are unknown (although the desired environmental state may be known).

The feature selection problem in pattern classification is similar to the typical control problem in that the correct set of features is usually unknown a priori, preventing the computation of an error signal and the use of the standard error-correcting pattern classification procedures to learn an appropriate set of features. If such a correct set of features were known, it could simply be built into the pattern classification system. This problem has been known for some time. Nilsson's (1965) analysis of layered machines, where the

first layer, or layers, could be viewed as computing features, showed that the use of error-correction training methods was limited. Due to a lack of understanding about how a specific parameter change affects the performance of the other layers, it is difficult to determine which parameters in which layer should be adjusted as training proceeds (Anderson and Hinton, 1981; Barto, Anderson, and Sutton, 1982). Minsky and Papert (1969) became aware of this problem in their study of layered networks of Perceptrons. They surmised that a training algorithm for a layered network might involve the magnitude of a reinforcement signal, rather than an error signal.

Reinforcement learning has been applied to feature selection by others in the pattern classification field. Their approach can generally be characterized as a generate-and-test approach -- new features are generated and added to the current feature set, which is subsequently tested by applying the classification procedure using the new feature set. Various techniques for the generation of features have appeared. Some have simply supplied an initial set of features from which new features were chosen (Lewis, 1962) and others randomly altered or combined old features (Kamentsky and Liu, 1963; Klopf and Gose, 1969; Uhr and Vossler, 1963). Chow and Liu (1966) applied the generate-and-test viewpoint to the adaptation of the structure of a pattern recognition system, which can be considered as the adaptation of the feature set. Their demonstration contains a significant variation of the random methods of generation mentioned above: Rather than generating completely new

structures randomly, slight alterations were made to previous structures that had been shown to be useful. In this way, initially simple (linear) structures increased in complexity until the desired level of performance was achieved. The genetic algorithms developed by Holland (1975) are other examples in which slight alterations are used to systematically increase the complexity of the structure.

Although reinforcement learning and its application to feature selection has received much attention, its utility to control theory is not direct. Reinforcement learning has been applied to feature selection, but due to the nature of the typical pattern classification problem, the classifier can still use error-correction learning. Very little work has appeared concerning how two processes, such as feature selection and classification, can cooperate when both are driven by reinforcement learning. In the typical control problem, the action-selection mechanism, analogous to the classifier, must employ reinforcement learning. To develop general learning control systems we must deal with the interaction between the region formation process and the action-selection process as they learn under the influence of reinforcement feedback. The remainder of this report presents a study of this issue with the implementation of a learning control system that, through reinforcement learning, divides its input space into regions and associates with them the actions that produce good reinforcements.

6.0 A LAYERED NETWORK THAT LEARNS FEATURES AND ACTIONS

6.1 Introduction

Barto, Sutton and Brouwer (1981) have defined a reinforcement learning system, called the Associative Search Network (ASN), that posesses all of the desirable qualities expressed above: closed-loop learning, reinforcement learning, and cooperativity among its processes. The ASN is designed to search through the space of action vectors and with each input vector associate the "best" action ("best" refers to the highest payoff, to be defined later). An ASN is composed of elements, based on the theory of Klopf (1972, 1979, 1982), which are themselves reinforcement learning systems. Single layers of these elements have been investigated and their applicability to simple control problems has been demonstrated (Barto, Sutton, and Brouwer, 1981; Barto and Sutton, 1981a).

In the past, networks of two or more layers have been applied to pattern classification and control problems (Chow, 1966; Nilsson, 1965; Minsky and Papert, 1969; Rosenblatt, 1962; Widrow and Smith, 1964) with the additional layer or layers performing a feature extraction process. However, learning rules for implementing a feature selection process in these layers were very limited, and more modern methods are still tailored to specific problems. Barto and Sutton (1981b) suggest that the types of elements used in these early networks were a hindrance to the development of robust learning systems. They believe that much can be gained by using elements, such as the ASN element and its extensions, that are alone capable of solving control

problems that are difficult along certain dimensions. In fact, initial work has shown that a two-layered ASN can solve problems requiring the selection of features that are simple combinations of the components of the original input (Barto, Anderson and Sutton, 1982). The results of further experimentation with a two-layered ASN as it is applied to simulated control tasks are presented following a description of the control tasks and the ASN.

6.2 Description of the Tasks

The solution to the type of control problems considered here requires the control system to produce the "best" action for every input from the environment. This general formulation is shown in Figure 1. The current situation of the environment determines the input to the system, which then applies a situation-to-action mapping (the control surface) to generate an action. The action then affects the environmental situation. The critic within the system evaluates the new situation by computing a performance index, or payoff, based on the situation or how the situation is changing. The payoff is used to reinforce the actions just taken. After an appropriate mapping is learned, the payoff input is no longer required. We split the situation-to-action mapping into a feature encoder and a feature-to-action mapping as in Figure 2. Note that the same payoff (evaluation) goes to both the feature encoder and the adaptive feature-to-action mapping.

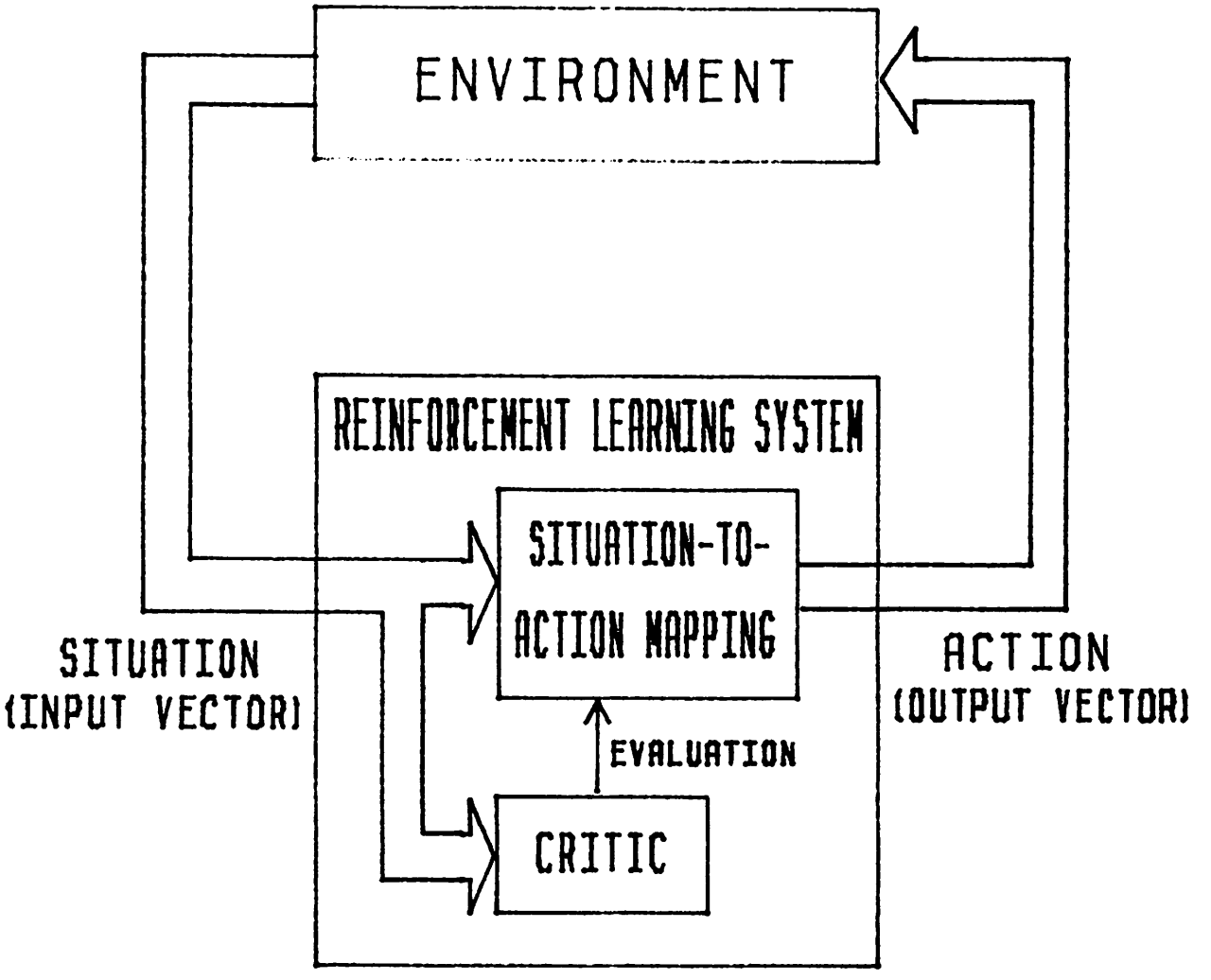

Figure 1: A General View of a Reinforcement Learning System.

A reinforcement learning system is here defined as a system that 1) receives an input vector from its environment, 2) implements a situation-to-action mapping (control surface) that is updated as a function of the evaluation, and 3) produces an output vector that is applied as a control signal to the environment.

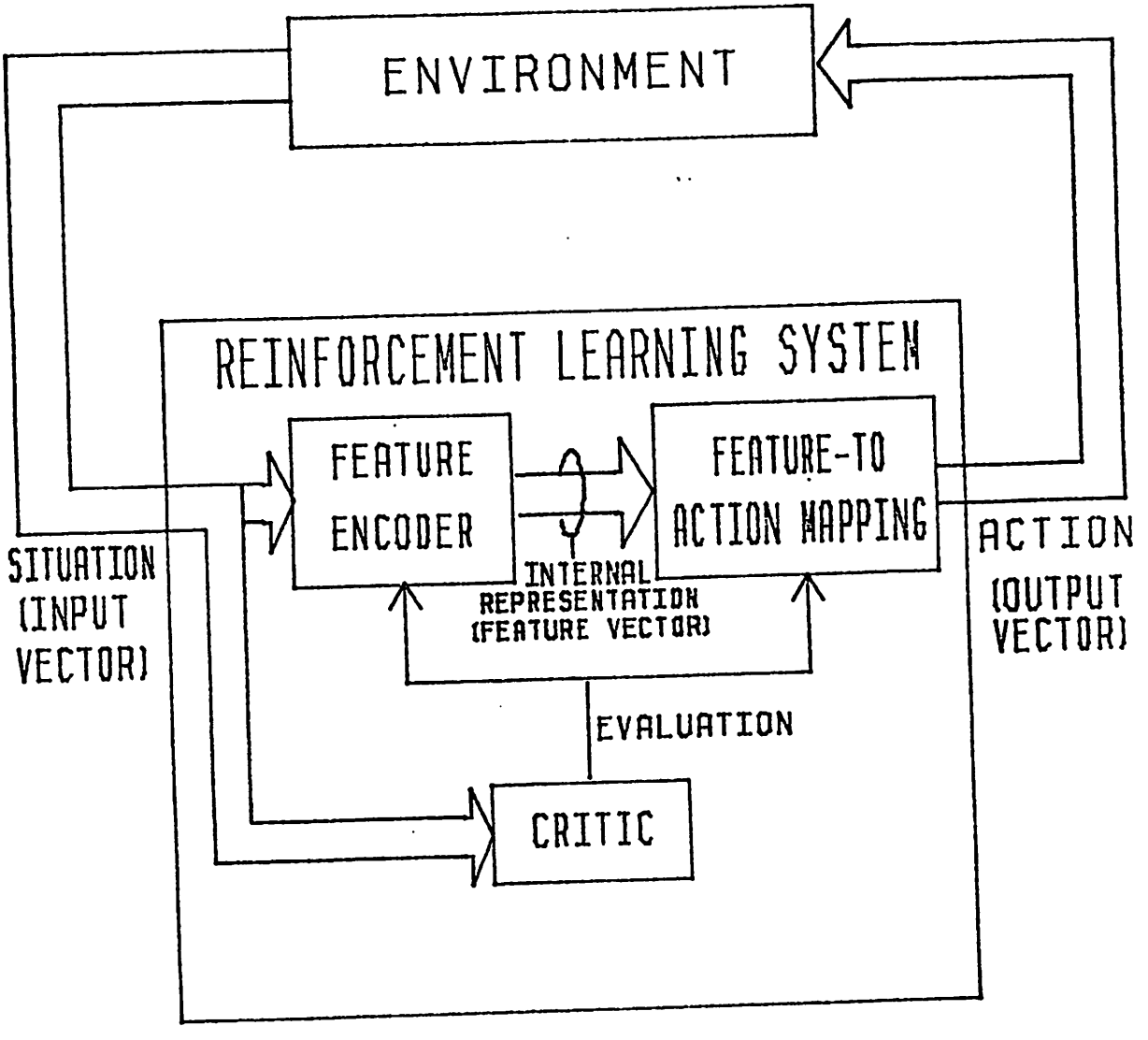

Figure 2: Addition of a Feature Encoding Mechanism.

A feature encoding mechanism is inserted between the input to the reinforcement learning system and the situation-to-action mapping.

This type of control problem is greatly simplified compared to more general problems by the availability of a payoff signal at every time step that always evaluates the action made at the preceding step. We have temporarily adapted this simplification to allow us to focus on basic feature-selection issues. Work is currently in progress concerning the use of the ASN for tasks involving delayed evaluations (Barto, Sutton, and Anderson, 1982). The ASN is augmented by a component that adaptively develops a more useful evaluation function than is originally supplied in the control task. Future reports will consider this additional complexity.

The layered ASN is applied to simple control tasks using a two-dimensional input, or situation, vector (x_1,x_2) and a two-dimensional output, or action, vector (u_1,u_2). A payoff, z, is calculated from the input vector indicating how desirable the current situation is. This system is shown in Figure 3. A payoff function is defined by constructing a surface over the two-dimensional input space whose height at each point, or situation, (x_1,x_2) is the desirability of the situation. For example, if one situation is more desirable than all others, then the payoff function would appear as in Figure 4a (see Experiment 1). If one situation is desirable while another is to be avoided a payoff function like that in Figure 4b would be used (see Experiment 4).

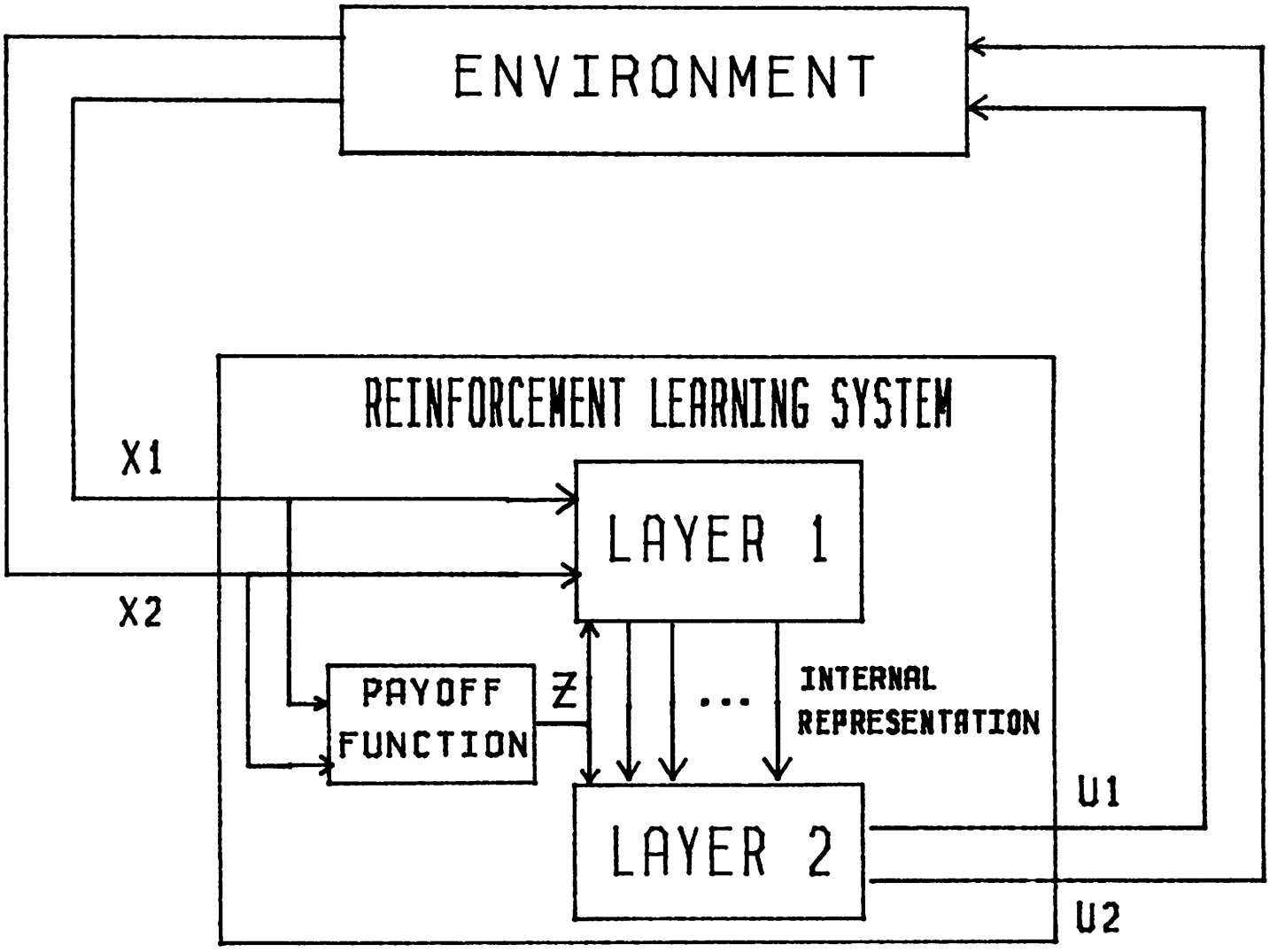

Figure 3: A Two-layered Network of Adaptive Elements as a Reinforcement Learning System.

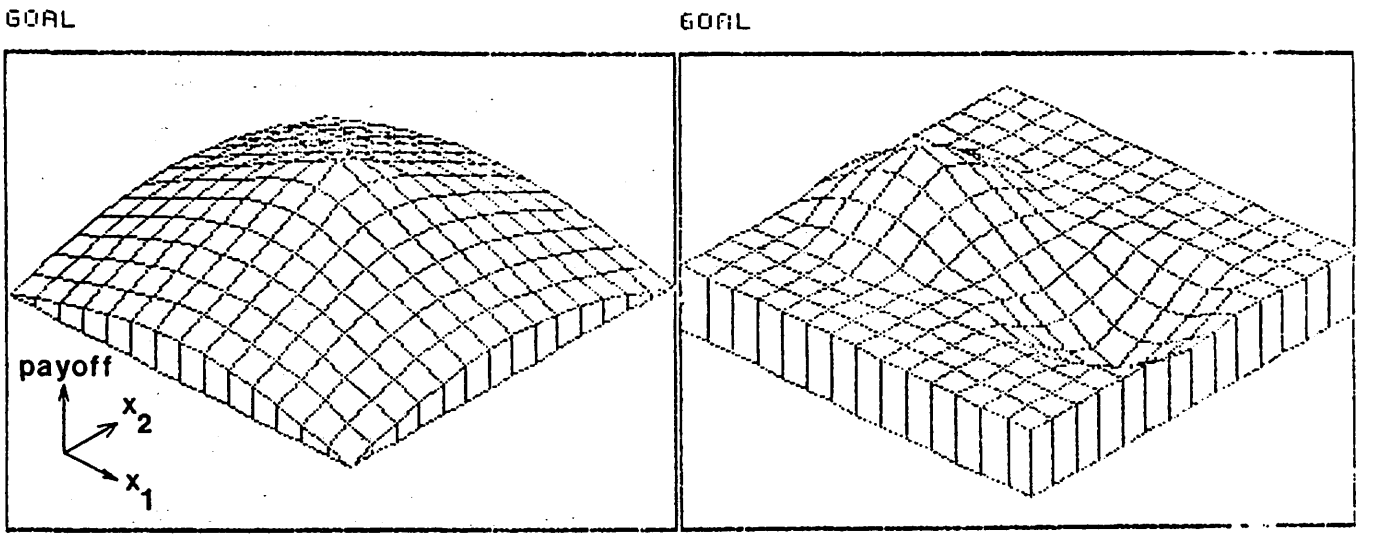

Figure 4: Two Possible Payoff Functions.

A payoff function is defined for all points in the two-dimensional input space. The height of the surface indicates the desirability of each situation.

6.3 Layer 2 of the Network

Let us look at the second layer of the ASN in isolation. It is actually a network of adaptive elements, with each element receiving the same inputs and generating one output that is one component of the output vector. The structure of layer 2 is shown in Figure 5.

Four adaptive elements are used in this layer. They are represented by the four numbered circles with inputs coming in from the left and outputs going out to the right. The vertical input lines (in this case the output lines from layer 1) intersect each element's input pathway. Associated with each intersection is a real-valued weight $w2_{i,j}(t)$ designating the strength of the connection of input $r_i(t)$ with element j's input pathway. In the figures, the magnitude of each weight is shown as the radius of a circle or disk at the intersection, with a positive weight being an open circle and a negative weight being a filled-in disk. For example, two positive weights and one negative weight are shown in Figure 5. The output of each element is computed by first evaluating the function

$$s2_j(t) = \sum_{i=1}^{n} w2_{i,j}(t)\, r_i(t) + Noise_j(t),$$

for $j=1,\ldots,4$, where $Noise_j(t)$ is a normally distributed random variable of mean 0.0 and a standard deviation of 0.01 for all simulations. The outputs of the elements are then given by the following equations:

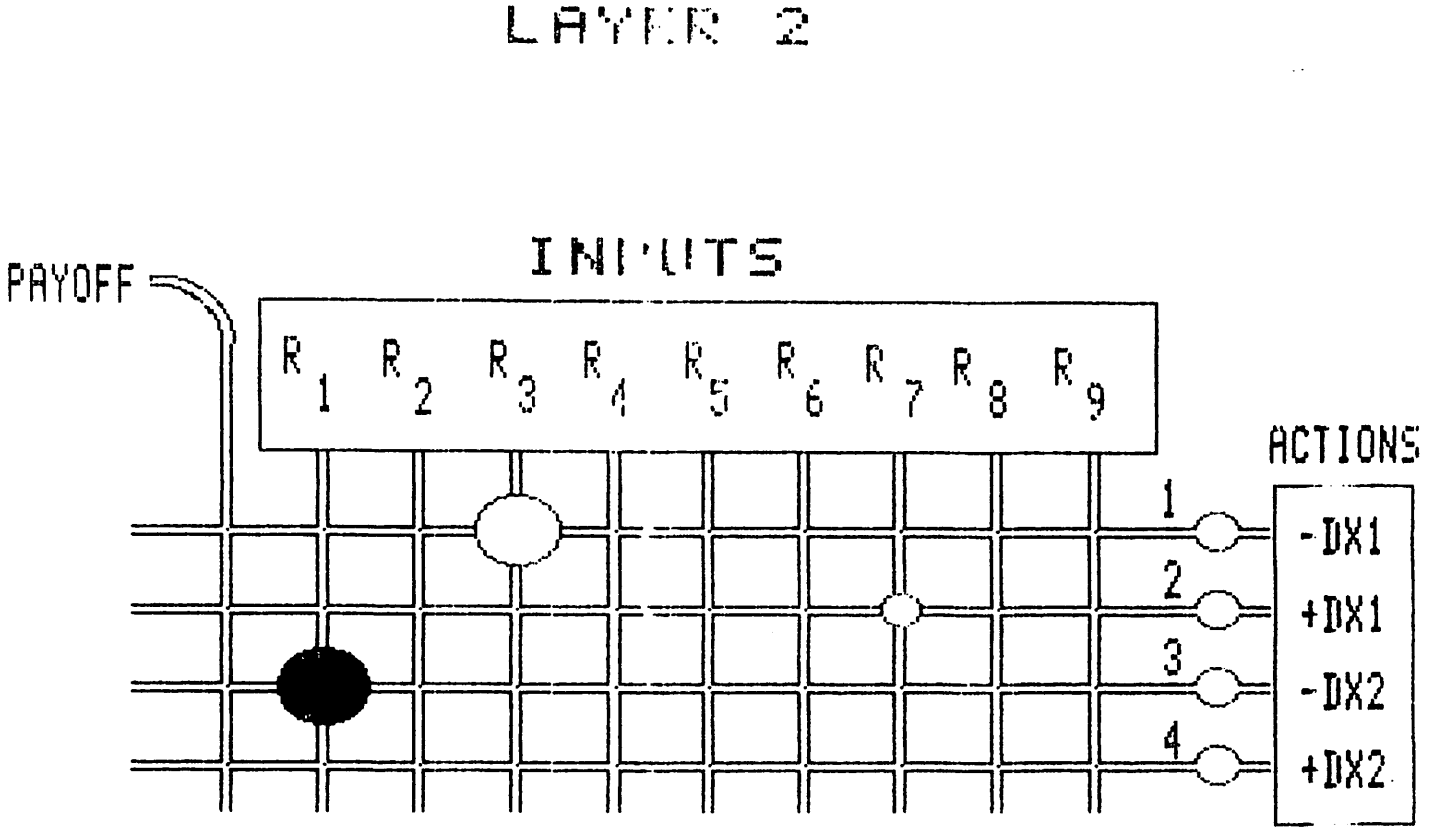

Figure 5: Layer 2 of the Network.

The second layer of the network receives the payoff and the features r_1 through r_9 from the first layer and computes the action, or output, vector. The circles represent the values of the weights of each adaptive element. Their radii indicate magnitudes and their shading indicates sign, i.e., open circles represent positive weights and filled-in disks represent negative weights.

$$y_1(t) = \begin{cases} 1, & \text{if } s2_1(t) - s2_2(t) > e \\ 0, & \text{otherwise} \end{cases}$$

$$y_2(t) = \begin{cases} 1, & \text{if } s2_2(t) - s2_1(t) > e \\ 0, & \text{otherwise} \end{cases}$$

$$y_3(t) = \begin{cases} 1, & \text{if } s2_3(t) - s2_4(t) > e \\ 0, & \text{otherwise} \end{cases}$$

$$y_4(t) = \begin{cases} 1, & \text{if } s2_4(t) - s2_3(t) > e \\ 0, & \text{otherwise} \end{cases}$$

with e being 0.01 for all simulations. The outputs of the network, action components $u_1(t)$ and $u_2(t)$, can be -1, 0, or 1 and are computed by

$$u_1(t) = y_2(t) - y_1(t)$$

$$u_2(t) = y_4(t) - y_3(t).$$

A very simple environment is assumed for this demonstration. The environment is represented by the following equations:

$$x_1(t+1) = x_1(t) + a\, u_1(t)$$

$$x_2(t+1) = x_2(t) + a\, u_2(t),$$

where a = 0.2 for all simulations. Thus, the action $(u_1(t), u_2(t))$ directly effects a change in the situation. Action

vectors will be represented as ($+dx_1$ or $-dx_1$, $+dx_2$ or $-dx_2$). This simple environment is used to facilitate the conprehension of what situation-to-action mapping would be required to solve each control task as specified by the payoff function.

At each time step the inputs are received and the outputs are computed. One time step later the outputs effect a change in the environmental situation, and the payoff, z(t+1), is assigned as the desirability of the new situation. At this time the weights are updated by the equation

$$w2_{i,j}(t+1) = w2_{i,j}(t) + c2\,[z(t+1) - z(t) - s2_j(t)]\;\; y_j(t)\; r_i(t),$$

for i=1,...10 and j=1,...,4, where c2 = 0.4 for most of the simulations. This learning rule is referred to as the "conditional-predictor" rule because the weights are adjusted in a manner that makes each $s2_j$ an estimate, or prediction, of the change in payoff received on the condition that the corresponding element j contributed to the calculation of the action last taken, i.e., on the condition that the element had an output greater than 0. One effect of this rule is to limit the magnitude of the weights. If, for a particular input vector, the best action is always chosen (producing the maximum reinforcement), the weights will increase until the weighted sum, s2, matches the change in payoff, which is a bounded value.

As shown above, each element of the ASN computes its output by applying a threshold function to a weighted sum of its inputs. This places severe restrictions on the complexity of the mappings from the input vector (x_1,x_2) to the action vector (u_1,u_2) that the single-layered ASN can perform. In order to relax these restrictions, another layer is introduced between the input vector and the layer already described, forming the layer 1 in Figure 3.

6.4 Layer 1 of the Network

The role of this layer is to place each input vector into one of several classes, with each class corresponding to one component of this layer's output vector. The classes can be considered as regions of the input space. Thus, layer 1 will implement a feature encoding of the inputs (x_1,x_2). The combination of the first and second layers will be able to form situation-to-action mappings of greater complexity than those formed by a single layer.

Figure 6 shows layer 1 in isolation. Each element of this layer must be able to learn a function that is greater than zero only when the input vector falls within a certain region of the input space. A single element that computes a linear function of its inputs (a weighted sum in this case) and a threshold is only capable of dividing the input space with hyperplanes. To provide the element with the capability of classifying an input vector into a more general type of region, an input representation other than (x_1,x_2) is needed. The

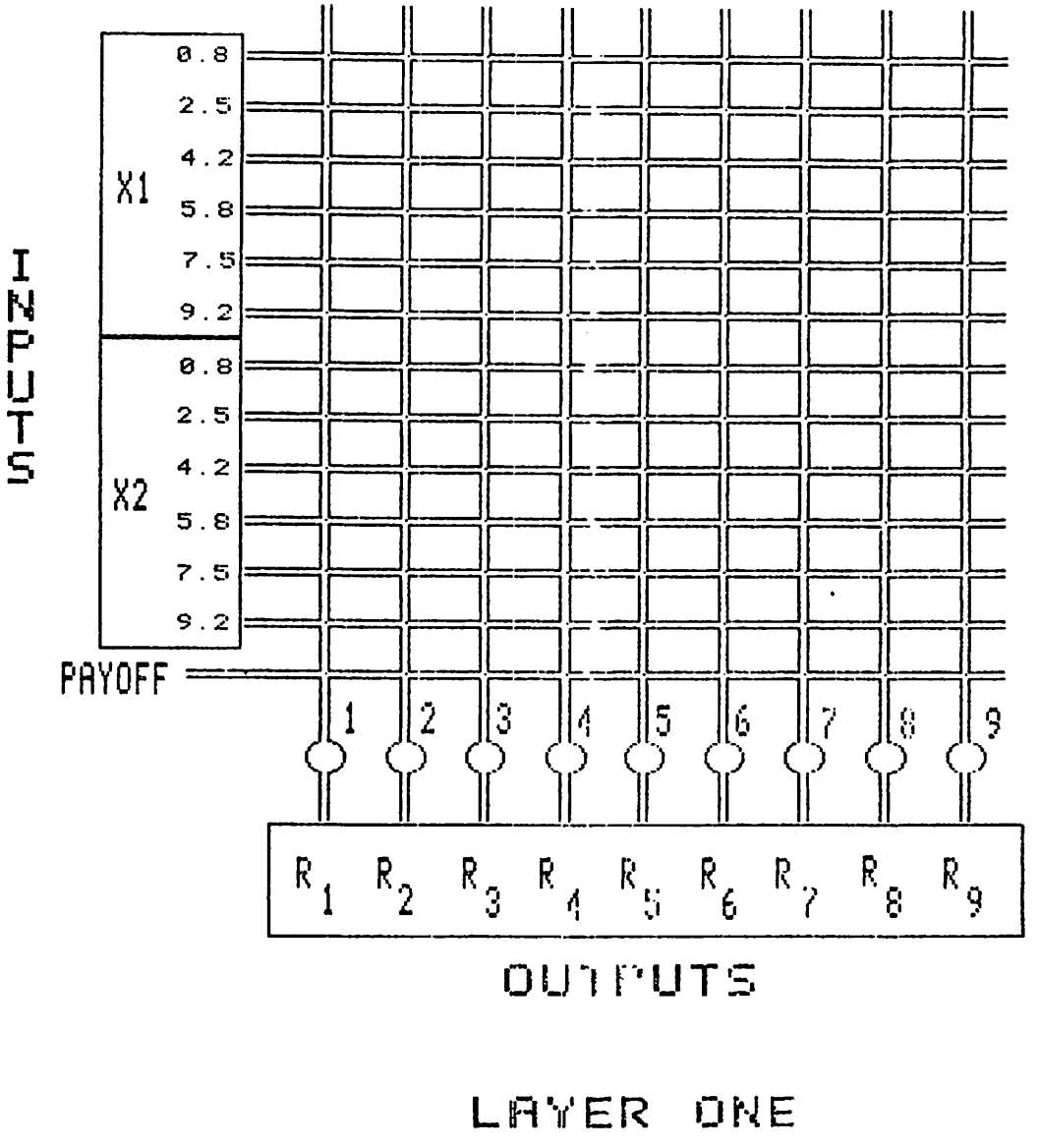

Figure 6: Layer 1 of the Network.

The first layer receives the payoff and the input vector (x_1,x_2) encoded in a manner described in the text. It computes the values r_1 through r_9 which are the components of the input vector to the second layer.

approach taken in this system is to divide each dimension of the input space into six overlapping intervals, for a total of 12 intervals. The new representation consists of the 12 variables $q_1,\ldots,q_{12}$, where values are determined as follows: given (x_1,x_2), the value of q_i, i=1,...,6, is maximal when x_1 is the center of the interval corresponding to q_i; the value of q_i, i=7,...,12, is maximal when x_2 is the center of the interval corresponding to q_i. The values of the q_i decrease as the points x_1 (or x_2) move away from the center of the corresponding intervals. More specifically, the value of q_i, i=1,...,12, is determined by a Gaussian curve of unit variance and mean m_i centered over the interval corresponding to q_i. That is:

$$q_i = e^{(m_i - x_1)/2}, \quad i = 1,\ldots,6$$

$$q_i = e^{(m_i - x_2)/2}, \quad i = 7,\ldots,12.$$

In the simulations to be described below, the values of x_1 and x_2 range from 0.0 to 10.0. The means of the Gaussian curves $m_1,\ldots,m_6$ and $m_7,\ldots,m_{12}$ are assigned 0.83, 2.50, 4.17, 5.83, 7.50, and 9.17.

This representation was chosen as a compromise in the storage vs. computation trade-off. An extreme storage approach could be taken by forming all the cross-product terms of the variables quantizing each dimension, thereby dividing the input space into a grid of overlapping boxes (Albus, 1979) or nonoverlapping boxes (Michie and Chambers, 1969). If each dimension were divided into six intervals as specified above,

then 36 cross-product components would result. In general, the number of components is an exponential function of the number of dimensions, making this approach unsatisfactory for high dimensional problems. On the other hand, an extreme computational approach would entail very few regions and more complicated output functions in the elements. This latter alternative was not chosen because we wished not to alter the basic ASN element. The representation outlined above is an acceptable compromise since the number of components is a linear function of the number of dimensions. This leads to an architecture that is extensible to problems of higher dimensionality. The objective of layer 1 is to allocate its limited resources to forming just those regions that are necessary to solve a given problem. The variables q_i, $i=1,\ldots,12$, serve as a substrate for the expression of a wide class of possible regions.

The output function of each element in layer 1 is given by the equations

$$s1_j(t) = \sum_{i=1}^{n} w1_{i,j}(t)\ \ q_i(t) + Noise_j(t)$$

$$r_j(t) = \begin{cases} s1_j(t), \text{ if } s1_j(t) > 0 \\ 0, \text{ otherwise,} \end{cases}$$

for $j=1,\ldots,10$, where $Noise_j(t)$ is a normally distributed random variable of mean 0.0 and a standard deviation of 0.1. The outputs of these elements are not binary as they are in layer 2. The value of an element's output designates the degree to which

the current input vector is within the region to which the element is sensitive; input vectors near the edge of the region will produce an output value slightly greater than zero, while input vectors near the center of the region will produce greater output values.

The learning rule for layer 1 requires special consideration. If the learning rule of layer 2 were used in layer 1, then many of the elements in layer 1 would learn to respond to the same regions, i.e., if one element of layer 1 became tuned to a region and connected to the appropriate action elements of layer 2, then the other elements of layer 1 would also tend to tune to this region because a high reinforcement would be received while experiencing a signal in that region. Some way of allowing the learning that occurs in one element to influence that occurring in other elements must be employed; a method of "enforcing variety" is needed. One approach to this problem is to assume that the learning (updating the weights) of each element is dependent on the output, or activity, of every element. By adding a neural-like lateral inhibitory mechanism to the layer, all but the most active elements can be suppressed (see Barto, Anderson, Sutton, 1982 for further discussion and references). Another approach is to apply this selection process to the learning rule only and leave the outputs of the elements unchanged. The second approach is used here since it does not place any restrictions on the number of components in the output vector, i.e., the number of features, that can have nonzero (unsuppressed) values. Therefore, layer 1 is capable of encoding

the inputs in a nontopographic, or distributed, manner, which is not the case if the output of only one element of layer 1 is allowed to be nonzero at any given time. The nontopographic encoding allows a large set of possible output vectors from layer 1. In particularly, it allows vectors that contain more than one nonzero component and is thus a more efficient input representation for layer 2: Each input component to layer 2 can be associated with a certain action, and when more than one component is present (nonzero) their associated actions can be combined to produce different actions.

At each time step (defined later), the layer 1 element with the largest output is selected as the one that is eligible for learning. Let that element be element j. Its learning rule is

$$w_{i,j}(t+1) = w_{i,j}(t) + c1\,[z(t+1) - z(t) - s1_j(t)]\; r_j(t)\; q_i(t),$$

for $i=1,\ldots,12$ and $j=1,\ldots,10$, where $c1 = 0.08$ for all simulations. The reinforcement $z(t+1)-z(t)$ is the same value used in the learning rule of layer 2.

The behavior of layer 1 can be illustrated as follows. The output of an element can be viewed as a confidence measure that the current input vector is contained within the region to which the element is sensitive. Only the most confident element is able to update its weights. If a negative reinforcement is received, meaning the action produced by layer 2 was not good

given the situation indicated by the current input vector, then the learning rule alters the weights with the result of lowering the confidence of the selected element (the currently most confident element). This will continue whenever that input vector is encountered, until the confidence of another element becomes greater than the lowered confidence of the updated element. If the action associated with the new element's region of sensitivity is good, then a positive reinforcement is received causing the confidence of that element to increase. This behavior allows the retention of what is learned in different areas of the input space; learning within one region can proceed with minimal effects on the knowledge already acquired in other regions.

It is instructive to consider briefly the difficulties that would arise if error-correction elements, such as Perceptrons, were used in layer 1. Since each such element must be provided with its own error signal, some agency must know from the start what output values these elements should produce for some training set of input signals. It is not enough for this agency to know the correct control actions for each input, but it must also know the correct representation, produced by layer 1. Therefore, the necessary a priori knowledge is not just a function of the environment, but of the controller as well. This places a great demand on the 'teacher' of such a control system since it must specify the correct outputs of every element in the layered network. This problem has been discussed by several investigators in the past, but no general solution has been found

(Nilsson, 1965; Minsky and Papert, 1969). Rosenblatt (1962) considered the use of "backward-chaining" to propagate the effect of the error-correction in the final layer to previous layers. However, for the ASN the only training signal present is the reinforcement, which is the same value for each element in each layer. This enables us to use the same ASN element, whose behavior we have studied in isolation, to construct both layers of the ASN described here.

6.5 Operation of the Combined Layers

Figure 7 illustrates the structure of the entire two-layer system. To illustrate clearly the sequence of operations that take place during one time step, the following list gives the order of execution:

1. Input vector $(x_1(t),x_2(t))$ is received from the environment and payoff z(t) is computed.
2. The weights in both layers, W1(t) and W2(t), are updated.
3. The weighted sums S1(t) are calculated.
4. The layer 1 elements produce the outputs R(t).
5. The weighted sums S2(t) are calculated.
6. The layer 2 elements produce the outputs Y(t).
7. The action $(u_1(t),u_2(t))$ is calculated and applied to the environment.
8. The environment is forced into the new situation $(x_1(t+1),x_2(t+1))$.
9. Repeat steps 1 through 8 after incrementing t by 1.

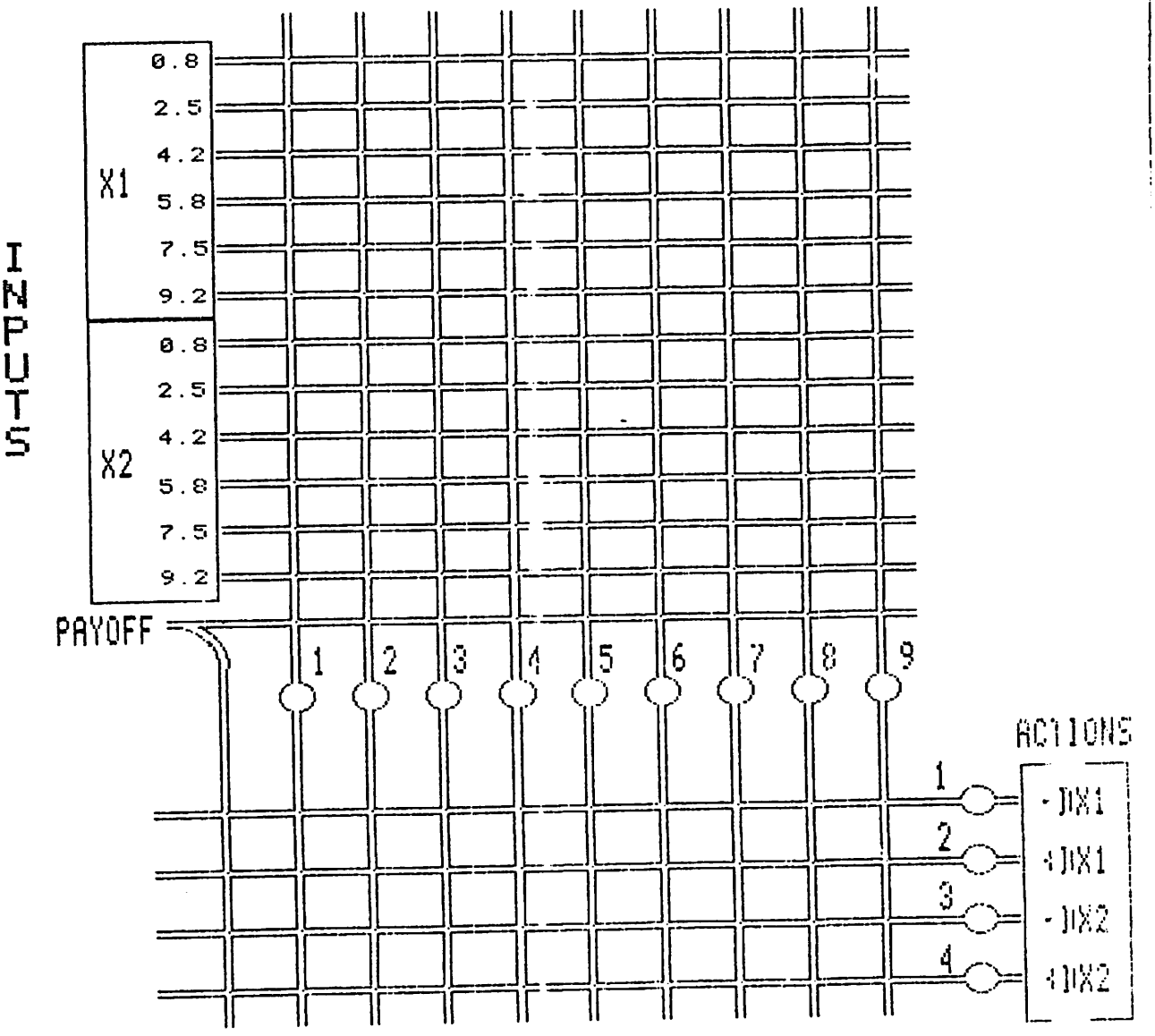

Figure 7: Layers 1 and 2 Combined.

It is important to study the effects of basing the updating of the weights in both networks on the same reinforcement signal. Let us first note that the cause of an incorrect action, indicated by a decrease in payoff, could be twofold. One cause could be an incorrect mapping in layer 2 from the representation (produced by layer 1) to the action vector. A second cause could be a poor representation of the input vector provided by layer 1. A poor representation is one which divides the input space into regions within which more than one best action vector exists. This motivates the question: when a low reinforcement is received, should layer 1 or layer 2 be updated? The reinforcement does not indicate the answer. One possible way of proceeding is to update both. This leads to an interpretation of the learning procedure as a competition between two processes: 1) altering the action-producing map of layer 2, and 2) altering the representation produced by layer 1.

The proper balance of this competition must be found. If layer 2 learns faster than layer 1, then the action-producing map in layer 2 will change rapidly as the state of the environment moves to states requiring different actions. In this case, layer 1 will tend to form representations that contain regions covering large areas of the input space. For example, when a bad action is made, the action-producing map of layer 2 will quickly adapt to produce a good action, allowing an insufficient number of time steps for layer 1 to split the current representation into one containing a region with which layer 2 can associate the good action. If layer 1 learns faster than layer 2, then the

representation formed by layer 1 will change rapidly in relation to the change made in the action-producing layer 2 map. If a bad action is made, rather than altering the action-producing map, the representation is split into one containing many small regions. This splitting will proceed to the limit, i.e., all of the elements in layer 1 will be employed to form regions in the first few areas of the input space that have been visited. To illustrate this it would be necessary to vary the number of time steps between the activations of the learning process in both layers. For example, by allowing layer 1 to learn at every step and layer 2 to learn only every fifth step, layer 1 might be forced to split the representation to an unnecessary degree. This was not investigated in the experiments to be described here, in which both layers learned at every step. A set of learning rule parameters was found that produced the desired cooperative effect between the two layers.

7.0 DEMONSTRATION OF THE SYSTEM'S BEHAVIOR

Before descibing the behavior of the system, we refer to Figure 8 to describe the variables $q_1,\ldots,q_{12}$. Each three-dimensional graph has the axes labeled as shown at the bottom of the figure: the base plane is actually the two-dimensional input space, defined by x_1 and x_2, and the height above the plane represents the value of q_i. The first six graphs show the Gaussian curves $q_1,..,q_6$ that are used to encode the value of x_1. The last three graphs show just three of the six Gaussian curves q_7, q_9, and q_{12} that encode the values of x_2.

The behavior of the system is now demonstrated with a simple example. The goal of the system is the situation $(x_1,x_2) = (5,5)$, meaning that from every other situation the best action is the one that takes the system closer to the situation (5,5). The payoff function z is defined as shown in Figure 9. Note that this figure contains the same base plane as in Figure 8, but the height above the plane now depicts z. The payoff is at a maximum, which is 1.0, at situation (5,5).

For the first part of this demonstration, the situation was held constant at (3,3). This was done to emphasize the effects of learning in one situation; normally the situation is allowed to change at each time step as a result of the actions taken by the system. Figure 10 provides a useful view of the system's state after 100 steps have elapsed. Arrows are shown at representative points of the (x_1,x_2) input space. Each arrow represents the system's expected action, assuming that the system

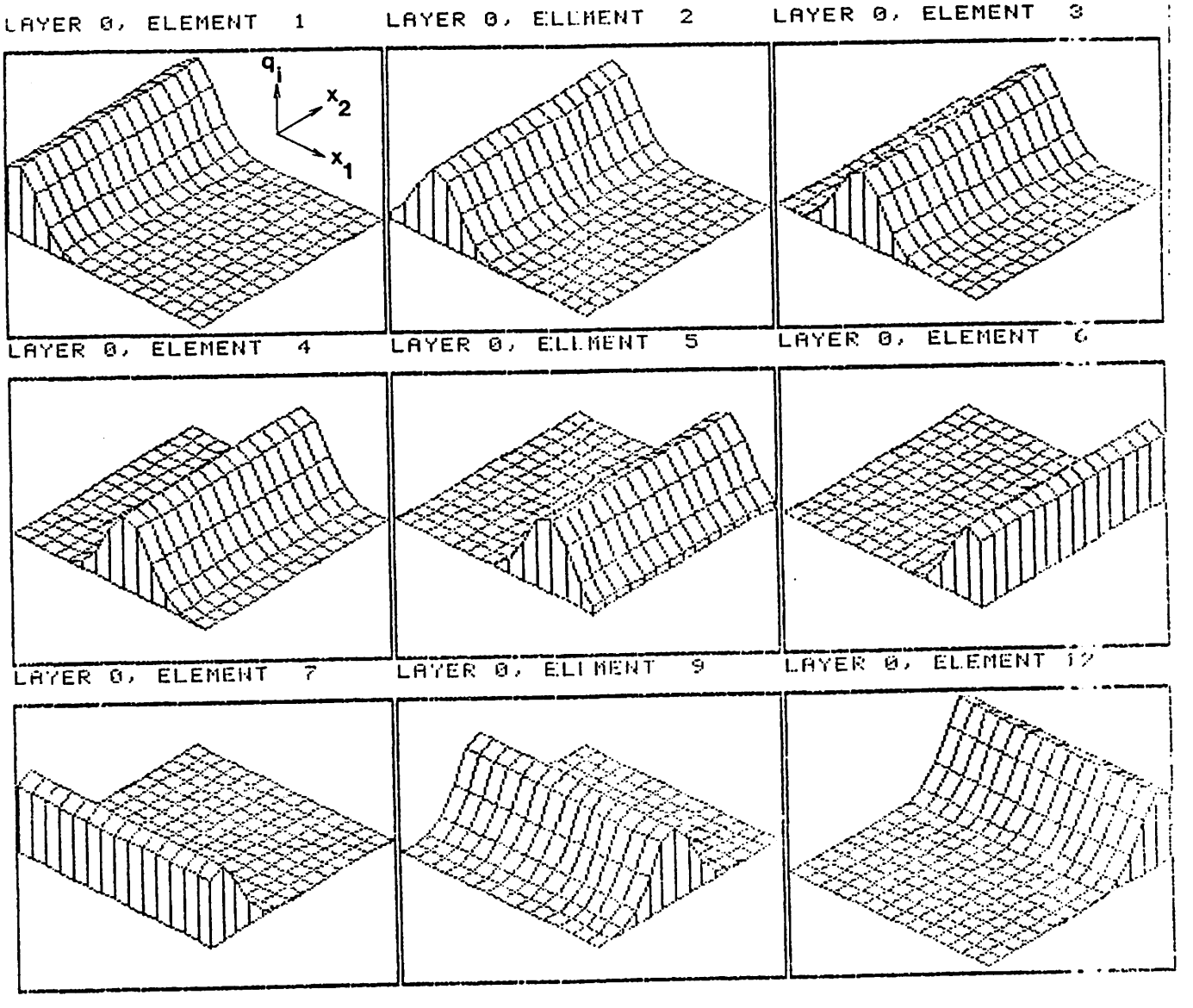

Figure 8: How the Current Situation is Presented to Layer 1.

Each dimension of the current situation vector (x_1, x_2) is quantized into six intervals, and 12 variables are defined such that they are maximal at the center of unique overlapping domains. The top six graphs show the variables used to represent x_1 and the bottom three graphs show three of the six variables representing x_2.

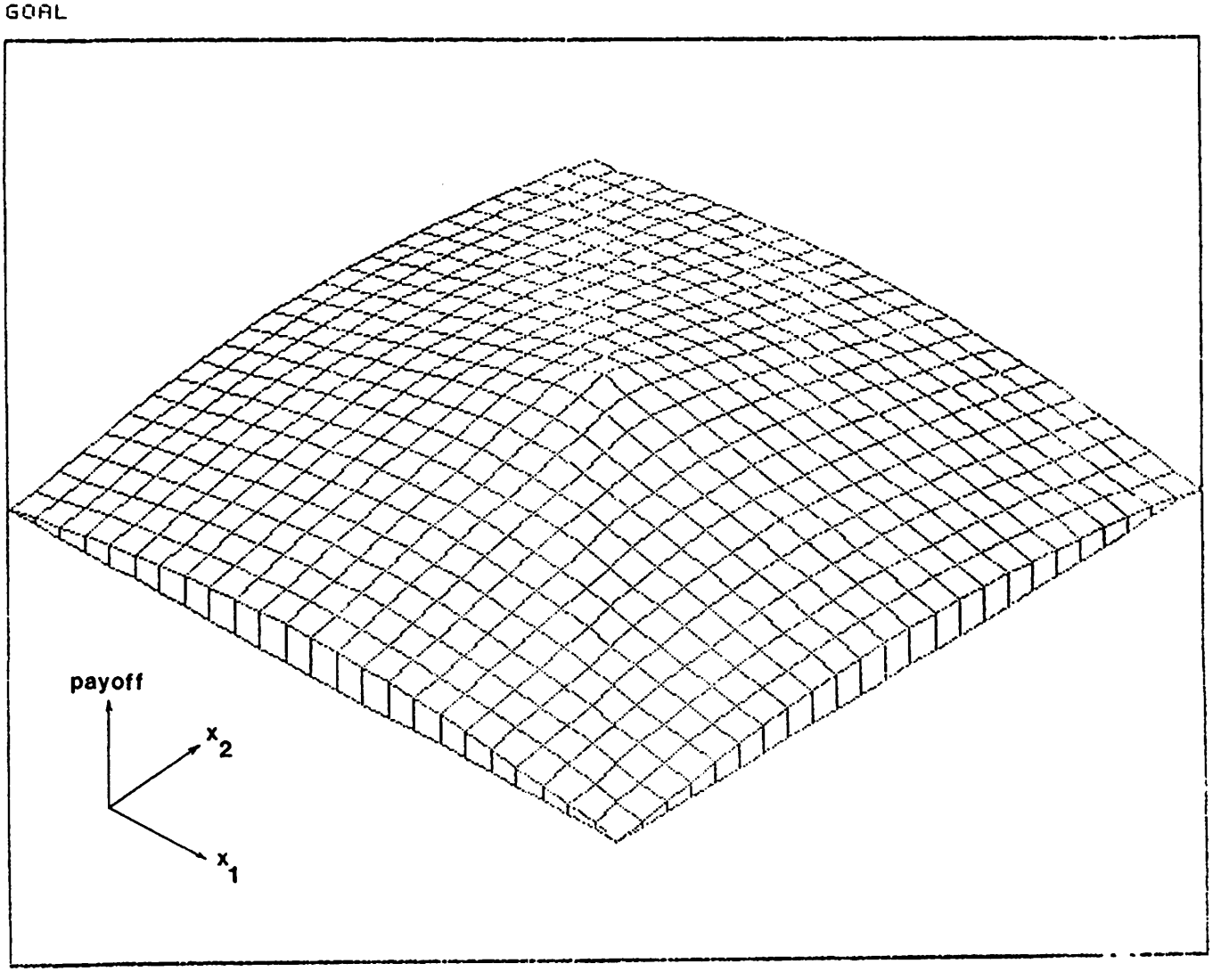

Figure 9: Demonstration: The Payoff Function.

This payoff function defines situation (5,5) as the most desirable situation. The task of the reinforcement learning system is to learn to generate for any given situation the action that results in the next situation being as close to (5,5) as the set of possible actions permits.

is presented with the input vector corresponding to that point and that the system uses its current weight values for determining the action. We call this type of display a "vector field display". The display is generated by first removing the noise from the network's elements and disabling the learning mechanism. The input space is then sampled at evenly spaced points (situations) and the system is simulated for one step starting at each point. The weighted summations that are produced determine the orientation of the arrows. Figure 10 shows that after 100 time steps the system learned to take action $(+dx_1,+dx_2)$ at situation (3,3), and that the effects of learning were generalized across much of the space.

Figure 11 shows the outputs of the nine layer 1 elements, indicating the effect of the system's experience on layer 1. Seven of the elements compute outputs that are very close to zero (their initial outputs), but the outputs of elements 3 and 8 have obviously been affected. These elements produce their greatest values for input situation (3,3). By comparing this figure with Figure 10, one can see that the generalization of the learned action is due to the shape of the "receptive fields" of elements 3 and 8 in layer 1. By the "receptive field" of an element we mean that region of the input space for which the element produces an output greater than zero.

The formation of the elements' receptive fields can be understood by studying the development of the weights in each layer. The values of the weights after the 100 steps are shown

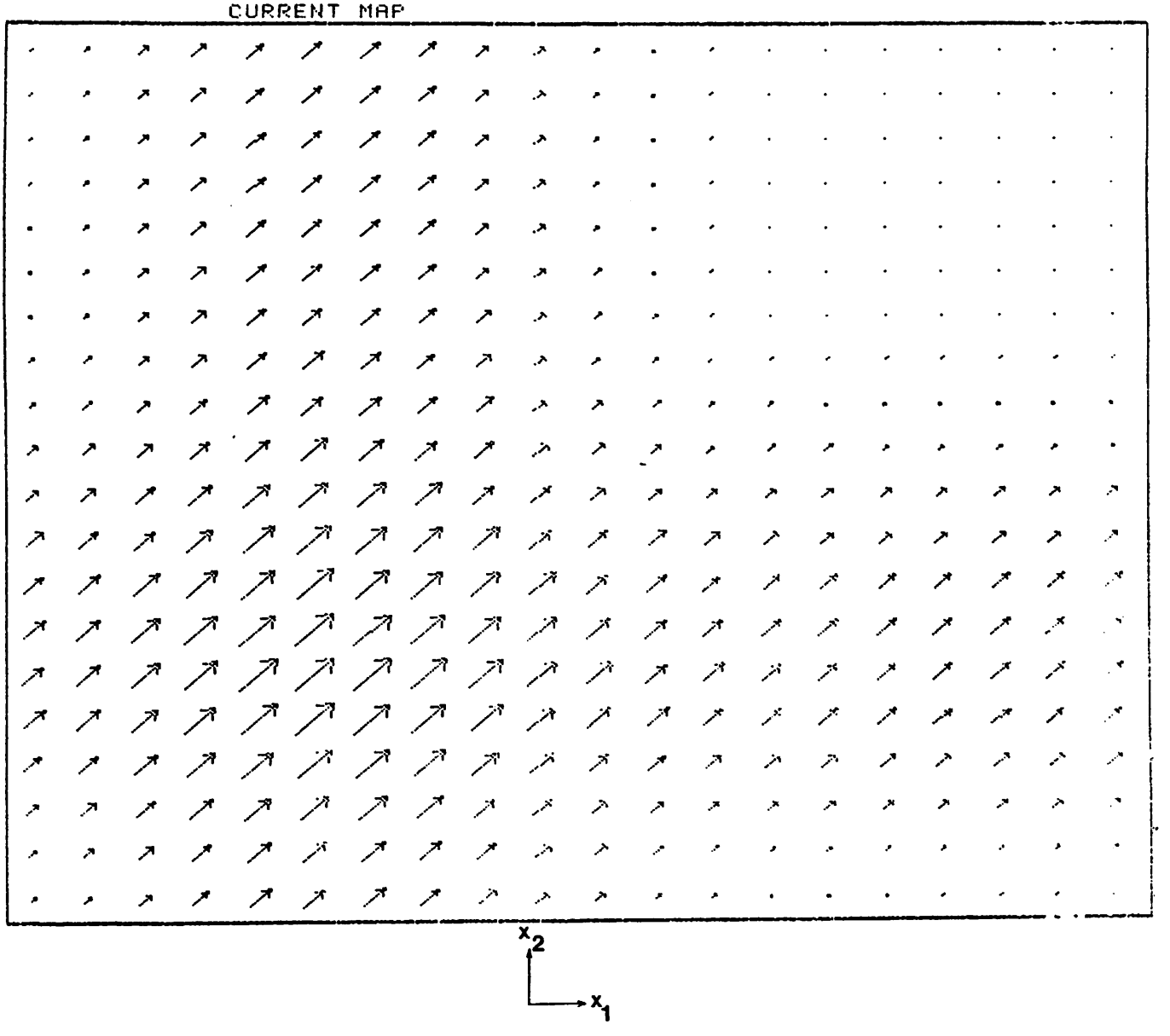

Figure 10: Demonstration: Vector Field Display of Actions After 100 Steps in Situation (3,3).

The situation was held constant at (3,3), and the system was allowed to learn for 100 steps. The results are presented by displaying for each sampled situation the expected situation change, and thus the expected action, that the system will generate if it continues to use the weights present by the 100th step.

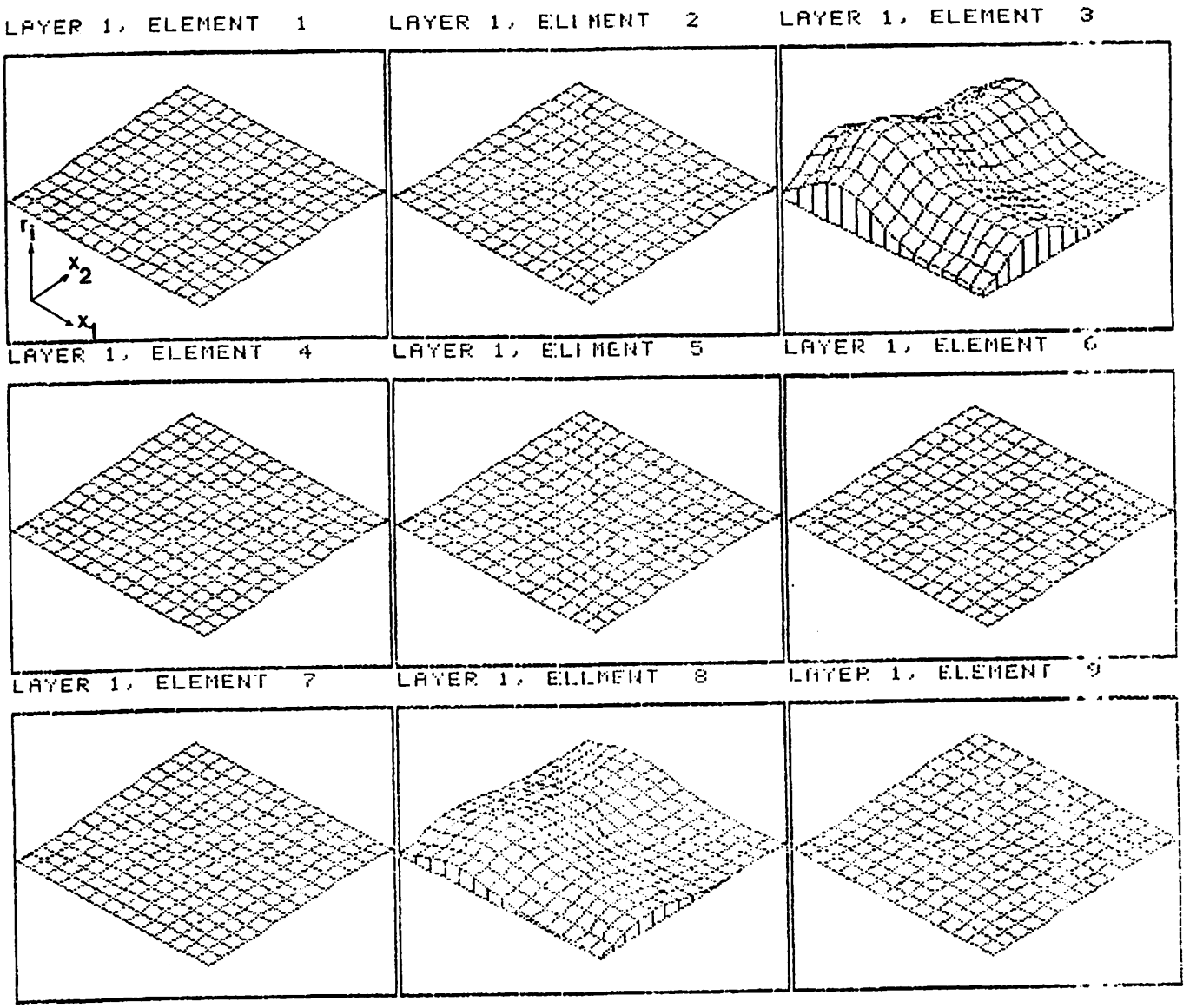

Figure 11: Demonstration: Functions Computed by Layer 1 Elements After 100 Steps.

The elements of the adaptive network learn by adjusting their weights and thereby altering the functions that they compute. The resulting functions computed by each of the nine elements of layer 1 are shown here.

in Figure 12. All weights, other than those associated with elements 3 and 8 of layer 1, have very small magnitudes. Large positive weights have formed between element 3 and the q_i's that have large values for situation (3,3), and similar weights have formed on element 8. It is these weights that determine the shape of the elements' receptive fields. The weights between elements 3 and 8 and the action-producing elements of layer 2 positively associate with each of the elements 3 and 8 the actions $+dx_1$ and $+dx_2$.

In addition to viewing the elements as becoming tuned to certain input situations, one can also view the elements as becoming tuned to certain compound output actions. Whereas the weights on the input side of an element determine its receptive field, the weights on its output side, through which its actions affect other elements, determine its "projective field". Elements 3 and 8 have developed projective fields consisting of the layer 2 elements determining actions $+dx_1$ and $+dx_2$. These elements have thus become very simple examples of "command cells" that trigger a complex of actions. If a system like this were being applied to the control of a two-jointed arm, for example, then activity in these elements might simultaneously increase the angular displacements of both joints (a simple "synergy").

It is important to realize why only two layer 1 elements have weights of significant magnitude. This is due to the competition inherent in the learning rule used for layer 1: At each step only the element producing the largest output is

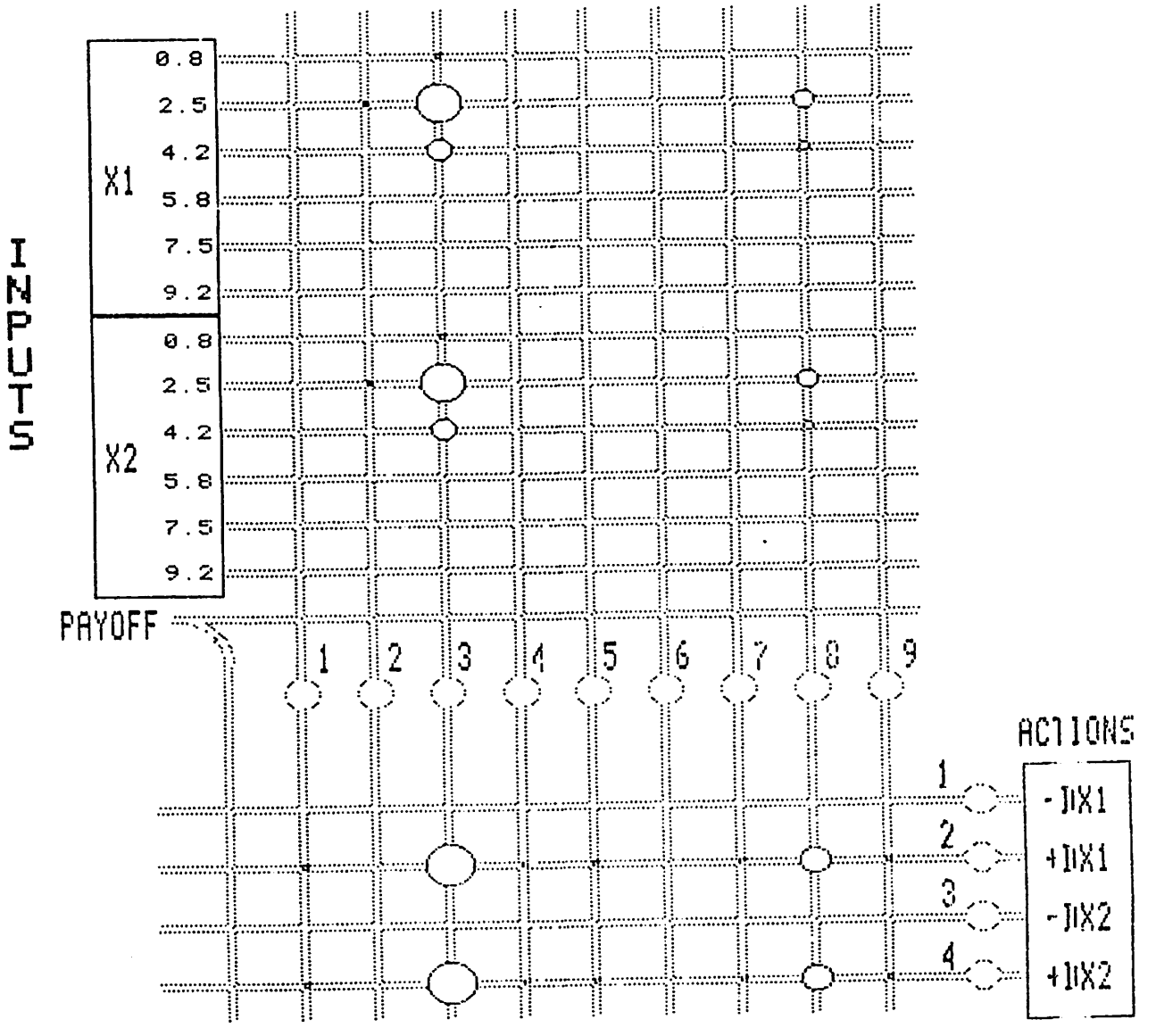

Figure 12: Demonstration: Network Weights After 100 Steps.

The values of both layers' weights after 100 steps of learning are displayed here. The weight values of elements 3 and 8 in layer 1 changed more than those of the other elements due to the competitive nature of the learning rule used in layer 1.

allowed to alter its weights. Since all weights are initially zero, the element producing the largest output is randomly chosen for the initial steps (due to the random noise component in the output function of each element). When a particular layer 1 element is selected for learning, an action produced by layer 2 (also produced randomly for the initial steps) that leads to positive reinforcement will increase the weights associated with that element. As the magnitudes of the weights increase, the selection of layer 1 elements for learning becomes less random until one becomes selected for every step. In this example, the random noise in the elements was such that elements 3 and 8 were initially selected for learning, and during later stages of learning, only element 3 was consistently selected.

Little a priori information was assumed in the design of this system about the structure of the desired situation-to-action mapping. Consequently, the generalization that occurs while experiencing one situation might either facilitate or degrade the learning performance in other situations. In this example, the generalization would certainly facilitate learning in the lower left quadrant of the input space (referring to Figure 10), but would degrade the learning in other quadrants. This degrading effect is seen in the following results.

For the second part of this demonstration, the situation was changed to (3,7) and held constant for another 150 steps. The new vector field display is shown in Figure 13. The best

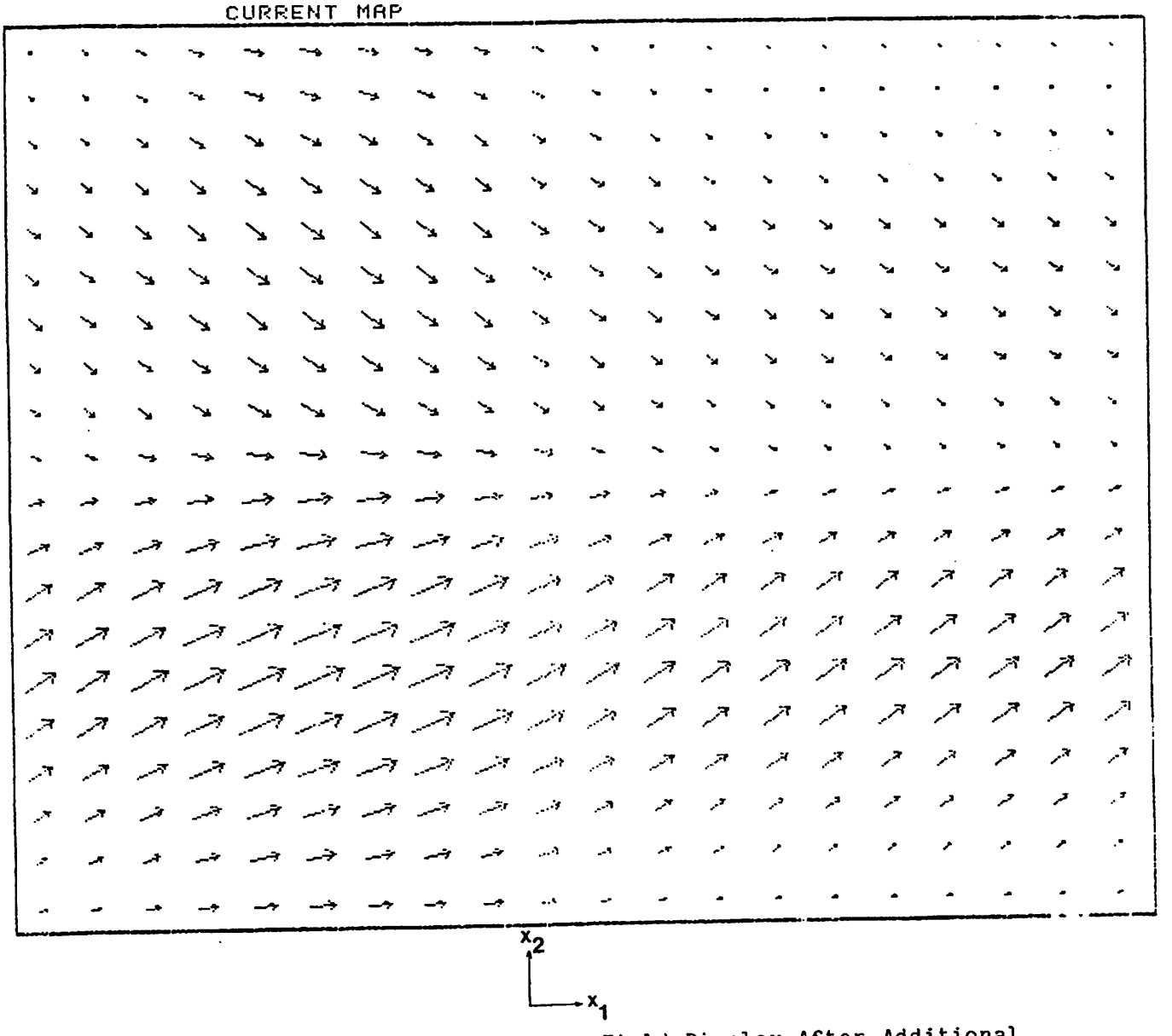

Figure 13: Demonstration: Vector Field Display After Additional 150 Steps in Situation (3,7).

After moving the current situation to (3,7) and again holding it constant, another 150 steps of learning were run. This figure shows that the appropriate action was learned at (3,7) while the action previously learned at (3,3) remains, though with somewhat decreased probability.

action has been learned in situation (3,7) and has been generalized over large areas, overlapping some of the regions in which the action learned earlier was generalized. Figure 14 shows the resulting receptive fields of the layer 1 elements. Now the three elements 3, 4 and 8 possess significant receptive fields. The most notable change in the receptive fields of elements 3 and 8 are the suppression of the output in the upper left quadrant of the base plane (input space), with the maximum suppression near situation (3,7). The receptive field of element 4 has formed in a manner similar to the original formation of the fields of elements 3 and 8. Element 4 produces its greatest output at situation (3,7), while elements 3 and 8 still produce their largest values at situation (3,3).

These new receptive fields are consequences of the weight values shown in Figure 15. These weights formed as follows. When the system was placed in situation (3,7) after learning in situation (3,3), the action $(+dx_1,+dx_2)$ was produced as a result of generalization. This action resulted in a negative reinforcement, causing the weights associated with elements 3 and 8 and the new situation to decrease. Some of these weights became negative (dark disks), thereby suppressing the output of elements 3 and 8 in situation (3,7). As its output approached zero, the other layer 1 elements were again able to compete with elements 3 and 8 for the largest output. Eventually, element 4 began to dominate the layer 1 elements and formed the positive weights shown. Element 4 is associated with the $(+dx_1,-dx_2)$ action-producing elements of layer 2. The graded changes in the

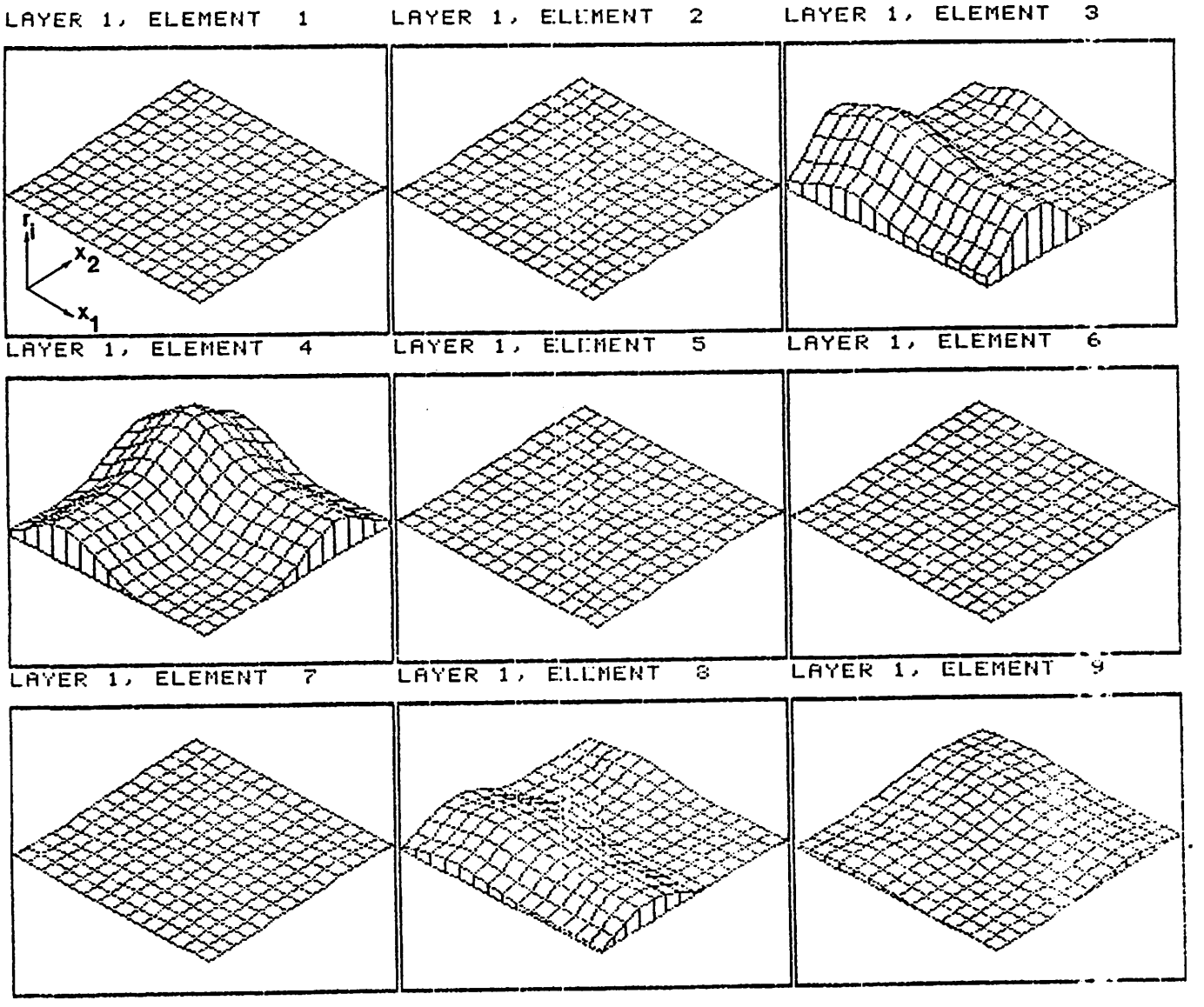

Figure 14: Demonstration: Functions Computed by Layer 1 Elements.

Elements 3, 4, and 8 became tuned to overlapping regions of the input space and therefore produce outputs greater than 0 for unequal but intersecting sets of situations.

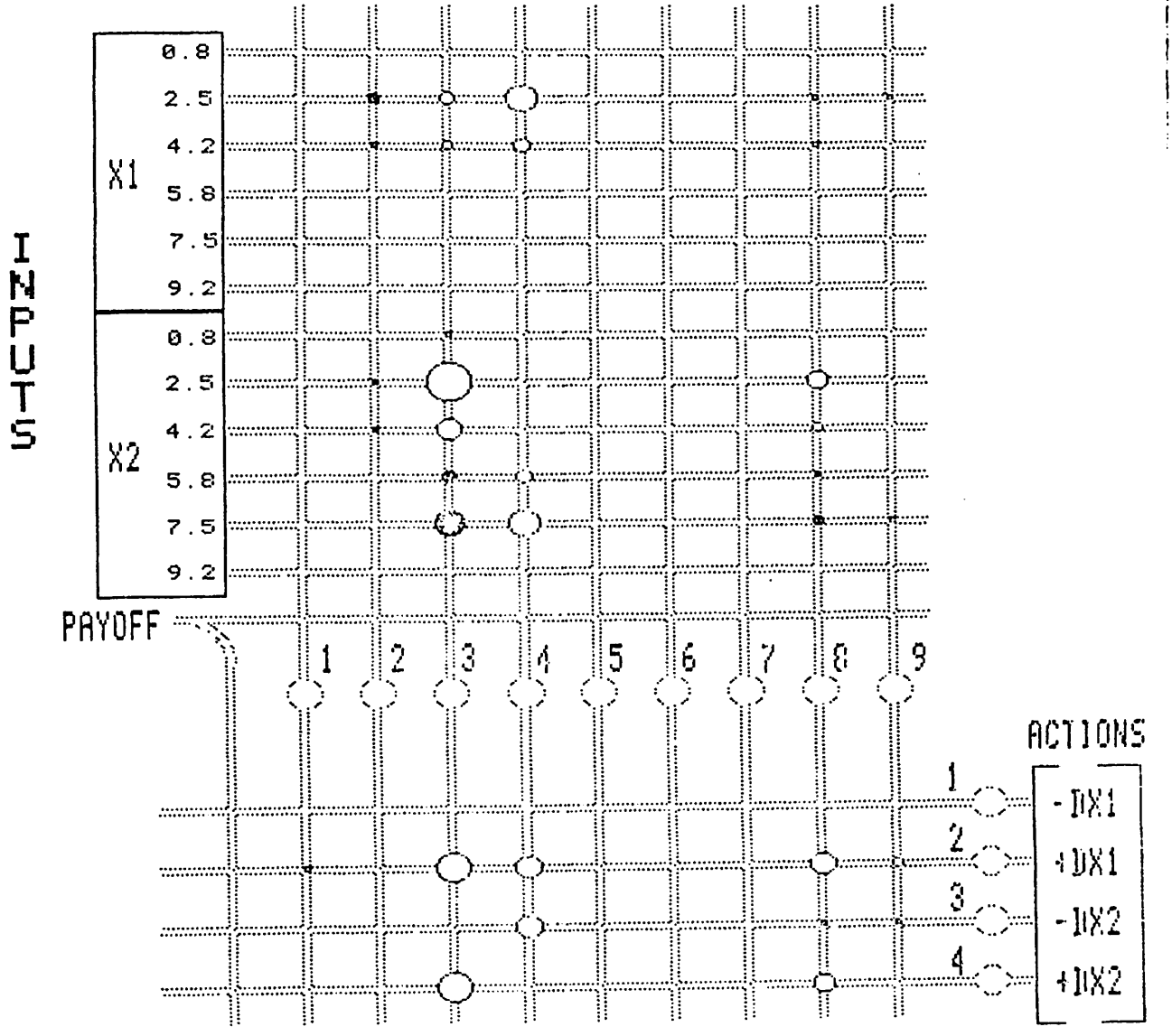

Figure 15: Demonstration: Final Network Weights.

Elements 3 and 8 learned to produce the $(+dx_1,+dx_2)$ action while element 4 learned to produce the $(+dx_1,-dx_2)$ action.

actions across the space in Figure 13 are due to the superposition of the two receptive fields.

8.0 RESULTS

This section presents the results of four experiments involving four simulated control tasks. The same parameter values were used for all experiments, and all weights were initially zero. The experiments primarily differed in the form of the payoff functions employed.

8.1 Experiment 1

The payoff function used in the first experiment was the same as the one used to demonstrate the behavior of the system in Section 7.0 and is shown in Figure 9. The experiment was initiated by setting the situation to (5,1) and starting the execution of the system. The action produced at each step was allowed to change the situation. The system was run for 700 steps in this manner. Figure 16 is a record of every fifth situation change produced, and the asterisk marks the current situation. The system successfully learned what actions to associate with each situation that it experienced. This can also be seen in the vector field display of Figure 17. Note the generalizaton to situations that had not been experienced. The situation-to-action mapping can be refined by experiencing more situations. This was demonstrated by randomly selecting 500 additional situations and running the system for one step in each. The resulting situation-to-action mapping appears in Figure 18.

Figure 19 shows the final receptive fields of the elements in layer 1. Elements 1, 3, 6 and 9 encode the only significant features that formed. This is more noticeable in Figure 20. In this network display, the weights in layer 2 show that only four layer 1 elements contributed to the action-selecting process. From this display, we can determine that the features computed by elements 1, 3, 6 and 9 of layer 1 are respectively associated with the actions $(+dx_1,-dx_2)$, $(-dx_1,+dx_2)$, $(-dx_1,-dx_2)$, and $(+dx_1,+dx_2)$.

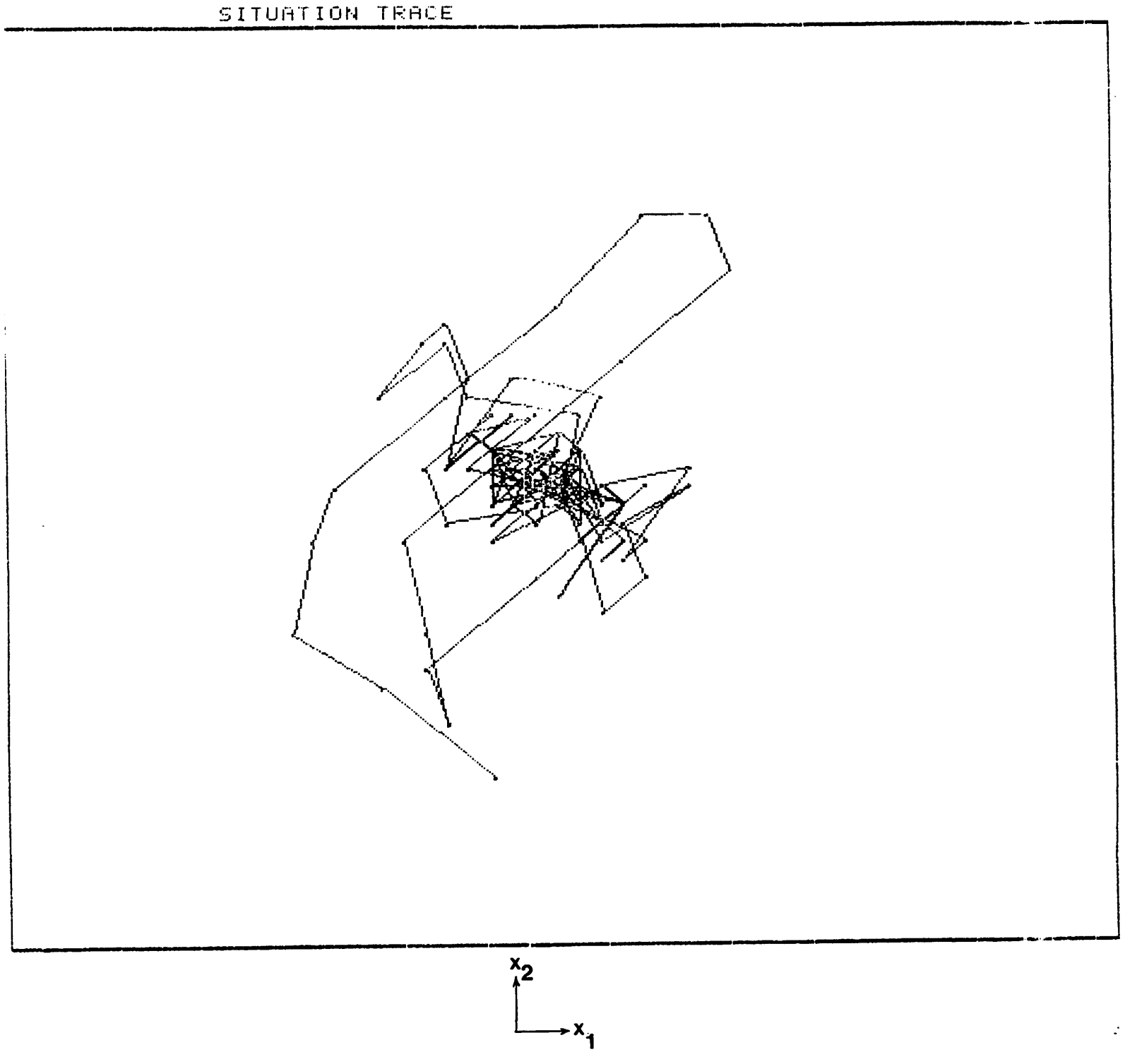

Figure 16: Experiment 1: Trace of the Situation Changes.

The situation was initialized to (5,1) and allowed to be changed by the system's actions. The system quickly learned to produce actions that maintained the situation close to (5,5).

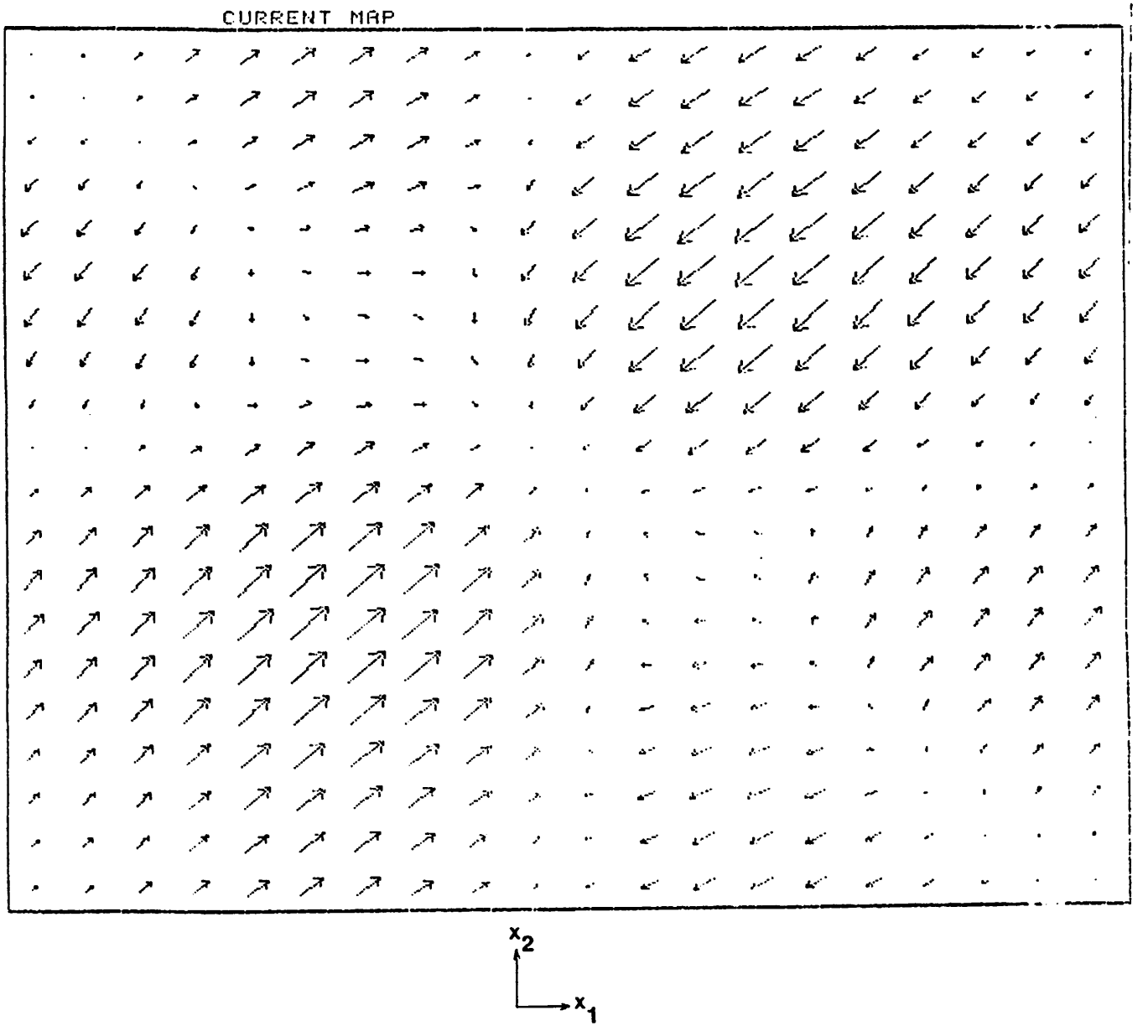

Figure 17: Experiment 1: Vector Field Display After 700 Steps.

After the 700 steps shown in the previous figure, the expected actions were as shown here. In some regions, such as near the corners of the situation space, the learned situation-to-action mapping will not result in good performance. This is due to a lack of experience with the situations in these areas.

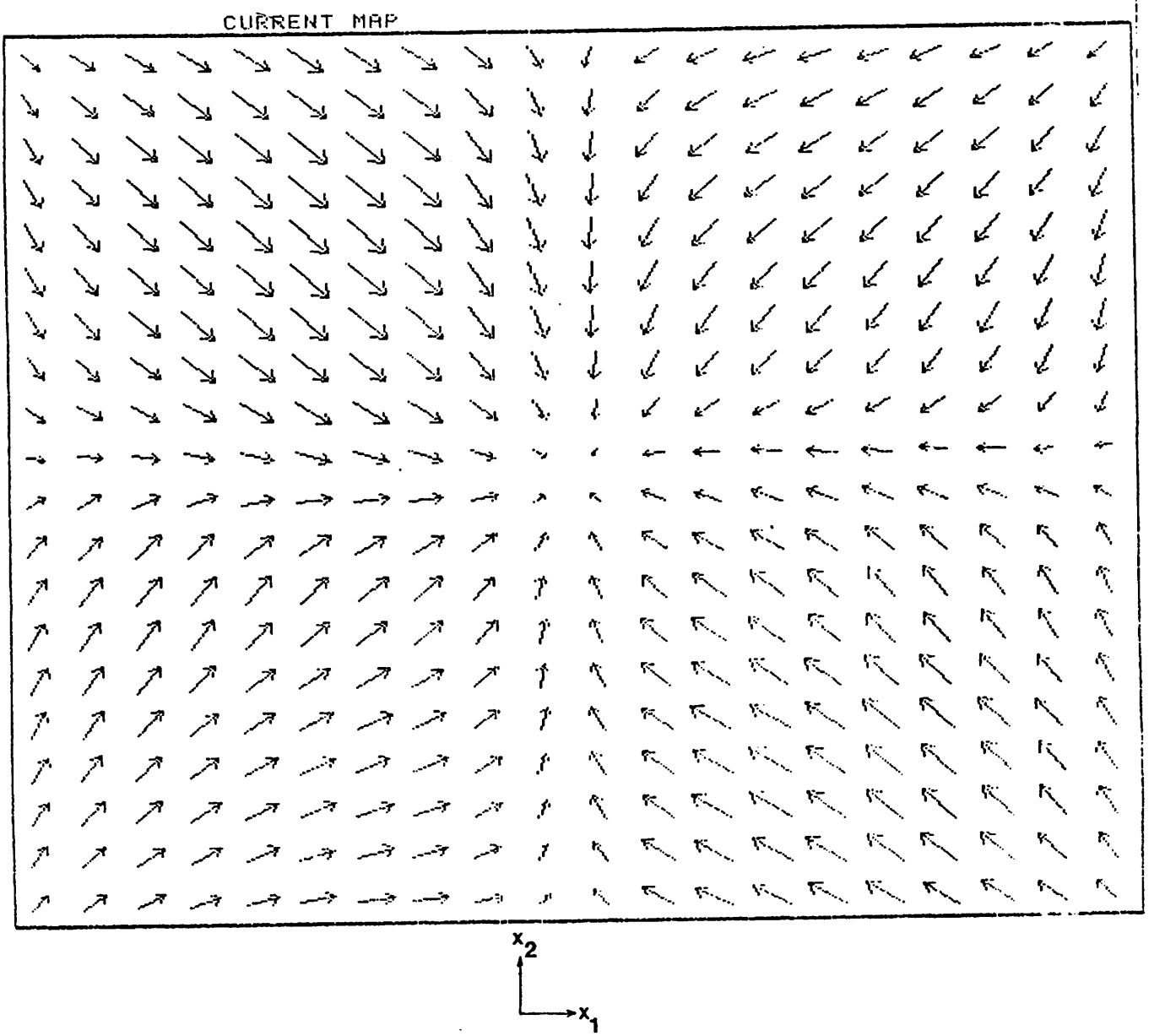

Figure 18: Experiment 1: Vector Field Display After an Additional 500 steps.

An additional 500 steps of learning were performed with random changes in the situation before each step. This resulted in learning experiences throughout much of the space, thus refining the learned situation-to-action map.

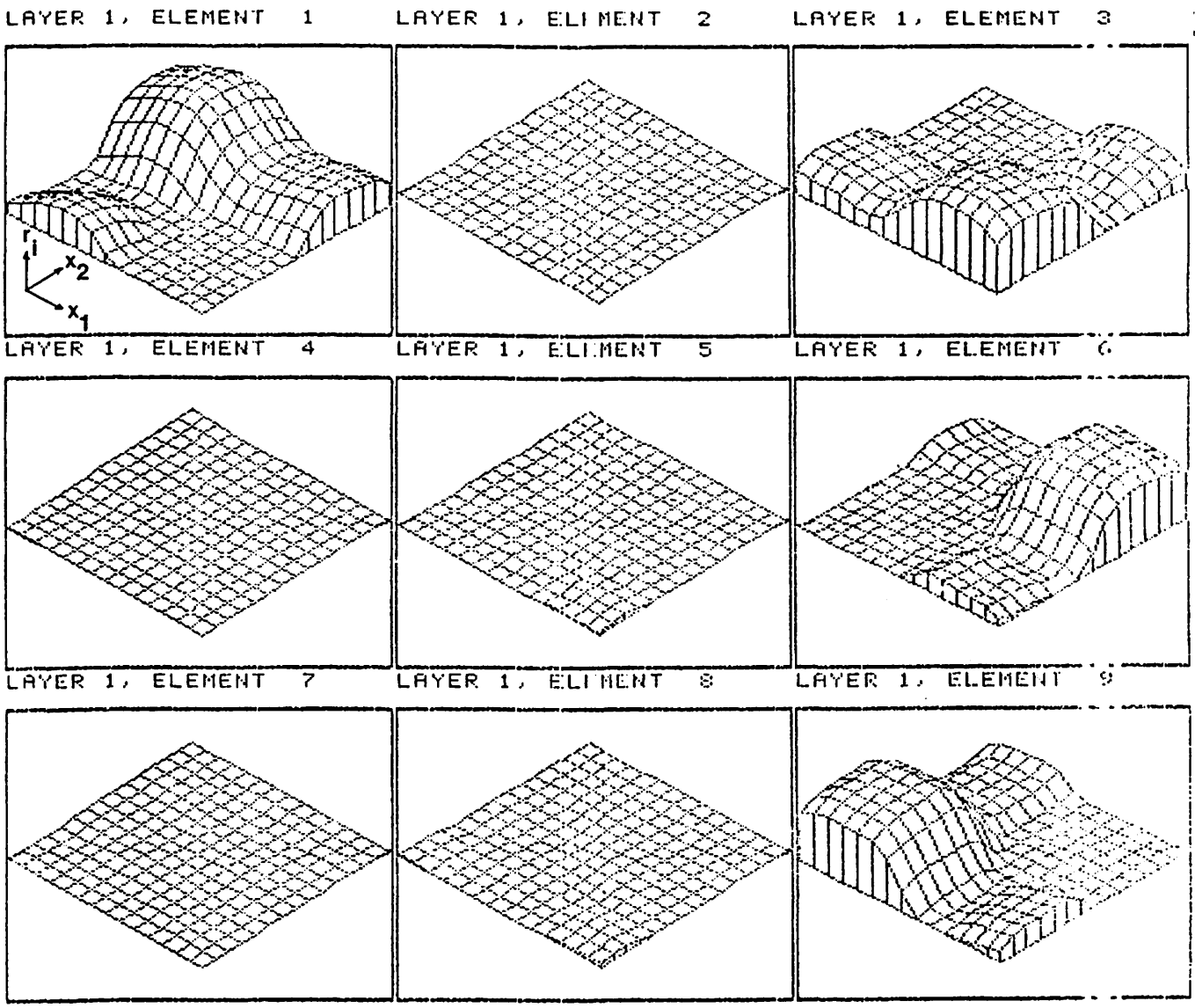

Figure 19: Experiment 1: Functions Computed by Layer 1 Elements.

Four of the layer 1 elements have developed significant receptive fields. They produce their maximum values in unique quadrants of the situation space.

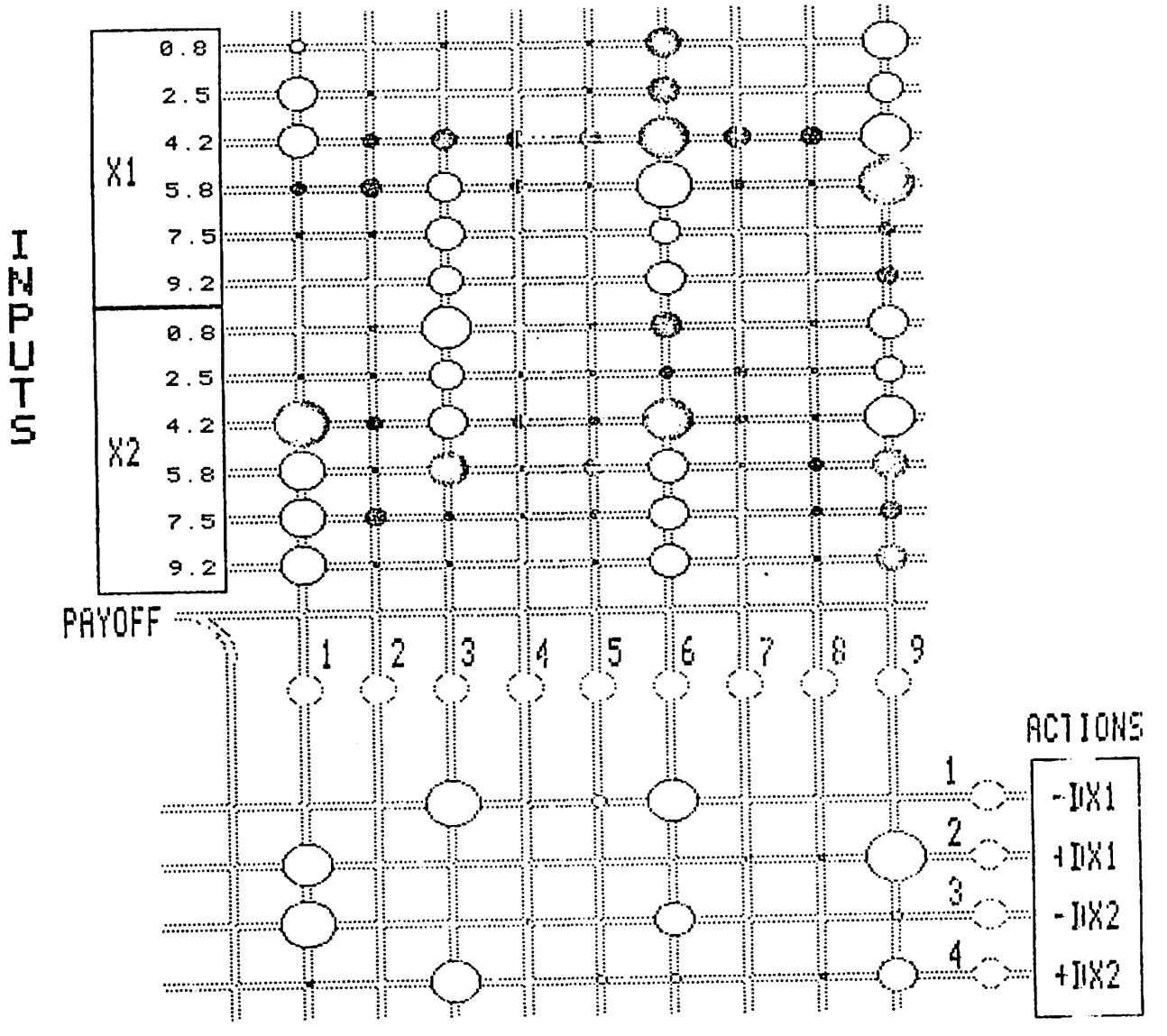

Figure 20: Experiment 1: Network Weights.

The weights on the inputs to layer 1 have adapted to produce the functions shown in the previous figure. The adaptation of the layer 2 weights resulted in the generation of actions $(+dx_1,-dx_2)$, $(-dx_1,+dx_2)$, $(-dx_1,-dx_2)$, and $(+dx_1,+dx_2)$ by elements 1, 3, 6, and 9, respectively, of layer 1.

8.2 Experiment 2

For the second experiment we used the payoff function in Figure 21. In this case there are two situations of equal desirability, (2.5,5) and (7.5,5). Figure 22 shows the actions produced by the first 1000 steps, starting from situation (5,5). The initial actions generated by the network resulted in situation changes that brought the system closer to the goal at (2.5,5). The network learned to generate those actions that produced situations near (2.5,5), thus avoiding the entire right half side of the input space. Figure 23 shows the resulting vector field display. The left half of the input space generates a more accurate mapping than that generated by the right half because the system's experience was limited to the left side. The system was restarted in situation (6,2) and the actions shown in Figure 24 were generated during the next 1000 steps, resulting in the much improved situation-to-action mapping in Figure 25.

Figure 26 shows the receptive fields of the elements in layer 1. Let us study the role that element 1 has assumed. Figure 27 shows that element 1 is associated with the elements of layer 2 causing the action $(-dx_1,-dx_2)$. This action is represented in Figure 23 by arrows pointing to the lower left, and the area of this figure in which these arrows appear corresponds to the receptive field of element 1. The system has learned that element 1 and its associated action can be applied to situations that are to the upper right of either peak. When considering only the feature generated by element 1, situations

that are similarly related to each peak are indistinguishable. This loss of information, or grouping of situations into one feature, is useful in a system of limited resources; situations requiring similar actions need not be uniquely identifiable. The layer 1 elements 4, 6 and 7 are also maximally sensitive to situations that are similarly related to either peak.

GOAL

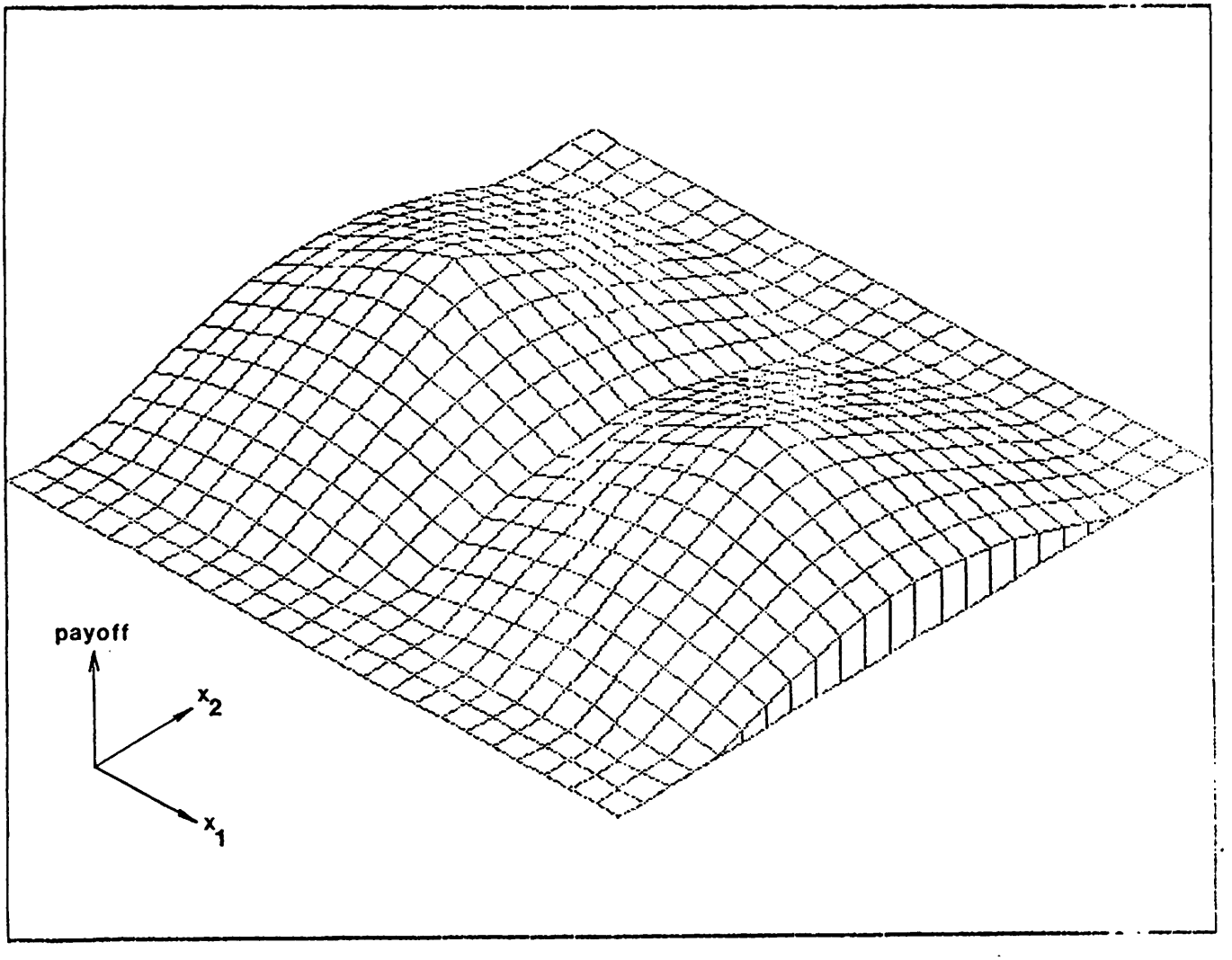

Figure 21: Experiment 2: Payoff Function.

The payoff function for experiment 2 contains two peaks defining two most-desirable situations, (2.5,5) and (7.5,5).

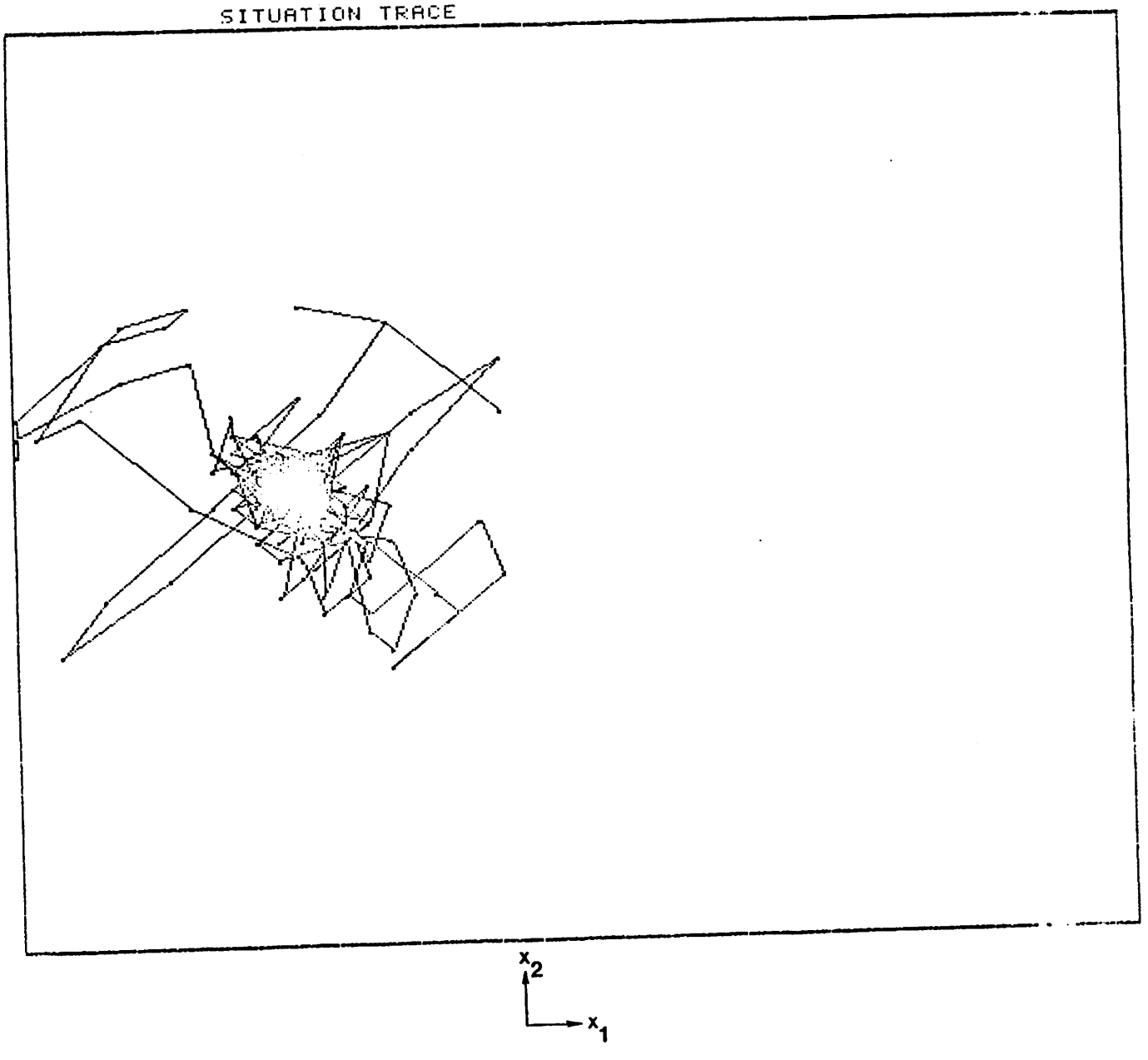

Figure 22: Experiment 2: Trace of the Situation Changes.

The situation was initialized to (5,5) and the system was allowed to run for 1000 steps producing the situation changes shown.

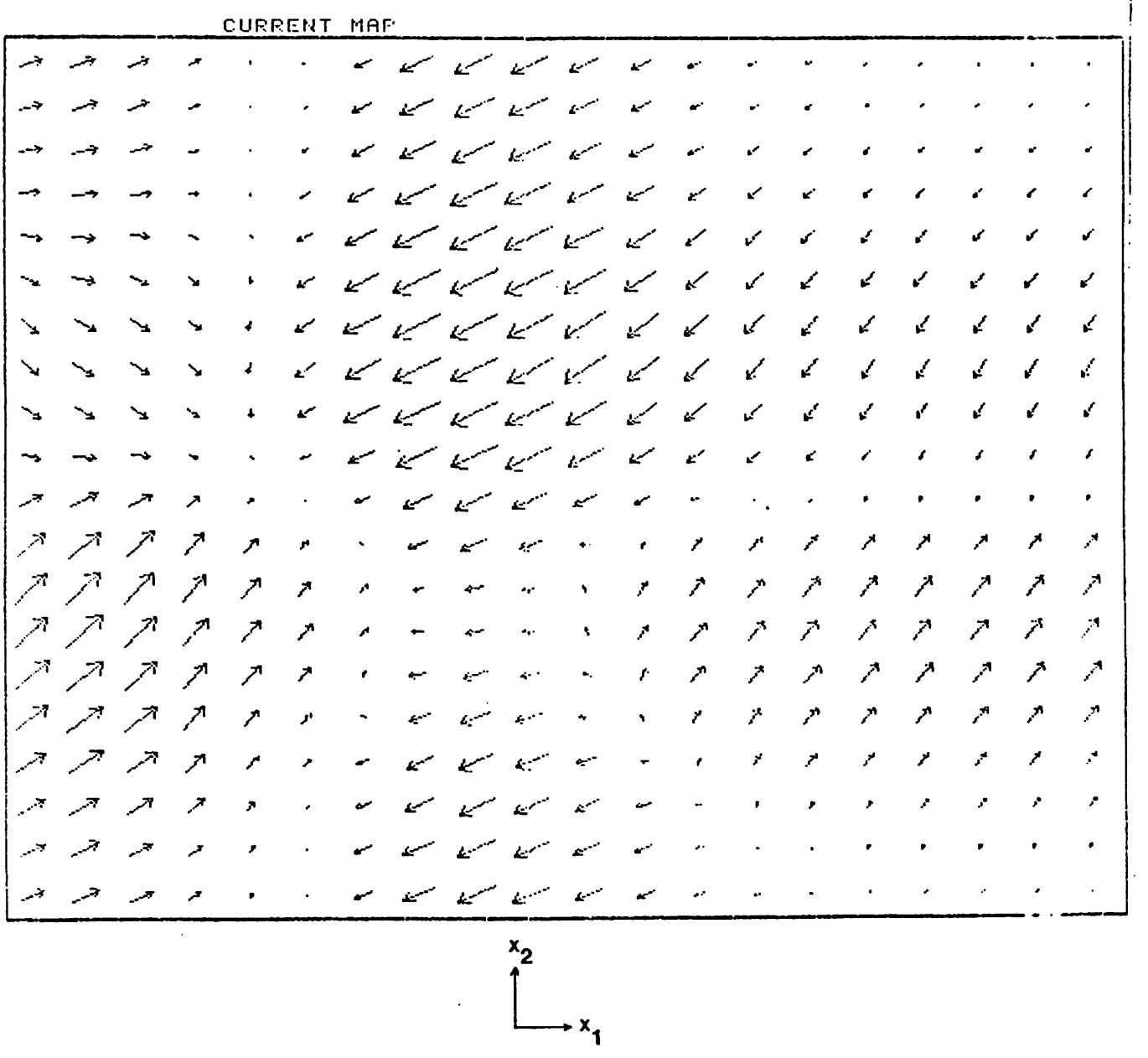

Figure 23: Experiment 2: Vector Field Display.

The expected actions are approximately correct only for situations near (2.5,5). This is due to the lack of experience in areas of the space distant from (2.5,5).

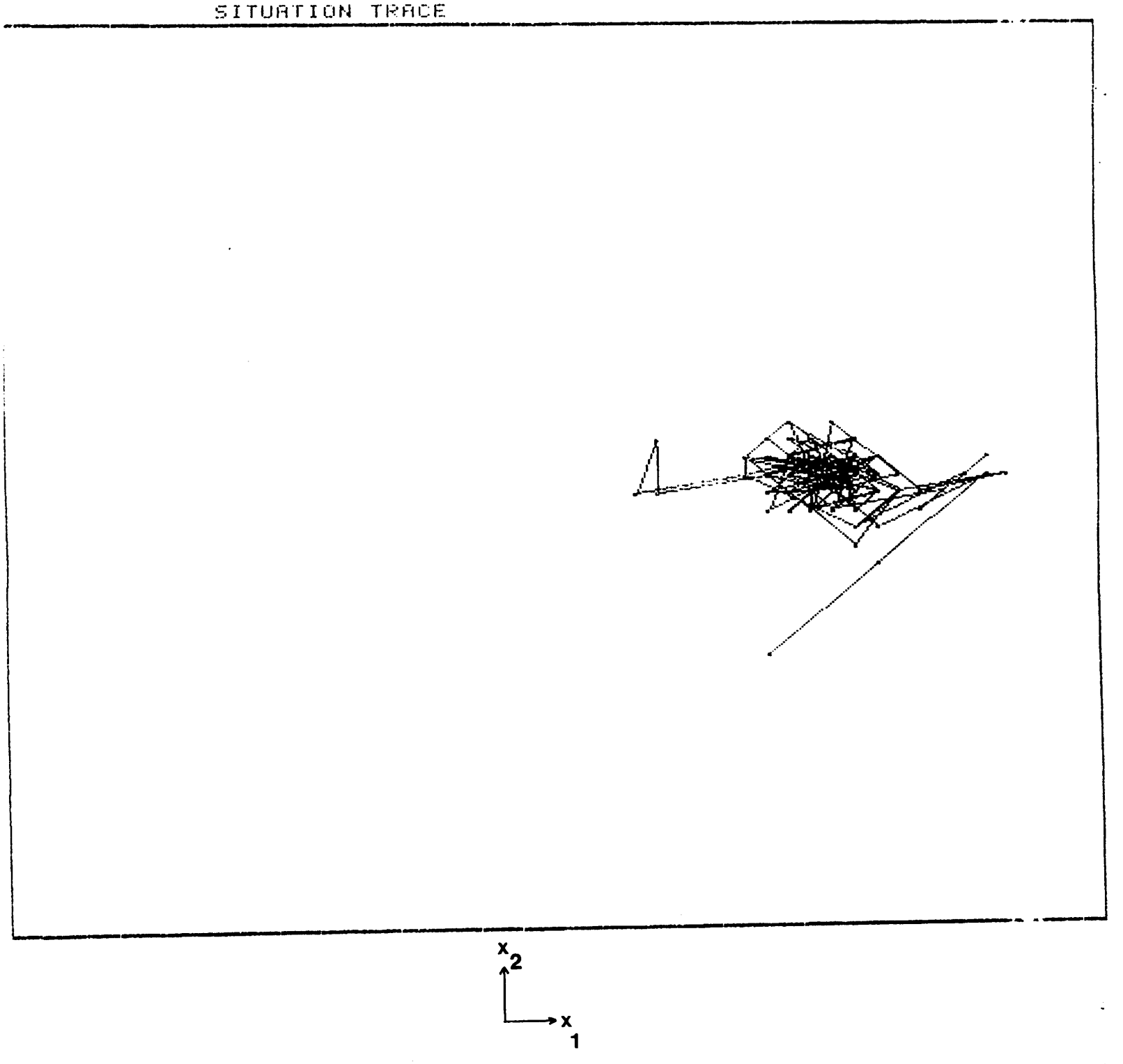

Figure 24: Experiment 2: Additional Situation Changes.

An additional 1000 steps were run starting from situation (6,2) producing the situation changes shown.

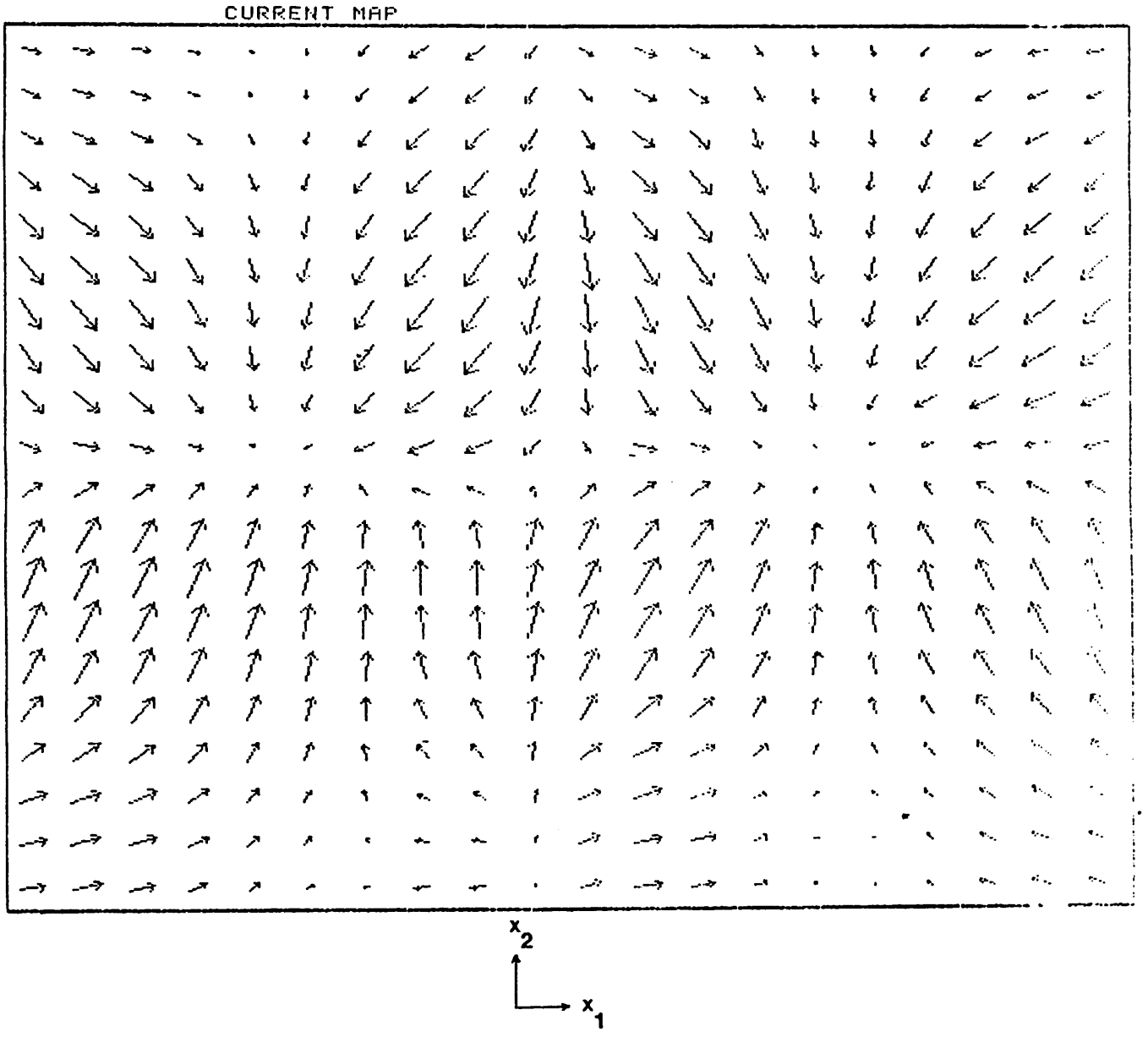

Figure 25: Experiment 2: Final Vector Field Display.

After learning near the right peak of the payoff function, the overall situation-to-action mapping is much improved.

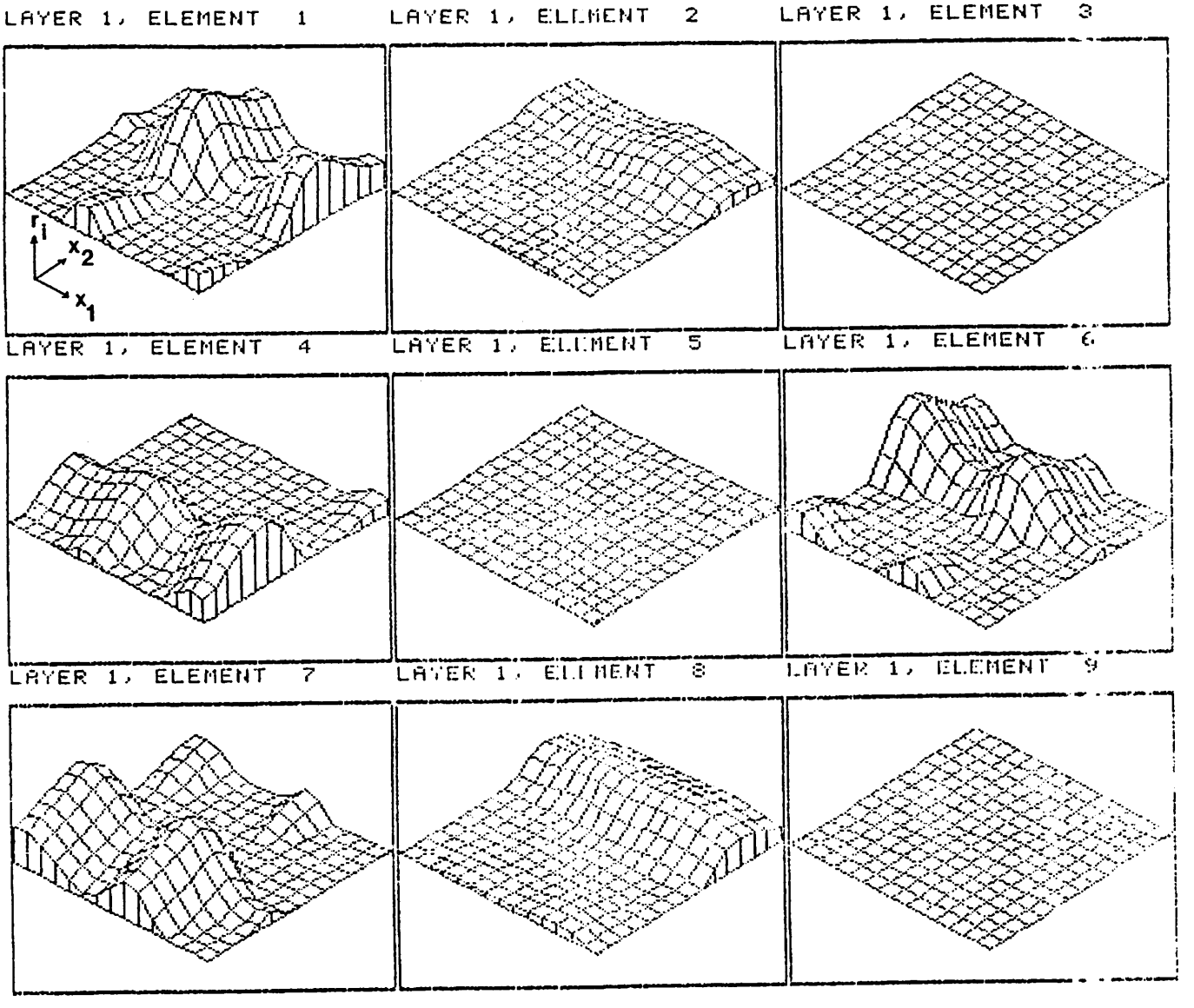

Figure 26: Experiment 2: Functions Computed by Layer 1 Elements.

Four of the layer 1 elements were primarily significant to this experiment. They became tuned to regions to either the upper left or right, or lower left or right of the peaks in the payoff function.

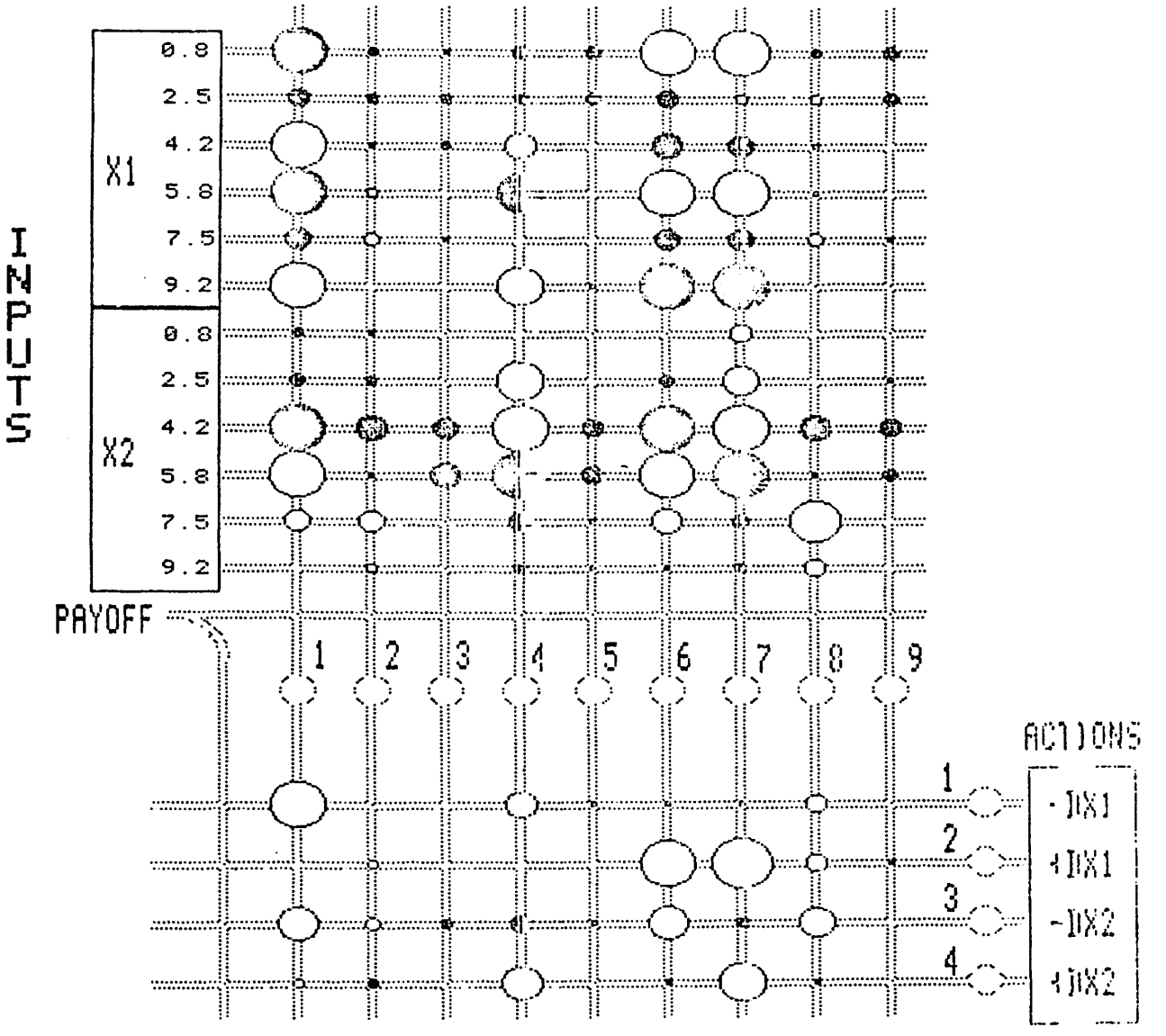

Figure 27: Experiment 2: Network Weights.

Elements 1, 4, 6, and 7 learned to generate actions $(-dx_1,-dx_2)$, $(-dx_1,+dx_2)$, $(+dx_1,-dx_2)$, and $(+dx_1,+dx_2)$, respectively.

8.3 Experiment 3

The rationale for performing experiment 3 was to study how the placement of the payoff function's peaks affected the solution. The payoff function for this experiment contained two peaks of equal height, but varied from the payoff function of experiment 2 in that the positions of the peaks differed not only in the x_1 component as in experiment 2, but also in the x_2 component. This payoff function is shown in Figure 28. One layer 1 element could become tuned to more than one region only if the centers of the regions were approximately on a line that was parallel to one of the dimensions, due to the manner in which the input space was represented by the q_i's, whose values were constant along one of the dimensions x_1 or x_2. For example, in experiment 2, one element became tuned to regions to the upper right of each peak. However, this was not possible in this experiment as the regions to the upper right of each peak were on a diagonal line that was not parallel to a dimension. Therefore, this experiment should result in the recruitment of more layer 1 elements.

We trained the system for 2600 steps during which we interrupted the simulation several times and altered the current situation to give the system experience in different areas of the (x_1,x_2) space. Figure 29 shows the resulting situation-to-action mapping as a vector field display. Figures 30 and 31 respectively show the layer 1 elements' receptive fields and the network's weights. By comparing Figures 30 and 27, we can see

that experiment 3 resulted in a greater utilization of the layer 1 elements than did experiment 2. In experiment 3 at least seven elements make significant contributions to the calculation of an action, whereas only four elements were significant in experiment 2.

To further contrast these experiments, let us analyze the production of the $(-dx_1,+dx_2)$ action in both cases. In experiment 2, element 4 is sensitive to the lower right regions of each peak and is positively connected to the $-dx_1$ and the $+dx_2$ layer 2 elements. However, in experiment 3 two elements, numbers 5 and 7, were used to generate the $(-dx_1,+dx_2)$ action, with element 5 sensitive to the upper left peak and element 7 to the lower right peak. These two experiments demonstrate that the behavior during learning is such that an "approximately minimal" number of layer 1 elements are recruited to perform the task. (The actual number of layer 1 elements used for a given task is dependent on the parameters of the system and is not guaranteed to be minimal.)

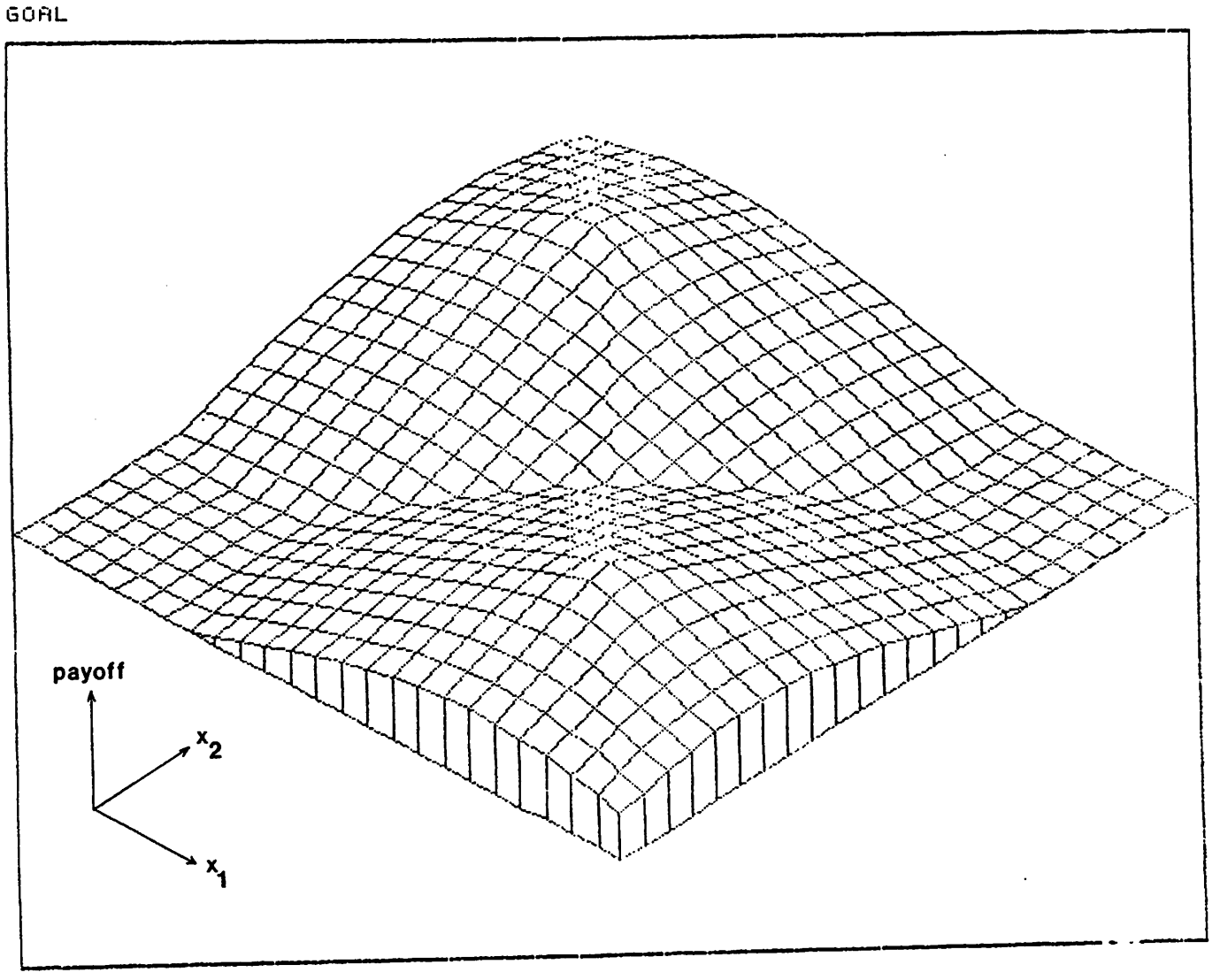

Figure 28: Experiment 3: Payoff Function.

The payoff function used in experiment 3 contains two peaks at (2.5,7.5) and (7.5,2.5).

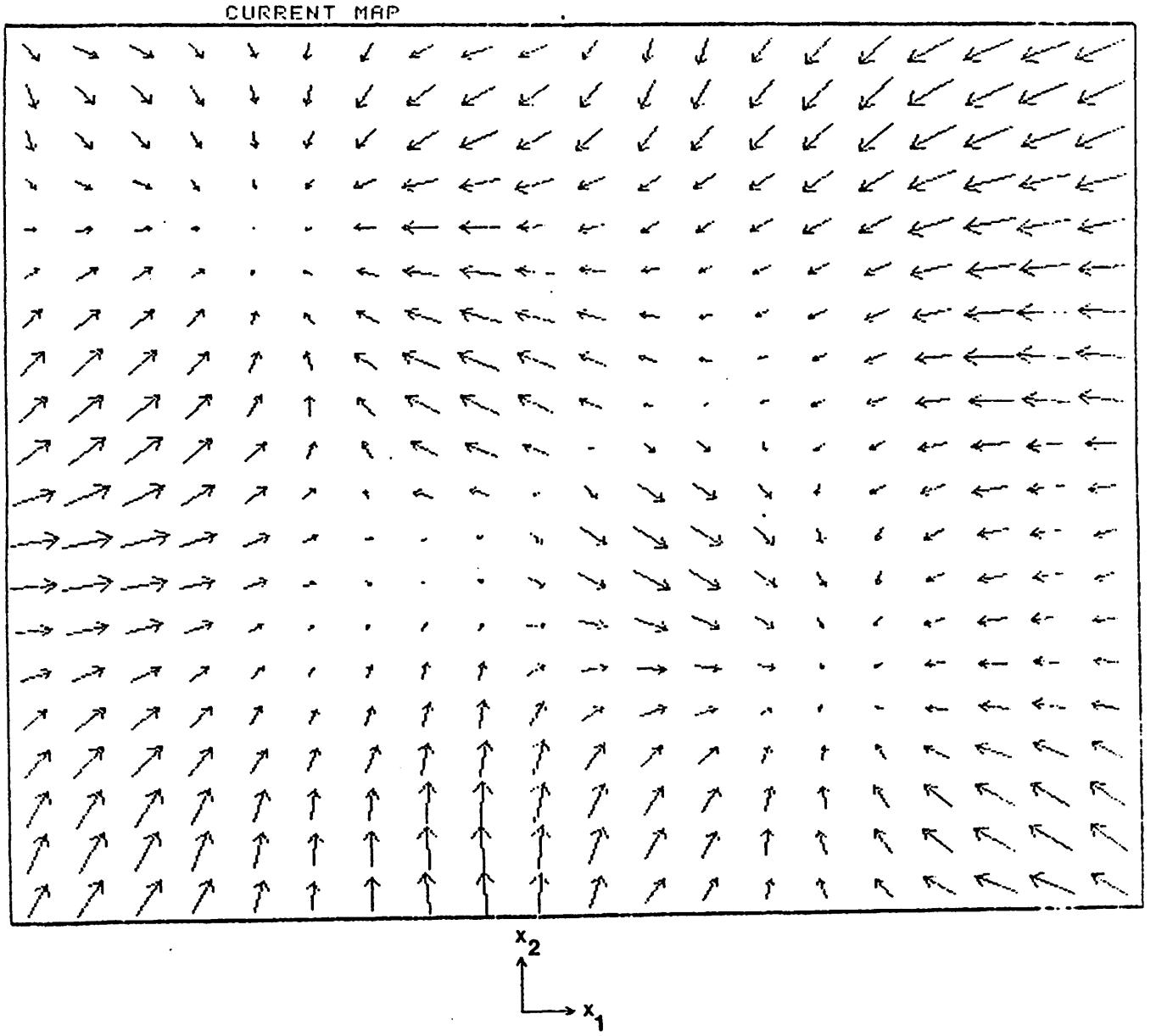

Figure 29: Experiment 3: Vector Field Display.

The system was trained for 2600 steps during which the situation was changed and the system restarted several times.

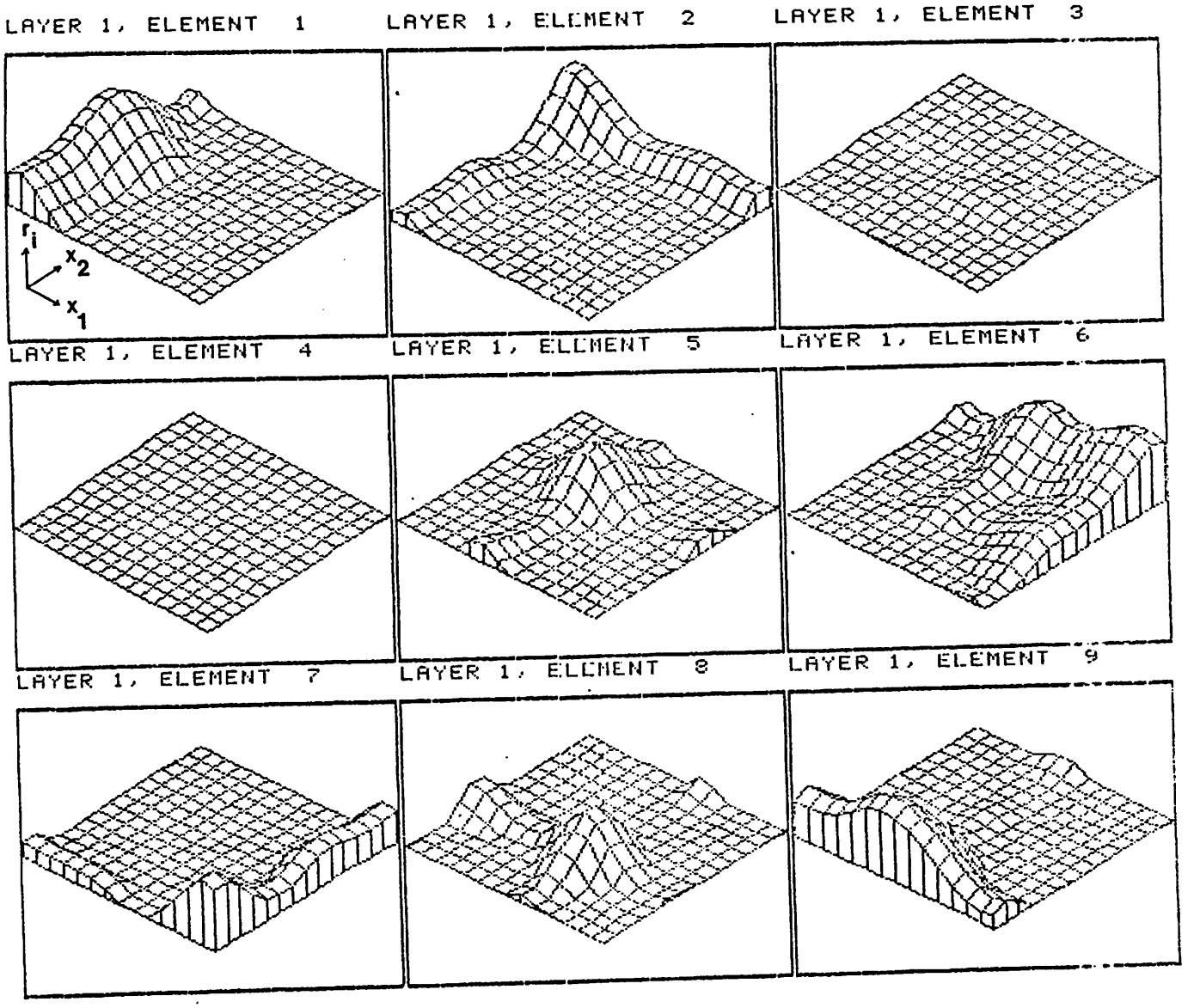

Figure 30: Experiment 3: Functions Computed by Layer 1 Elements.

In this experiment approximately seven layer 1 elements contributed significantly to the results.

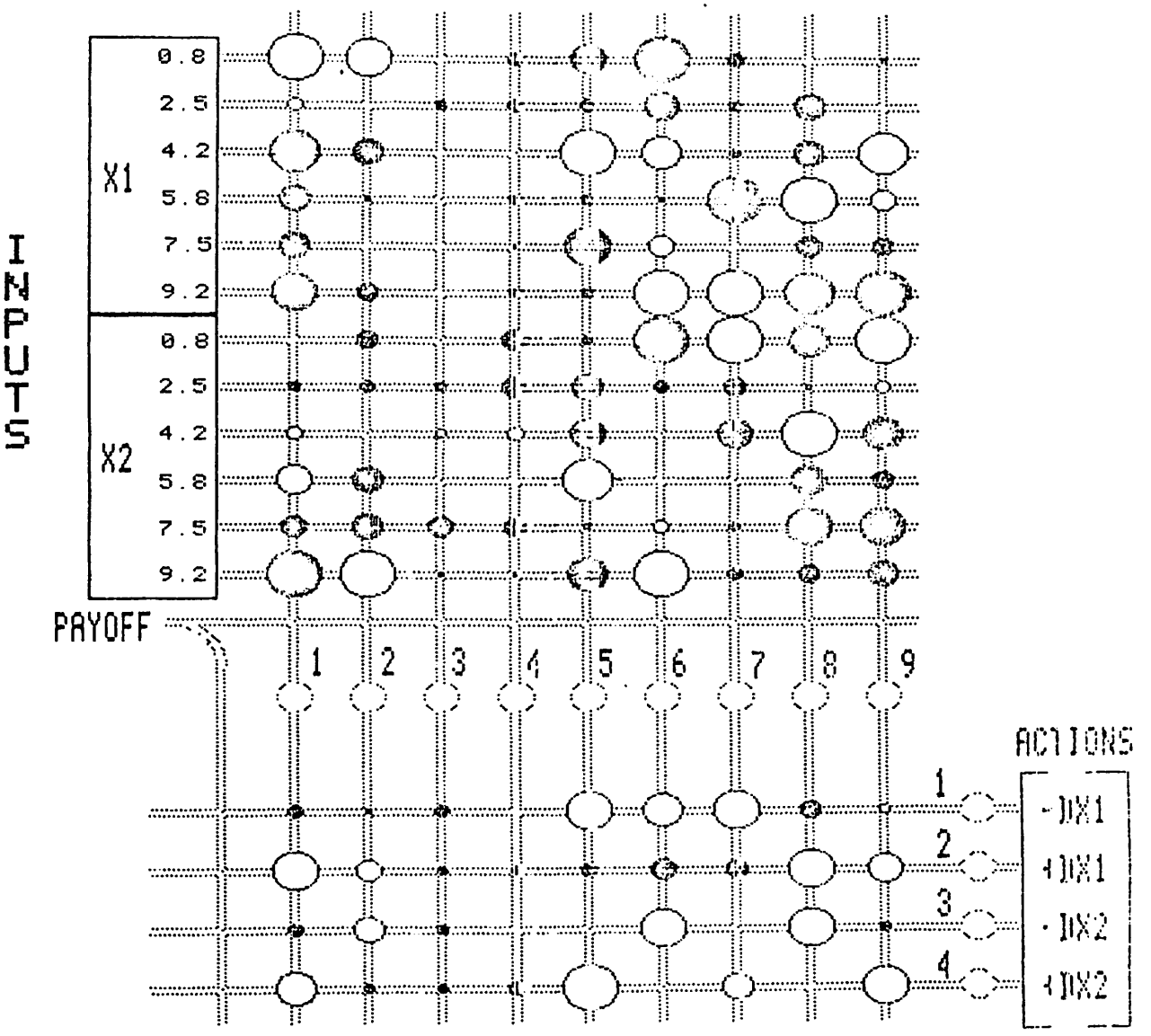

Figure 31: Experiment 3: Network Weights.

A larger proportion of layer 1 elements were tuned in this experiment in relation to experiment 2. This is due mainly to the fact that the placement of the two peaks of the payoff function were diagonally related rather than horizontally (and how this relates to the horizontal and vertical nature of the input representation to layer 1; see Figure 8).

8.4 Experiment 4

The purpose of this experiment was to demonstrate the performance of a task for which the layered structure of the system could be proven to be necessary. The payoff function defining the task is shown in Figure 32. To obtain the highest payoff, the system should learn to generate actions that result in situation changes such that situation (2.5,5) is sought and situation (7.5,5) is avoided. The appendix provides a proof that layer 2 alone cannot solve this task, assuming that the situations are represented in terms of the q_i's.

The training procedure for this experiment was slightly different from that used in the previous experiments. Since the system produced actions that rapidly carried it away from the peak to be avoided, it was necessary to place the system periodically into a situation near this peak in order to provide enough experience in this region. The system was trained for a total of 10,000 steps, with a random situation change after every 20 steps. Most of the situation-to-action mapping was formed early in the experiment, but many additional steps were required to learn the mapping completely near the peak to be avoided. The number of steps could have been reduced if the system would have been placed into those situations that required more experience rather than changing the situation randomly.

Figure 33 shows the resulting vector field display. It is clear that a good situation-to-action mapping has formed. Figure 34 shows the final receptive fields of the layer 1 elements. In addition, the receptive fields of the layer 2 elements are shown in Figure 35. The fact that layer 2 alone could not form these receptive fields can be realized by studying the fields of elements 3 and 4 of layer 2. A weighted sum of the q_i inputs could not form this type of function. However, the receptive fields of elements 1 and 2 of layer 2 could be formed from the q_i's. Thus, for this task the internal representation produced by layer 1 was necessary only for the generation of the dx_2 component of the actions (generated by elements 3 and 4 of layer 2). The dx_1 components of the actions for each situation can be computed by subtracting the output of layer 2's element 1 from the output of element 2. Similarly, the dx_2 components of the actions can be computed by subtracting the output of layer 2's element 3 from the output of element 4. Figure 36 shows the results of these subtractions. Notice that these functions approximate the partial derivatives of the payoff function, which is just what is desired. Finally, Figure 37 shows the network weights that were formed during this experiment.

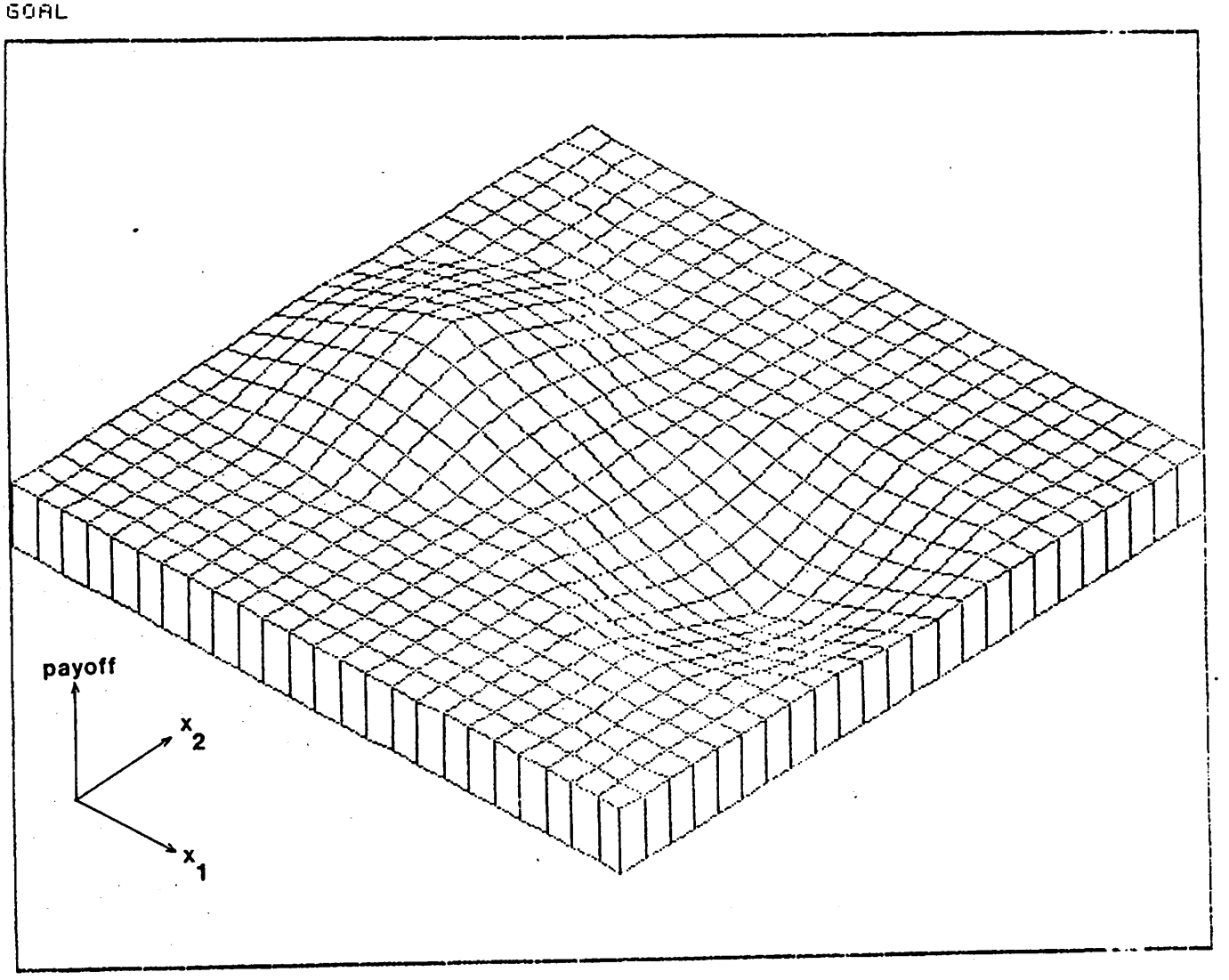

Figure 32: Experiment 4: Payoff Function.

This payoff function defines a most desirable situation, (2.5,5), and a least desirable situation, (7.5,5).

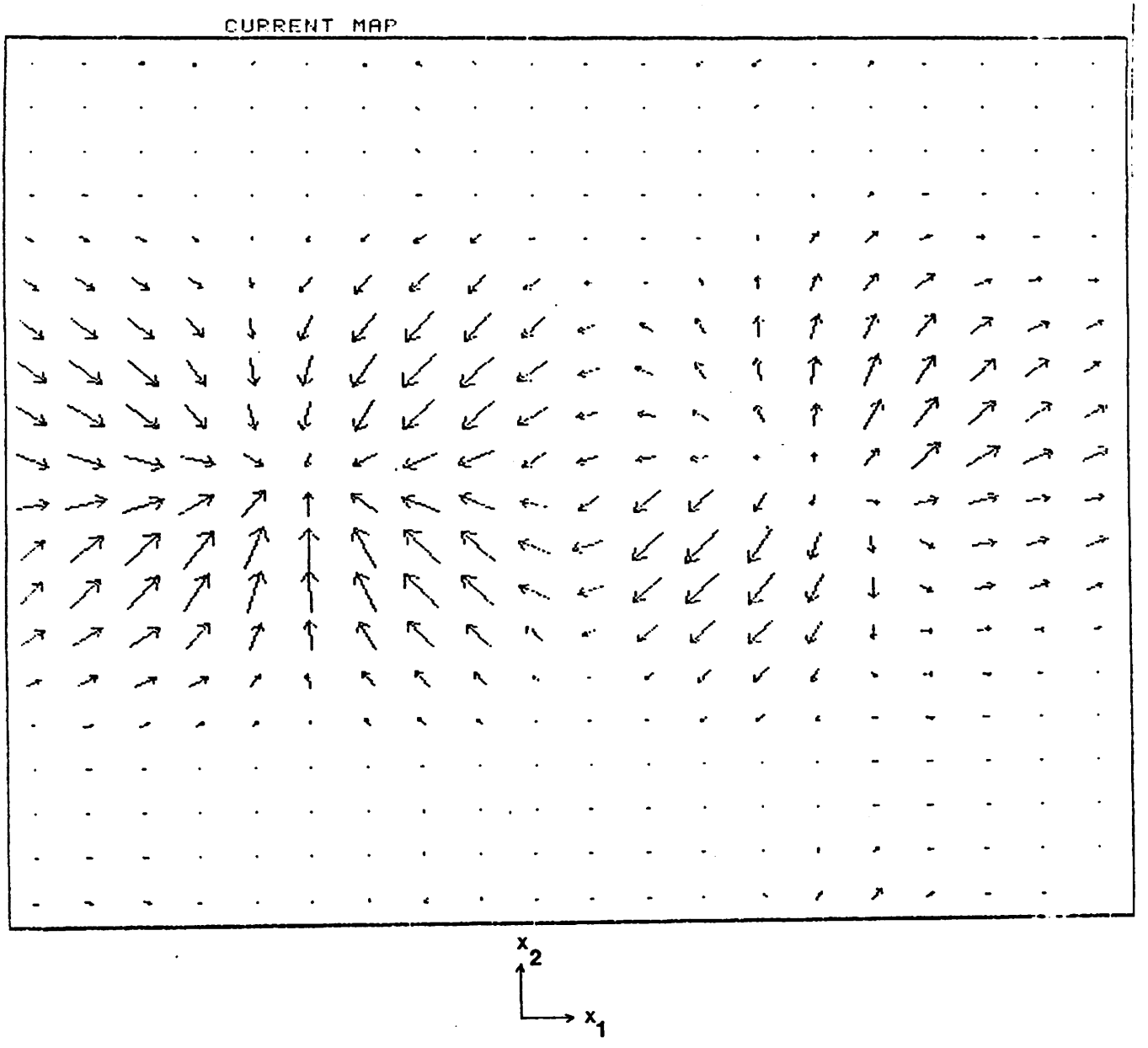

Figure 33: Experiment 4: Vector Field Display.

This is the vector field display obtained after approximately 10,000 steps, with random situation changes after every 20 steps.

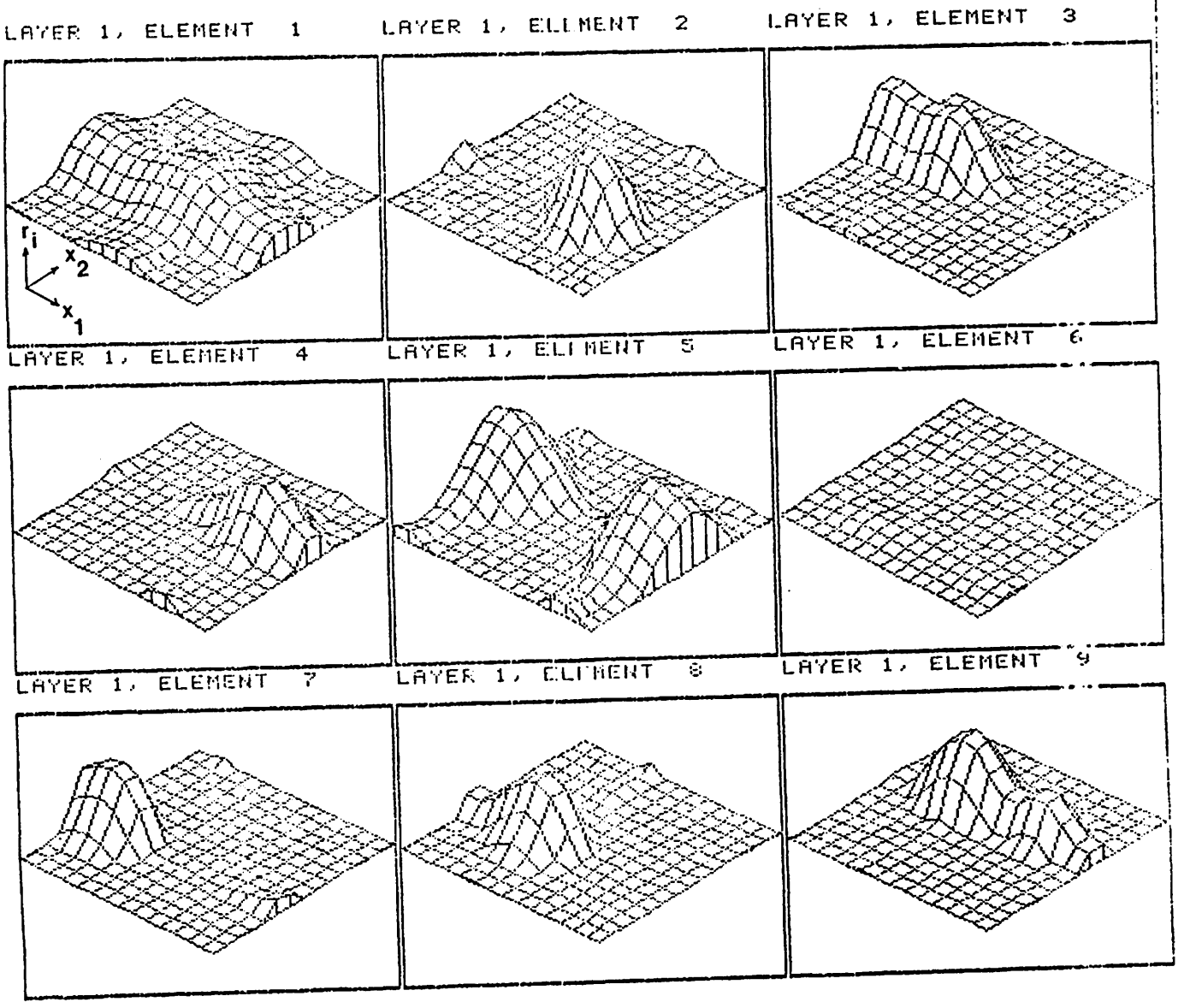

Figure 34: Experiment 4: Functions Computed by Layer 1 Elements.

In this experiment, eight of the layer 1 elements became tuned to regions of the situation space.

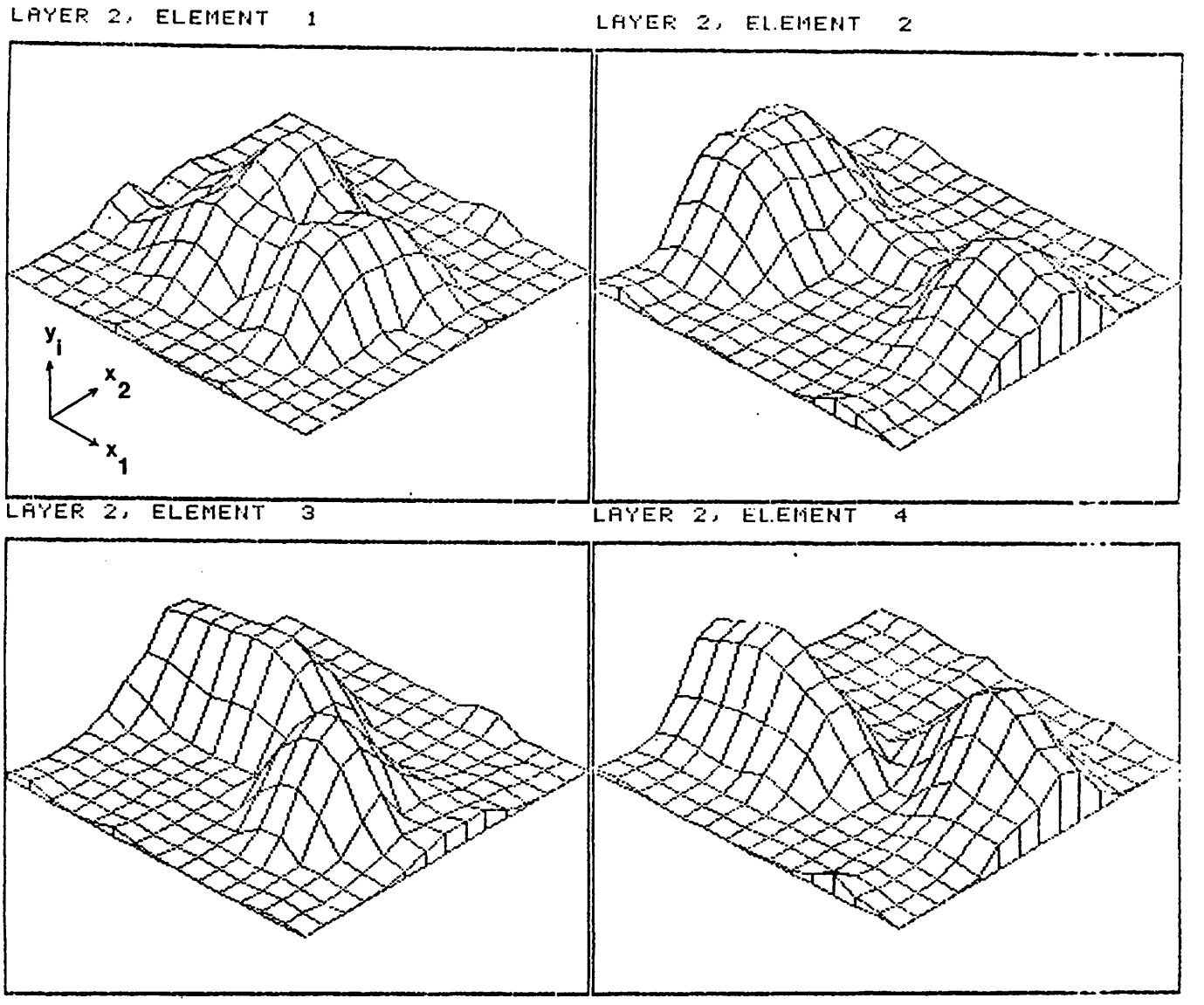

Figure 35: Experiment 4: Functions Computed by Layer 2 Elements.

These graphs can be interpreted as displaying the probability of producing a $-dx_1$, $+dx_1$, $-dx_2$, and $+dx_2$ actions for all situations.

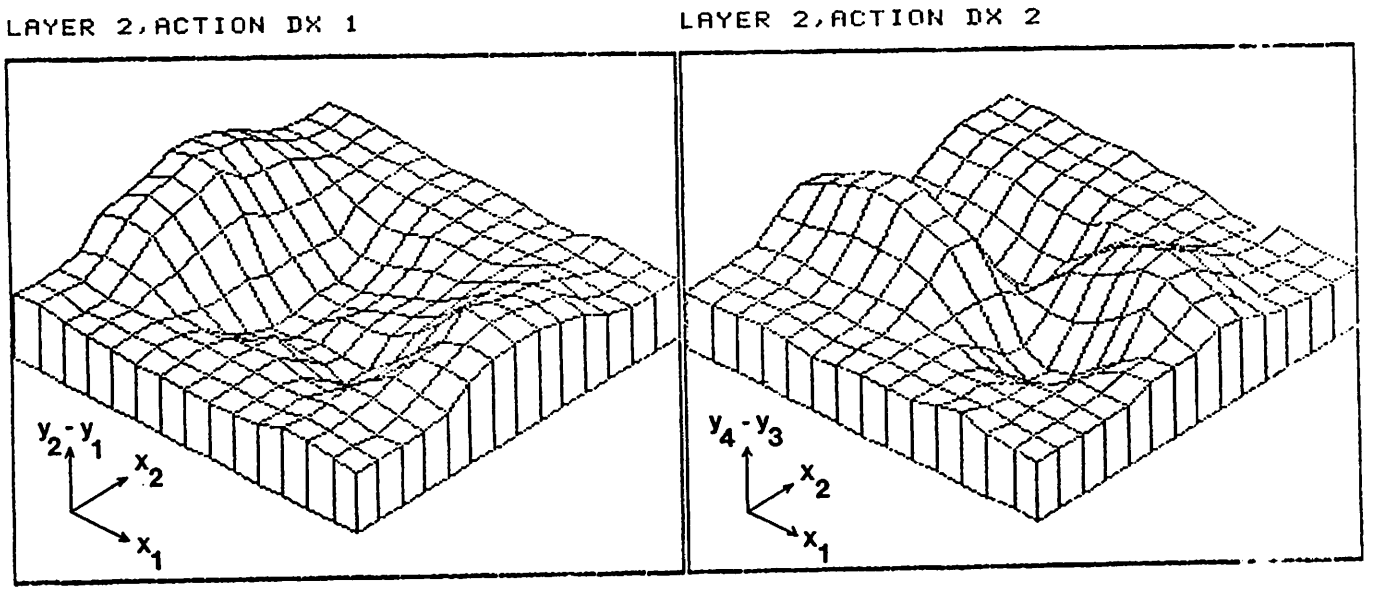

Figure 36: Experiment 4: Final Situation-to-Action Mapping.

An alternative way of displaying the knowledge learned by the system is to graph the values of dx_1 and dx_2 as functions of the situation. This was done by subtracting the output of element 1 from element 2 and element 3 from 4, all of layer 2. The resulting functions approximate the partial derivatives of the payoff function.

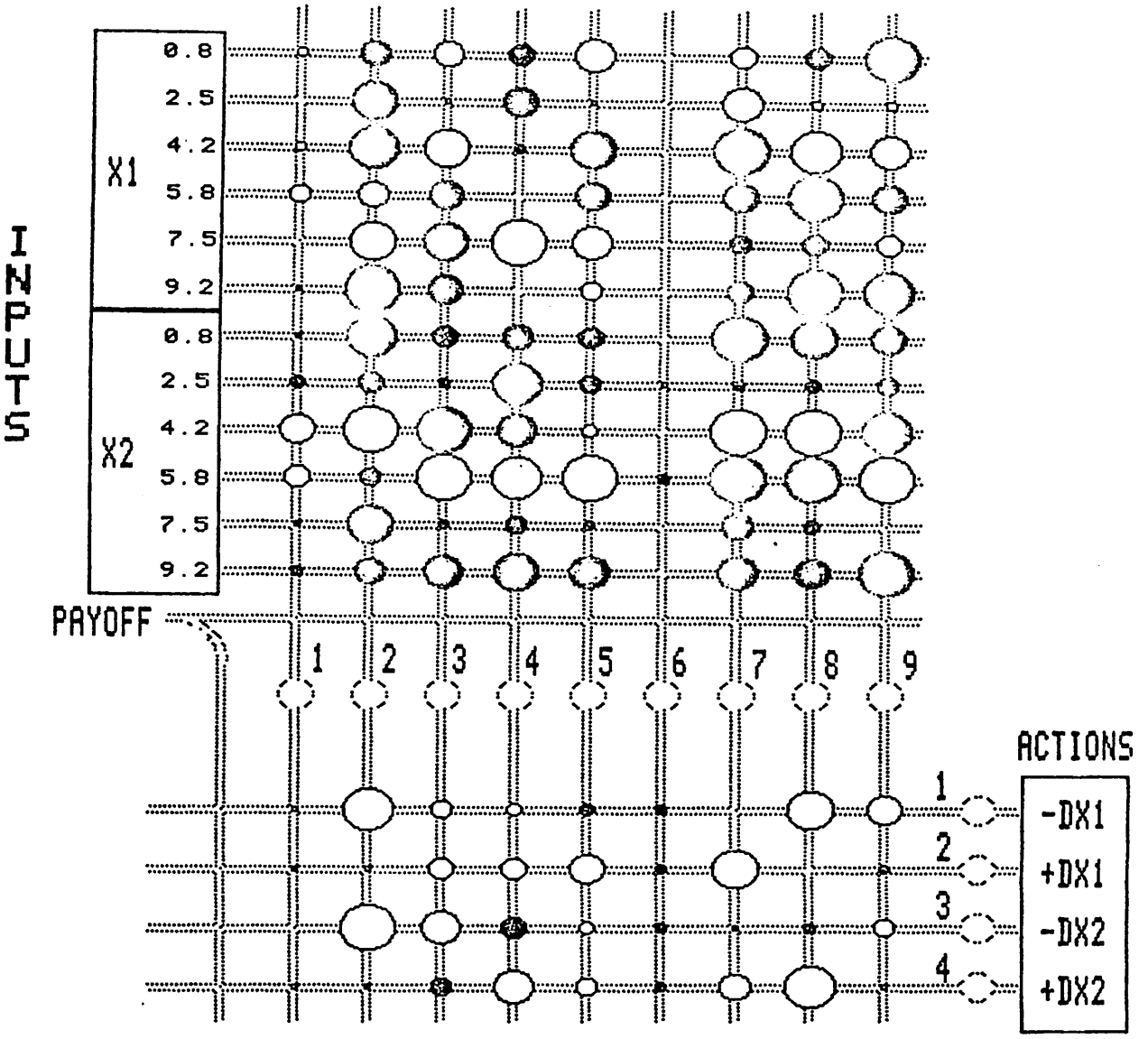

Figure 37: Experiment 4: Network Weights.

The values of the weights show that most of the layer 1 elements were tuned during this experiment.

9.0 DISCUSSION

The system presented in this report illustrates a method by which a reinforcement learning system can acquire, through experience in an environment, an internal representation (a feature encoding) that enables the action-selecting mechanism to perform a task that it might otherwise be unable to do. The capabilities of the system primarily result from its use of the "divide and conquer" approach, closed-loop reinforcement learning, and goal-seeking components that cooperate. Although the demonstrations involved relatively simple control tasks using two-dimensional input and output vectors, it is claimed that this method is extensible to more complex problems of higher dimensionality. However, the following aspects of the method require further investigation before more complex tasks can be attempted.

All of the experiments presented here were initiated with zeroed weights. Certainly if the system is to be applied to a known class of tasks, then all a priori knowledge about those tasks should be used to set some of the initial weight values to produce features that might be useful. This type of information could greatly decrease the time required to search for a good internal representation.

The process by which the layer 1 elements are forced to be tuned to different stimuli should also be investigated. Rather than selecting only one element of layer 1 to be eligible for learning, the layer 1 elements could be divided into smaller

groups with the competition for activity or learning being restricted to within each group. This would result in a greater redundancy of features, which is desirable if the elements are unreliable. One could also apply a small excitatory and large inhibitory field around each layer 1 element, which might result in a more "biological" organization in which groups of neighboring elements would tune to similar situations.

Another aspect to be noted is the unavailability of the original representation (i.e., the x_i's and q_i's), to the action-selecting layer. If these representations were provided as input to layer 2 in addition to any features that might be formed by layer 1, then the search for a good feature set could be much easier. The only required features would be those that are defined on situations for which the action-selecting elements could not generate the correct actions from the original representation. Layer 1 would provide the additional inputs needed to correct, or "de-bug", layer 2's mapping.

Finally, the method does not dictate that the system be composed of only two layers. Perhaps additional layers would increase the efficiency of the search for both the input representation and the output representation. Here again it might be advantageous to provide the original inputs to each layer and send the outputs of each layer to all subsequent layers, including the final action-selecting layer. In this case, a particular layer could be considered as receiving "higher level" inputs from the previous layers, whose function would be

to modify the receptive fields of the elements in that layer.

10.0 BIBLIOGRAPHY

Albus, J. S.: Mechanisms of planning and problem solving in the brain. Math. Biosci. 45, 247-293 (1979)

Anderson, J. A., Hinton, G. E.: Models of information processing in the brain. In: Parallel models of associative memory, pp. 9-48. Anderson, J. A., Hinton, G. E., eds. Hillsdale, NJ: Lawrence Erlbaum Assoc., Inc. (1981)

Arbib, M. A.: The metaphorical brain. New York: Wiley-Interscience 1972

Barr, A., Feigenbaum, E. A.: The handbook of artificial intelligence. Los Altos, Ca: William Kaufmann, Inc. (1981)

Barto, A. G., Anderson, C. W., Sutton, R. S.: Synthesis of nonlinear control surfaces by a layered associative search network. Biol. Cybern. 43, 175-185 (1982)

Barto, A. G., Sutton, R. S.: Landmark learning: an illustration of associative search. Biol. Cybern. 42, 1-8 (1981a)

Barto, A. G., Sutton, R. S., Anderson, C. W.: Adaptive neuron-like elements that can solve difficult learning control problems. COINS Technical Report 82-20, University of Massachusetts, Amherst. (1982)

Barto, A. G., Sutton, R. S., Brouwer, P.: Associative search network: a reinforcement learning associative memory. Biol. Cybern. 40, 201-211 (1981)

Berliner, H. J., Ackley, D. H.: The QBKG system: generating explanations from a non-discrete knowledge representation. Proc. National Conf. on Art. Intell., 213-216 1982)

Block, H. D., Nilsson, N. J., Duda, R. O.: Determination and direction of features in patterns. In: Computer and information sciences: collected papers in learning, adaptation, and control in information systems, pp. 75-110. Tou, J. T., Wilcox, R. H., eds. Washington, D. C.: Spartan Books 1964

Chow, C. K.: A class of nonlinear recognition procedures. IEEE Trans. Sys. Sciences and Cyber. SSC-2, 2, 101-109 (1966)

Chow, C. K., Liu, C. N.: An approach to structure adaptation in pattern recognition. IEEE Trans. Sys. Sciences and Cyber. SSC-2, 2, 73-80 (1966)

Fu, K. S.: Learning control systems -- review and outlook. IEEE Trans. Auto. Control, April, pp. 210-221 (1970)

Fukushima, K.: Visual feature extraction by a multilayered network of analog threshold elements. IEEE Trans. Sys. Sci. Cybern., SSC-5, 4, 322-333 (1969)

Gilstad, D. W., Fu, K. S.: Two-dimensional adaptive model of a human controller using pattern recognition techniques. IEEE Trans. Systems, Man, and Cyber., SMC-1, 3, pp. 261-266 (1971)

Gose, E. E.: Introduction to biological and mechanical pattern recognition. In: Methodologies of pattern recognition, pp. 203-252. Watanbe, S.,ed. New York: Academic Press 1969

Holland, J. H.: Goal-directed pattern recognition. In: Methodologies of pattern recognition, pp. 287-296. Watanbe, S.,ed. New York: Academic Press 1969

Holland, J. H.: Adaptation in natural and artificial systems. Ann Arbor: Univ. of Michigan Press 1975

Kamentsky, L. A., Liu, C. N.: Computer-automated design of multifont print recognition logic. IBM Journal Res. Dev. 1, 2-13 (1963)

Kaufman, H.: An experimental investigation of process identification by competitive evolution. IEEE Trans. Systems Sciences and Cyber., SSC-3, 1, pp. 11-16 (1967)

Klopf, A. H., Gose, E.: An evolutionary pattern recognition network. IEEE Trans. Syst. Sci. Cybern. SSC-5, 3, 247-250 (1969)

Klopf, A. H.: Brain function and adaptive systems - a heterostatic theory. Air Force Cambridge Research Laboratories Research Report AFCRL-72-0164, Bedford, MA, 1972 (A summary appears in: Proc. Int. Conf. Syst., Man, Cybern., IEEE Syst., Man, Cybern. Soc. Dallas, Texas, 1974)

Klopf, A. H.: Goal-seeking systems from goal-seeking components: implications for AI. Cogn. Brain Theory Newsletter 3, 2(1979)

Klopf, A. H.: The hedonistic neuron: a theory of memory, learning, and intelligence. Washington, D.C.: Hemisphere Publishing Corp. 1982

Lewis, P. M.: The characteristic selection problem in recognition systems. IRE Trans. info. theory IT-8, pp. 171-178 (1962)

MacKay, D. M.: Recognition and action. In: Methodologies of pattern recognition, pp. 409-416. Watanbe, S.,ed. New York: Academic Press 1969

Mendel, J. M.: Synthesis of quasi-optimal switching surfaces by means of training techniques. In: Adaptation, learning, and pattern recogntion systems: Theory and applications, pp. 163-195. Mendel, J. M., Fu, K. S., eds. New York: Academic Press 1970

Mendel, J. M., McLaren, R. W.: Reinforcement-learning control and pattern recognition systems. In: Adaptive, learning, and pattern recognition systems: Theory and applications, pp. 287-317. Mendel, J. M., Fu, K. S., eds. New York: Academic Press 1970

Mendel, J. M., Zapalac, J. J.: The application of techniques of artificial intelligence to control system design. In: Advances in control systems: theory and application, v. 6, pp. 2-94. Leondes, C. T., ed. New York: Academic Press 1968

Michie, D., Chambers, R. A.: BOXES: An experiment in adaptive control. Machine Intelligence 2, pp. 137-152. Dale E., Michie, D., eds. Edinburgh: Oliver and Boyd 1968

Minsky, M. L.: Steps toward artificial intelligence. Proc. IRE 49, 8-30 (1961)

Minsky, M. L., Papert, S.: Perceptrons: An introduction to computational geometry. Cambridge, MA: MIT Press 1969

Nagy, G.: Feature extraction on binary patterns. IEEE Trans. Syst. Sci. Cybern., SCC-5, 273-278 (1969)

Nilsson, N. J.: Learning Machines. New York: McGraw-Hill 1965

Raibert, M. H.: A model for sensorimotor control and learning. Biol. Cybern. 29, 29-36 (1978)

Riseman, E. M.: Logical networks for feature extraction. IEEE Trans. Sys., Man, Cyber., SMC-1, 1, 43-55 (1971)

Rosenblatt, F.: Principles of neurodynamics. New York: Spartan Books 1962

Saridis, G. N.: Self-organizing control of stochastic systems. New York: Marcel Dekker, Inc. 1977

Sears, R. W.: Adaptive representation for pattern recognition. IEEE Trans. Sys. Sci. Cyber., SMC-1, 1, pp. 59-66 (1965)

Uhr, L., Vossler, C.: A pattern recognition program that generates, evaluates and adjusts its own operators. Proc. Western Joint Comp. Conf., 555-569 (1961)

Waterman, D. A.: Generalization learning techniques for automating the learning of heuristics. Artificial Intelligence, 1, pp. 121-170 (1970)

Widrow, B., Smith, F. W.: Pattern-recognizing control systems. In: Computer and information sciences: collected papers in learning, adaptation, and control in information systems, pp. 318-343. Tou, J. T., Wilcox, R. H., eds. Washington, D. C.: Spartan Books 1964

A.0 THE NECESSITY OF LAYER 1 FOR EXPERIMENT 4

Consider the ranges of the input variables x_1 and x_2 as each being divided into two intervals, as shown in Figure 38a. Attention will be restricted to those regions of the input space that are contained within the intersections of these intervals, and specifically to the points p_1, p_2, p_3, and p_4. The arrows indicate the actions that should be learned at each point. Figures 38b and 38c show these actions in terms of their x_1 and x_2 components.

Each point is encoded by the maximum value of two of the quantized measurements. Thus, the points are represented by vectors of the form $(q_1^i, q_2^i, q_3^i, q_4^i, 1)$ for point p_i. The maximum value of a q_j^i is specified by v. The x_2 components of the actions are specified as either +u or -u. The following is a list of the points, the vectors representing them, and the x_2 component of the actions at each point.

p_1 = (v 0 0 v 1) -u
p_2 = (v 0 v 0 1) +u
p_3 = (0 v 0 v 1) +u
p_4 = (0 v v 0 1) -u

Layer 2 outputs the action +u only if the $+dx_2$ element produces a value greater than the value produced by the $-dx_2$ element, and vice versa for action -u. If we assume that layer 2 can learn these actions, then the vectors $(a_1,\ldots,a_5)$ and $(b_1,\ldots,b_5)$ must exist, such that the following inequalities are satisfied:

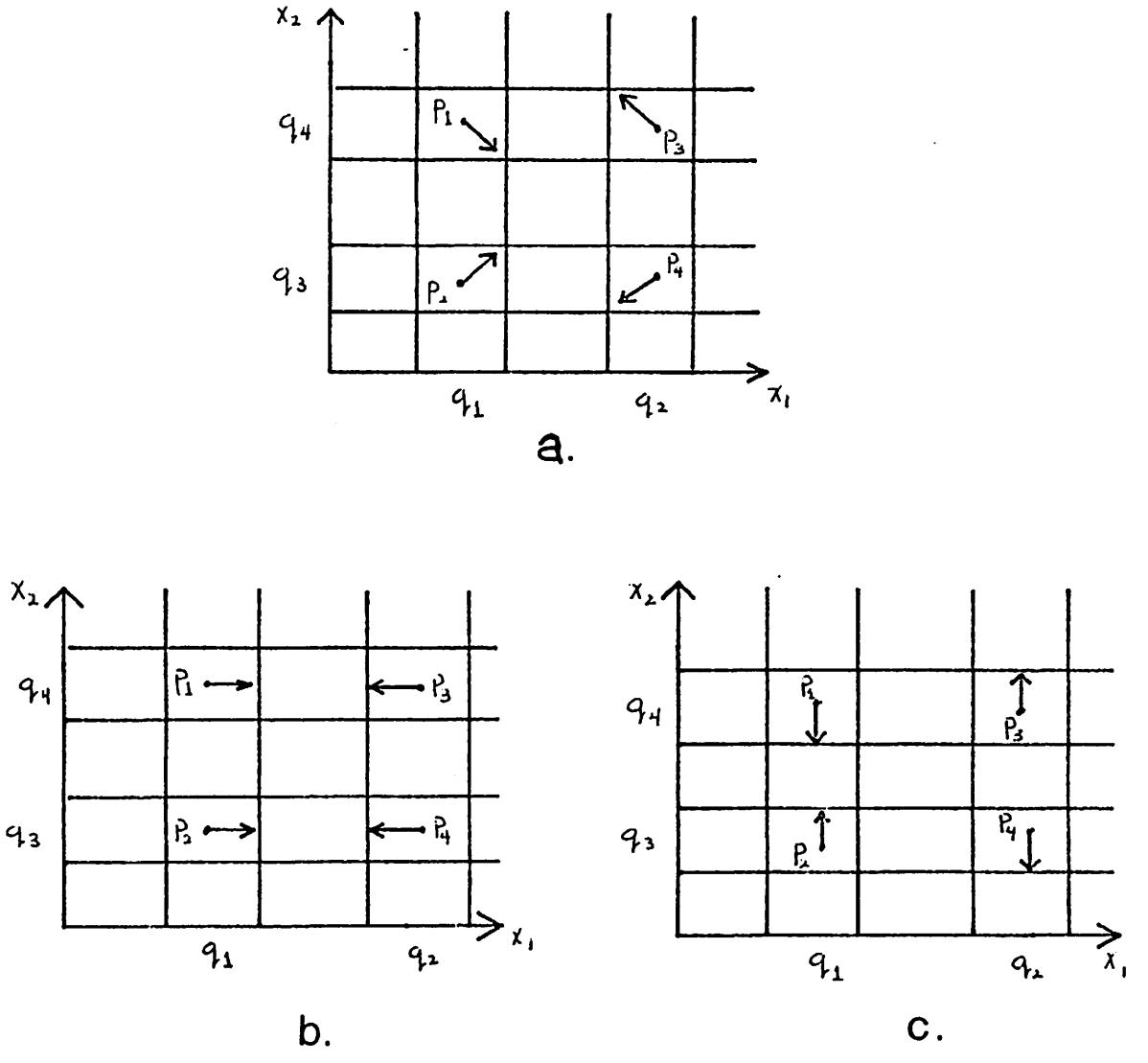

Figure 38: Diagrams used in the Proof of the Necessity of Layer 1 for Experiment 4.

$$\sum_{j=1}^{5} a_j q_j^i > \sum_{j=1}^{5} b_j q_j^i \quad , \qquad \text{for } i = 2,3$$

$$\sum_{j=1}^{5} a_j q_j^i < \sum_{j=1}^{5} b_j q_j^i \quad , \qquad \text{for } i = 2,3$$

These are expanded to

$$va_1 + va_3 + a_5 > vb_1 + vb_3 + b_5 \qquad (1)$$

$$va_2 + va_4 + a_5 > vb_2 + vb_4 + b_5 \qquad (2)$$

$$va_1 + va_4 + a_5 < vb_1 + vb_4 + b_5 \qquad (3)$$

$$va_2 + va_3 + a_5 < vb_2 + vb_3 + b_5 \qquad (4)$$

Adding va_4 to (1) and va_3 to (3) we arrive at

$$va_1 + va_3 + va_4 + a_5 > vb_1 + vb_3 + b_5 + va_4$$

$$va_1 + va_3 + va_4 + a_5 > vb_1 + vb_4 + b_5 + va_3$$

which are combined into

$$vb_1 + vb_4 + b_5 + va_3 > va_1 + va_3 + va_4 + a_5 > vb_1 + vb_3 + b_5 + va_4$$

and reduced to

$$vb_4 + b_5 + va_3 > vb_3 + b_5 + va_4. \qquad (5)$$

In a similar manner, add va_3 to (2) and va_4 to (4) to get

$$va_2 + va_3 + va_4 + a_5 > vb_2 + vb_4 + b_5 + va_3$$

$$va_2 + va_3 + va_4 + a_5 > vb_2 + vb_3 + b_5 + va_4$$

which are combined into

$$\sum_{j=1}^{5} a_j q_j^i > \sum_{j=1}^{5} b_j q_j^i \quad , \qquad \text{for } i = 2,3$$

$$\sum_{j=1}^{5} a_j q_j^i < \sum_{j=1}^{5} b_j q_j^i \quad , \qquad \text{for } i = 2,3$$

These are expanded to

$$va_1 + va_3 + a_5 > vb_1 + vb_3 + b_5 \tag{1}$$

$$va_2 + va_4 + a_5 > vb_2 + vb_4 + b_5 \tag{2}$$

$$va_1 + va_4 + a_5 < vb_1 + vb_4 + b_5 \tag{3}$$

$$va_2 + va_3 + a_5 < vb_2 + vb_3 + b_5 \tag{4}$$

Adding va_4 to (1) and va_3 to (3) we arrive at

$$va_1 + va_3 + va_4 + a_5 > vb_1 + vb_3 + b_5 + va_4$$

$$va_1 + va_3 + va_4 + a_5 > vb_1 + vb_4 + b_5 + va_3$$

which are combined into

$$vb_1 + vb_4 + b_5 + va_3 > va_1 + va_3 + va_4 + a_5 > vb_1 + vb_3 + b_5 + va_4$$

and reduced to

$$vb_4 + b_5 + va_3 > vb_3 + b_5 + va_4. \tag{5}$$

In a similar manner, add va_3 to (2) and va_4 to (4) to get

$$va_2 + va_3 + va_4 + a_5 > vb_2 + vb_4 + b_5 + va_3$$

$$va_2 + va_3 + va_4 + a_5 > vb_2 + vb_3 + b_5 + va_4$$

which are combined into

$$vb_2 + vb_3 + b_5 + va_4 > va_2 + va_3 + va_4 + a_5 >$$

$$vb_2 + vb_4 + b_5 + va_3$$

and reduced to

$$vb_3 + b_5 + va_4 > vb_4 + b_5 + va_3. \qquad (6)$$

The inequality (6) directly contradicts (5). Therefore, layer 2 by itself, cannot learn the function that generates the x_2 component of the actions from the vector representation. The services of layer 1 are required to transform the vector representation into a different representation from which layer 2 can produce the correct actions.