Machine Learning Research Group Working Paper 91-5

Visualizing Learning and Computation in Artificial Neural Networks

Mark W. Craven
Jude W. Shavlik
Computer Sciences Department
University of Wisconsin
1210 West Dayton St.
Madison, WI 53706
(608) 263-0475
craven@cs.wisc.edu

Abstract

Scientific visualization is the process of using graphical images to form succinct and lucid representations of numerical data. Visualization has proven to be a useful method for understanding both learning and computation in artificial neural networks. While providing a powerful and general technique for inductive learning, artificial neural networks are difficult to comprehend because they form representations that are encoded by a large number of real-valued parameters. By viewing these parameters pictorially, a better understanding can be gained of how a network maps inputs into outputs. In this article, we survey a number of visualization techniques for understanding the learning and decision-making processes of neural networks. We also describe our work in *knowledge-based neural networks* and the visualization techniques we have used to understand these networks. In a knowledge-based neural network, the topology and initial weight values of the network are determined by an approximately-correct set of inference rules. Knowledge-based networks are easier to interpret than conventional networks because of the synergy between visualization methods and the relation of the networks to symbolic rules.

Keywords: neural networks,
knowledge-based neural networks,
scientific visualization

(Submitted (12/91) to the *International Journal on Artificial Intelligence Tools.)*

1 Introduction

Artificial neural networks (ANNs) provide an approach to machine learning which is inspired by the architecture of the brain; ANNs are characterized by large numbers of simple processing units linked to each other by weighted connections. Neural network learning algorithms have been successfully applied to a number of real-world problems as varied as recognizing handwritten digits (LeCun90), finding important features in DNA (Noordewier91), and learning to pronounce English text (Sejnowski87). In addition to these practical successes, several empirical studies have concluded that neural networks often provide better performance than common symbolic learning algorithms (Atlas89; Fisher89; Shavlik91; Weiss89). Neural learning algorithms, such as backpropagation (Rumelhart86), enable an artificial neural network to inductively learn how to solve a problem by exploiting regularities in a set of training examples. A fundamental problem of ANNs, however, is that the information that they encode cannot be easily understood by humans because concepts are represented by a large number of real-valued parameters. A number of researchers have employed the techniques of *scientific visualization* to assist in understanding neural networks. Scientific visualization is a newly-emerging field that involves using computer graphics to aid in understanding complex systems. An underlying premise of visualization is that people are better at recognizing regularities, anomalies, and trends in *images* than in long lists of numbers. The parameters of an ANN can be more easily understood when they are represented using graphical attributes such as color, size, and spatial organization.

This article surveys visualization techniques that have been used to understand the learning and decision-making processes of ANNs. Also discussed are the techniques that our research group has used to understand *knowledge-based* neural networks. In a knowledge-based neural network, the topology and initial weight values of the network are determined by an approximately-correct set of inference rules.

The next section provides a brief introduction to artificial neural networks. The third section discusses why ANNs are difficult to comprehend, why it is important to be able to understand the representations that they form, and how visualization can help. Section 4 provides a short introduction to visualization and the fifth section surveys visualization techniques that have been applied to ANNs. The sixth section provides a brief introduction to knowledge-based neural networks and discusses our experiences with using visualization methods to understand these networks. Section 7 outlines approaches other than visualization that have been taken to understand the representations formed in ANNs. The final section discusses some of the recurring issues that arise in visualizing ANNs.

2 Artificial Neural Networks

This section provides a brief overview of computation and learning in artificial neural networks. It is assumed that the reader is already familiar with the basic principles of ANNs. More comprehensive introductions to the topic can be found elsewhere (Hertz91; Knight90). This paper will discuss neural network visualization only in the context of feed-forward ANNs, although many of the techniques presented herein are applicable or can be generalized to other classes of ANNs. Additionally, we will focus on the application of ANNs to *classification*. Classification involves determining, for each pattern given to a network, whether or not the pattern is a member of one or more predefined classes of interest. Although they have been applied to other types of problems, most of the work in neural networks has involved classification.

2.1 Computation in ANNs

A feed-forward artificial neural network is composed of several layers of simple processing *units*. The state of a unit at any given time is represented by its *activation*, which is typically a real-valued number in the range [0, 1]. The *input* layer of a network contains units whose activations represent feature values of the problem domain to which the ANN is being applied. The units in the *output* layer of a network represent the decisions made by the network. Interposed between the input units and the output units, there can be a number of *hidden* layers of units. The units of a network are related by weighted connections. Figure 1 depicts a simple ANN which implements the exclusive-or (XOR) function [1] when given input values that are either 0 or 1.

A network which has only input units and a layer of thresholding output units is capable only of making linear discriminations in its input space (Minsky69). In order to make more complex discriminations, it it necessary to add hidden units to the network. The role of hidden units is to transform the input space into another space in which it is more profitable for the output units to make linear discriminations.

Computation in a feed-forward network proceeds by setting the activation values of the input units to represent a particular instance in the problem domain. The activation of the inputs feeds forward through the weighted connections to the units at the hidden layers and then to units at the output layer. The answer provided by the network is the resultant activations on the output units.

The activation of a hidden or an output unit is determined by passing the weighted input to the unit through a *transfer* or *activation* function. The net input to a unit in a hidden or output layer is given by:

[1]Exclusive-or is a boolean function of two inputs that returns *true* when at least one, but not both, of the inputs is true.

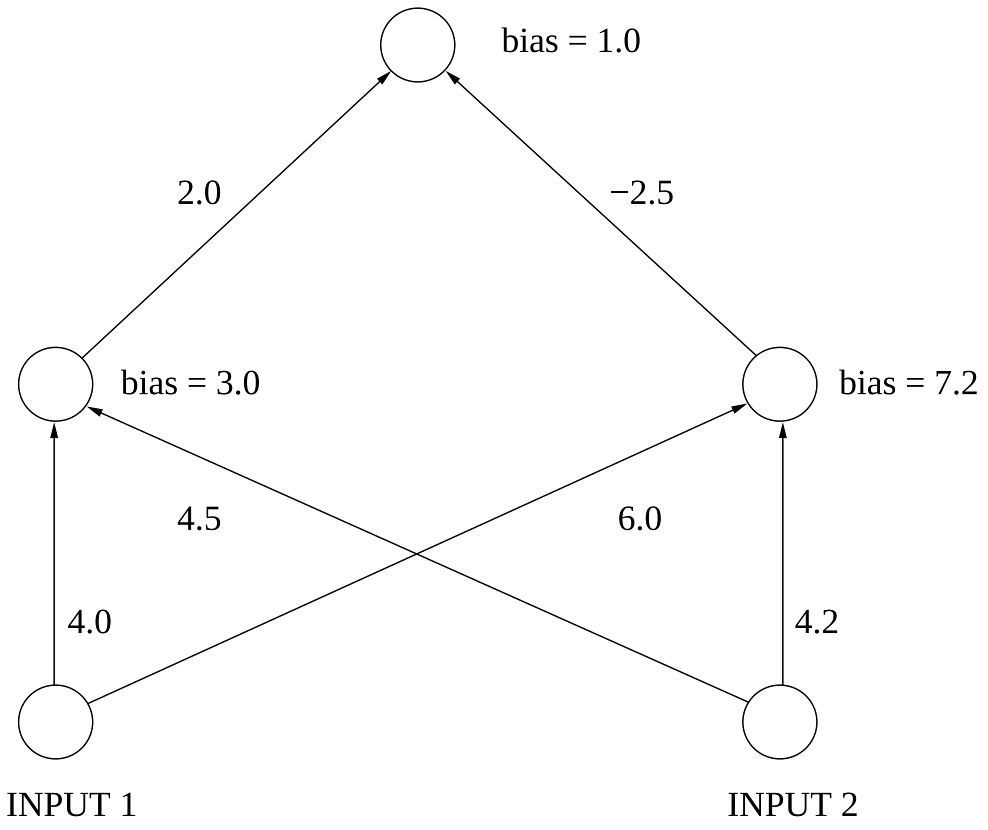


Figure 1: **A simple artificial neural network.** When given input values which are either 0 or 1, this network implements the XOR function. The network can be thought of as returning *true* when the activation on the output unit is greater than 0.5.

$$net_{pi} = \sum_j w_{ij} o_{pj} \tag{1}$$

where w_{ij} is the weight from unit j to unit i, and o_{pj} is the output of unit j in response to pattern p. One possible transfer function is a threshold function, where the output, o_{pi}, of a hidden or output unit is given by:

$$o_{pi} = \begin{cases} 1 & \text{if } net_{pi} > \theta \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where θ specifies the threshold which must be exceeded by the net input in order for the unit to be active. Threshold transfer functions are usually not used in networks with hidden units. A more commonly used transfer function, the sigmoid, is a continuous approximation of the threshold function in which the output is determined as follows:

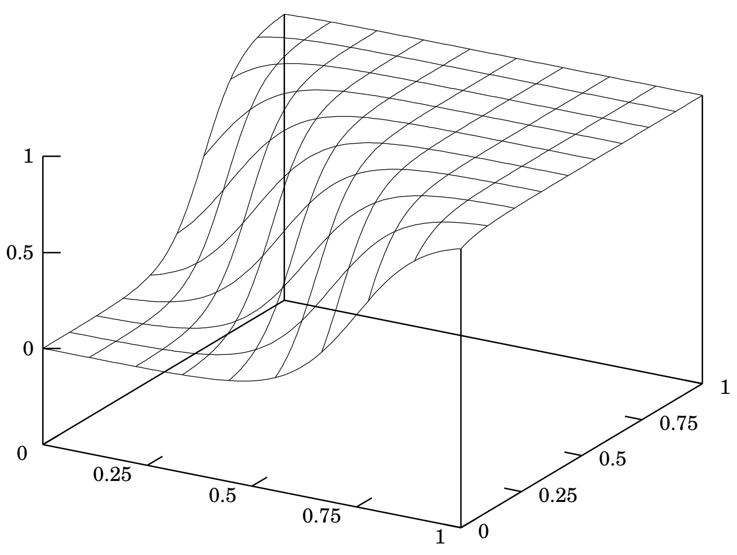


Figure 2: **The decision surface formed by a sigmoidal transfer function.** The input space is that of the leftmost hidden unit in Figure 1.

$$o_{pi} = \frac{1}{1 + e^{-(\sum_j w_{ij} o_{pj} - bias_i)}} \tag{3}$$

The *bias* of each unit is a parameter that has a similar function to θ in the threshold transfer function. Although the sigmoid is a continuous function, its "threshold" can be thought of as the point at which it is steepest; this occurs where $net_{pi} = bias_i$.

To understand the role of a unit in an ANN, it is helpful to visualize the activation value of the unit in terms of its input values. Each unit forms a *decision surface* in its N-dimensional input space (where N is the number of connections impinging on the unit) that determines the activation value. Each dimension of the input space is defined by an incoming connection to the unit. The decision surface is defined by the activation values that would occur for the points in the input space. The nature of the surface is determined by the transfer function.

Figure 2 and Figure 3 illustrate the decision surfaces formed by the two hidden units of the network shown in Figure 1. Both surfaces result from a sigmoidal transfer function. Figure 4 shows the decision surface that would result for the rightmost hidden unit if a threshold transfer function were to be used. The effect of a threshold unit is to partition the input space with an (N-1)-dimensional hyperplane. The ouput of the unit depends on which side of the hyperplane the net input to the unit falls: if the net input is not sufficient to exceed the threshold then the activation of the unit will be 0, otherwise it will be 1. The effect of a sigmoidal unit is similar. The difference, however, is that there is not a hard boundary as in the case of a threshold unit. Instead, there is a gradation of activation values.

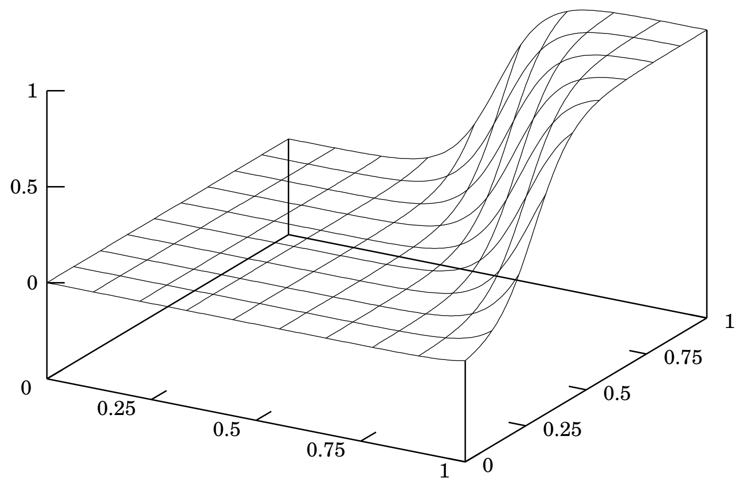


Figure 3: **The decision surface formed by a sigmoidal transfer function.** The input space is that of the rightmost hidden unit in Figure 1.

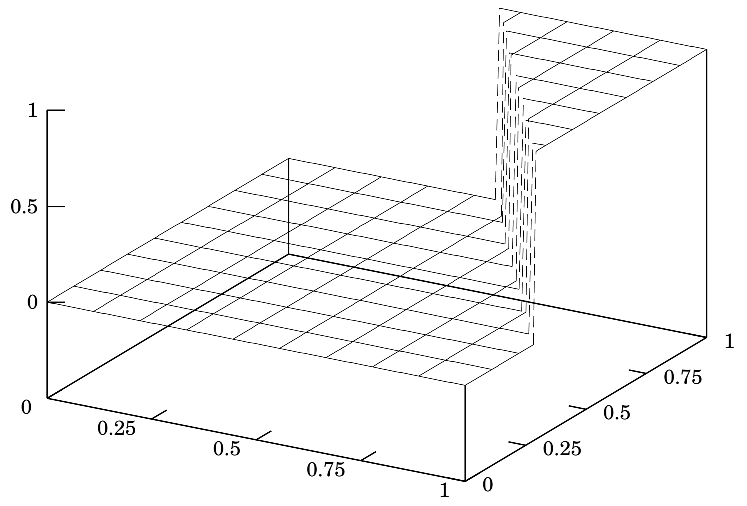


Figure 4: **The decision surface formed by a threshold transfer function.** The input space is that of the rightmost hidden unit in Figure 1. The bias value is treated as a threshold in this figure.

2.2 Learning in ANNs

Learning in a neural network involves modifying the weights and biases of the network so that the network produces the correct response for each of a number of training examples. The weights and biases can be adjusted after each presentation of the entire set of training examples, or they may be modified after each individual training example has been presented. We assume the latter approach for our description of the learning process. Learning for a given training example proceeds as follows:

- The activations of the input units are set according to the feature values of the example.
- Activations are propagated through the network to the output units.
- The activations of the output units are compared to the desired (or *target*) activations for the example and the amount of error at the output units is calculated.
- An error signal is propagated backward through the net to the hidden units, and the weights of the network are slightly modified to reduce the error at each unit.

Since only small changes are made to the weights on each iteration of this process, it is typically repeated numerous times for each training example.

There are several learning algorithms for feed-forward ANNs, e.g., (Barnard89; Moody89; Rumelhart86). The most common of these is the backpropagation algorithm (Rumelhart86). In backpropagation, the change to each weight is determined as follows:

$$\Delta w_{ij} = \epsilon \delta_{pi} o_{pj} \tag{4}$$

where w_{ij} is the weighted connection from unit j to unit i, δ_{pi} represents the error signal for pattern p at unit i, o_{pj} is the activation at unit j in response to pattern p, and ϵ is a parameter called the *learning rate* that determines the size of the weight change. The error signals are calculated by a recursive process. First the error at the output units is calculated by:

$$\delta_{pi} = (t_{pi} - o_{pi}) f'_i(net_{pi}) \tag{5}$$

where t_{pi} is the desired output, or *target*, for the ith unit when pattern p is presented, and $f'_i(net_{pi})$ is the derivative of the transfer function with respect to net input at unit i for pattern p. The error at the output units is then *backpropagated* to the hidden units:

$$\delta_{pi} = f'(net_{pi}) \sum_k \delta_{pk} w_{ki} \tag{6}$$

The process of backpropagating errors is one of "blame" assignment. The activations of the output units are determined by the activations of hidden units, which are in turn determined by the activations of the input units (or a lower level of hidden units). Error at the output units may be due, not only to the weights directly connected to the outputs, but also to weights farther down in the network (the weights impinging on the hidden units). In order to adjust the weights farther down, it is necessary to backward-propagate error values down to the lowest hidden units which contributed to erroneous output unit activations.

The biases of units must also be adjusted during learning. This is done by treating the biases as weights. A unit's bias can be thought of as just another weight feeding into the unit. This special weight (the bias) is treated as if it were connected to a unit whose activation is always -1.

3 Representations Learned in ANNs

An important criterion by which any machine learning algorithm should be judged is the comprehensibility of the representations formed by the algorithm. That is, does the algorithm encode the information it learns in such a way that it may be inspected and understood by humans? There are four reasons why this is an important criterion. First, if the designers and end-users of a learning system are to be confident in the performance of the system, then they must understand how it arrives at its decisions. Second, learning algorithms may discover important features in the input data whose importance was not previously recognized. If the representations formed by the algorithm are comprehensible, then these discoveries can be made accessible to human review. Third, if the representations are understandable, then an *explanation* of the classification made by a network on a particular case can be garnered. Finally, some researchers use neural networks to refine an existing body of knowledge (Pratt91; Towell90). When a network is being used in this way, it is important to understand the changes to the knowledge that have been imparted during the training process.

In order to understand how a network is making decisions – that is, how it is mapping input activations into output activations – it is necessary to answer two broad questions. First, what intermediate *concepts* are being constructed by the hidden units? Intermediate concepts are higher-level features which are derived from the input features in order to solve the problem at hand. Second, how do the output units use these intermediate concepts to reach their activations, the final answers of the network?

There are three primary reasons why these questions are difficult to answer:

- information is encoded by real-valued parameters;
- concept representations tend to be distributed across many units;
- network units typically have many incoming connections.

Each of these will be discussed in turn.

As mentioned previously, neural network algorithms learn to solve a particular problem by adjusting the weights and biases in the network; it is these weights and biases that capture the concepts learned. Networks for solving real-world problems typically have several thousand connections. Hence, understanding the representations formed during learning requires making sense out of this multitude of real-valued parameters.

Moreover, ANNs tend to form representations which are distributed (Hinton86). This means that each concept represented may be encoded by the activations of many network units, and each unit may play a part in representing many different concepts. Thus, understanding the representation for some concept is usually more difficult than understanding the decision surface of a single unit.

A further difficulty in understanding ANNs is that network units typically have many incoming connections. It is easy enough to graphically represent decision surfaces for units with a two-dimensional input space. Even three-dimensional input spaces can be depicted, especially when the decision surface is (or can be approximated by) a two-dimensional hyperplane. In cases where the input unit has more than three incoming connections, however, the dimensionality of the input space is too large to be clearly displayed.

This article focuses on scientific visualization as an approach to understanding ANNs. A neural network's real-valued parameters, distributed representations, and high-dimensionality can be made more perspicuous by graphically depicting salient aspects of the network's activity.

4 Scientific Visualization

Scientific visualization involves transforming numeric data into visual forms which can be more readily understood (Brown87; DeFanti89; Tufte83). Many scientific disciplines involve the analysis of large bodies of numeric data. The numeric form and sheer quantity of the data make it a painstaking and often intractable process to make sense of it. Visualization involves communicating these vast quantities of data to users in a qualitatively different form, one in which regularities and anomalies are often much more apparent. The value of visualization derives from the highly-developed visual pattern-recognition abilities of humans.

Visualization techniques transform the values of variables into visual forms in which color, shading, size, thickness, shape, and spatial arrangement communicate information about the variables. Additionally, dynamic characteristics of the data can be conveyed through animation. Visualization of two-dimensional data usually involves charts and graphs. For three-dimensional data, surface and volume models can be used. For higher-dimensional data, two-dimensional projections are chosen for viewing. These projections can be either planar cross-sections or non-linear subspaces of the higher-dimensional space.

Visualization can assist the scientist in a number of ways. Global and local patterns in the data may become more evident. The effects of changes in parameters can be readily grasped. Errors and anomalies in the data can often be easily detected since they appear as visual anomalies. Additionally, visualization techniques can provide interactive mechanisms which enable a user to adjust parameters and quickly see the effects of such changes.

5 ANN Visualization Methods

In this section, we describe a number of visualization techniques that have been used to understand ANNs. These techniques provide insight into both the decision-making process and the learning process of neural networks. Although many of the techniques discussed in this section were originally described in terms of understanding the decision-making processes of trained networks, all of the methods described below can be (or have been) generalized so that they dynamically depict a network's evolution during learning.

5.1 Hinton Diagrams

One of the first methods developed to visualize ANNs was the *Hinton diagram* (Hinton86). The Hinton diagram provides a compact visual display of the weights and biases related to a particular unit in a network. Figure 5 shows a set of Hinton diagrams for the network shown in Figure 1. The diagram for a unit shows the signs and magnitudes of all of the incoming and outgoing weights as well as the sign and magnitude of the unit's bias. Each weight is represented by a box drawn on the diagram. The area of the box represents the weight's magnitude while the color of the box indicates the sign of the weight. Typically, white boxes indicate positive weights and black boxes indicate negative weights. The diagram is organized so that each unit in the network has an assigned position in the diagram. Typically, output units occupy positions at the top of the diagram, hidden units occupy positions in the middle of the diagram, and input units have positions at the bottom of the diagram. Therefore the position of a box in a diagram indicates the unit at the other end of the weight. A bias for a given unit is drawn in that unit's diagram in the position where weights to and from the unit are shown in the other diagrams. In other words, a bias is illustrated as a weight from a unit to itself.

Hinton diagrams help in understanding an ANN by providing a concise visual representation of the network's weights and biases. Each diagram makes it easy to see the signs and magnitudes of the weights that contribute to a unit's activation, and the relative influence of the unit's activation on the units at the next level in the network. For some problems, the global weight picture provided by the diagram can offer a readily-understood depiction of the features being measured by hidden units. For example, Pomerleau (Pomerleau89) uses such a diagram to help understand an

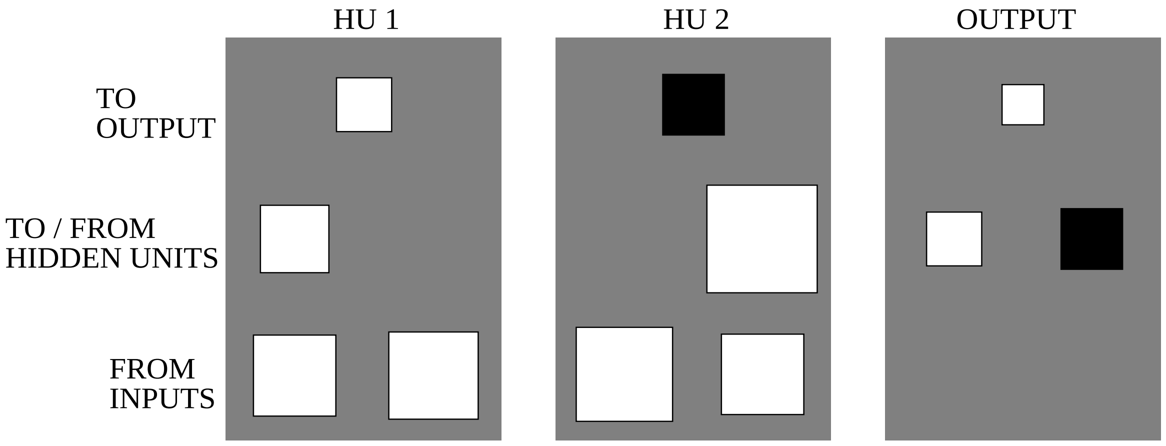


Figure 5: **Hinton diagrams.** These diagrams depict, from left to right, the two hidden units and the output unit of the network depicted in Figure 1. The boxes in the lower part of each diagram depict weights from input units, the boxes in the middle of each diagram depict weights from (to) hidden units, and the box at the top of each diagram depicts a weight to the output unit. A unit's bias is drawn in the position in the unit's diagram where weights to and from the unit are shown in the other diagrams.

ANN that guides an autonomous land vehicle in following a road. The diagram for a hidden unit in this network is able to clearly depict the patterns that the unit has learned to recognize in the video images which serve as input. Similarly, Tesauro and Sejnowski (Tesauro89) employ a Hinton diagram to gain insight into a network that learns to play backgammon. They are able to identify several types of board configurations that the network has learned to emphasize in making its decision. Still, for most problems, the Hinton diagram is a rather weak method for visualization. The topology of a network is not readily apparent from a set of diagrams. Furthermore, the diagram does not clearly show how a unit partitions its input space.

5.2 Bond Diagrams

Wejchert and Tesauro developed a visualization method that they call the *bond diagram* (Wejchert90). The bond diagram, like the Hinton diagram, illustrates the sign and magnitude of each weight and bias in a network. The primary strength of the diagram, however, is that the topology of the network is explicitly displayed in the diagram. Figure 6 shows the bond diagram of the network in Figure 1. Each unit is represented by a disk. For hidden and output units, the size of the disk indicates the magnitude of the unit's bias; weights are represented by bonds linking the disks. The amount of the bond that is displayed indicates the magnitude of the weight. Different colored bonds are used to distinguish positive and negative weights. Bonds can also be used to represent the rate of change of the weights.

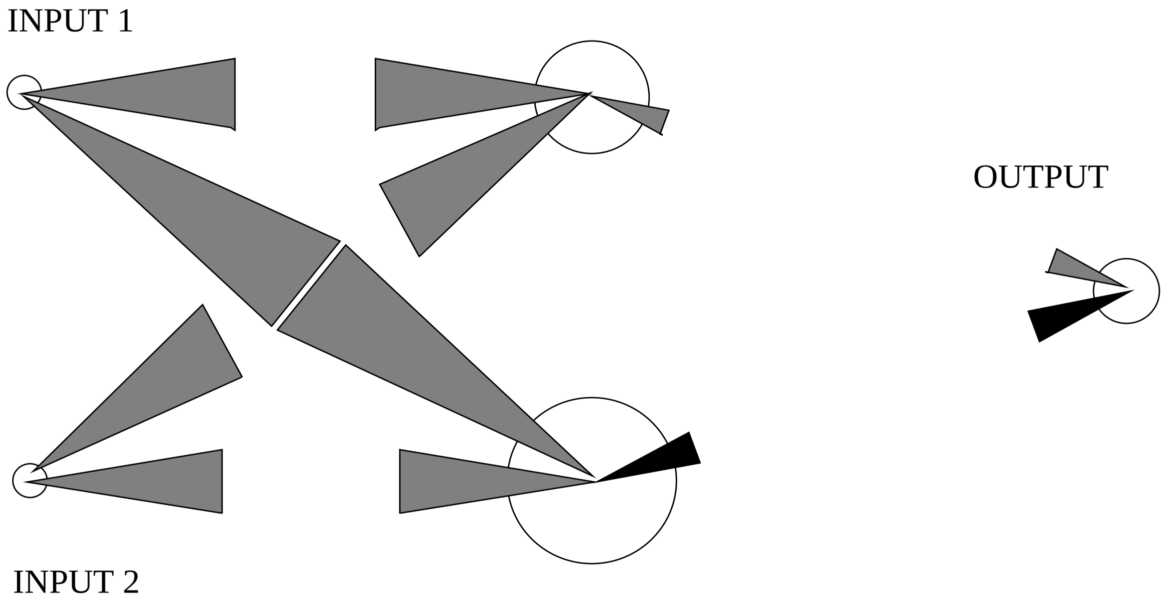


Figure 6: **A bond diagram.** The diagram depicts the network shown in Figure 1. The size of a unit indicates the magnitude of its bias. Positive and negative weights are shown as light and dark shaded bonds, respectively. The magnitude of a weight is indicated by the amount of the bond which is shaded.

Like the Hinton diagram, the bond diagram graphically depicts the values of the network weights and biases. Unlike the Hinton diagram, the bond diagram makes the topology of the network explicit in its graphics. A criticism of the diagram is that, although it depicts both weights and biases, it is difficult to gauge the relative magnitudes of the weights versus the biases. Since different geometric forms are used to depict weights and biases, it is not easy to see how the weights stack up against a bias. This is not a problem with Hinton diagrams since both weights and biases are represented with the same graphical object (boxes). It is important to be able to see the relative magnitudes of weights and biases so that questions such as "which input units need to be active in order for the net input to exceed the threshold (bias) of this hidden unit?" can be readily answered.

5.3 Hyperplane Diagrams

As mentioned previously, a neural network unit with a thresholding activation function partitions its N-dimensional input space into two regions with an (N-1)-dimensional hyperplane. This effect can be seen in Figure 4; the hyperplane in the two-dimensional input space is defined by the vertical face of the decision surface. A unit with a sigmoidal transfer function has a similar effect. One way to visualize the learning process is to graphically display the movement of a hyperplane in the input space of the unit that the hyperplane represents (Munro91; Pratt91).

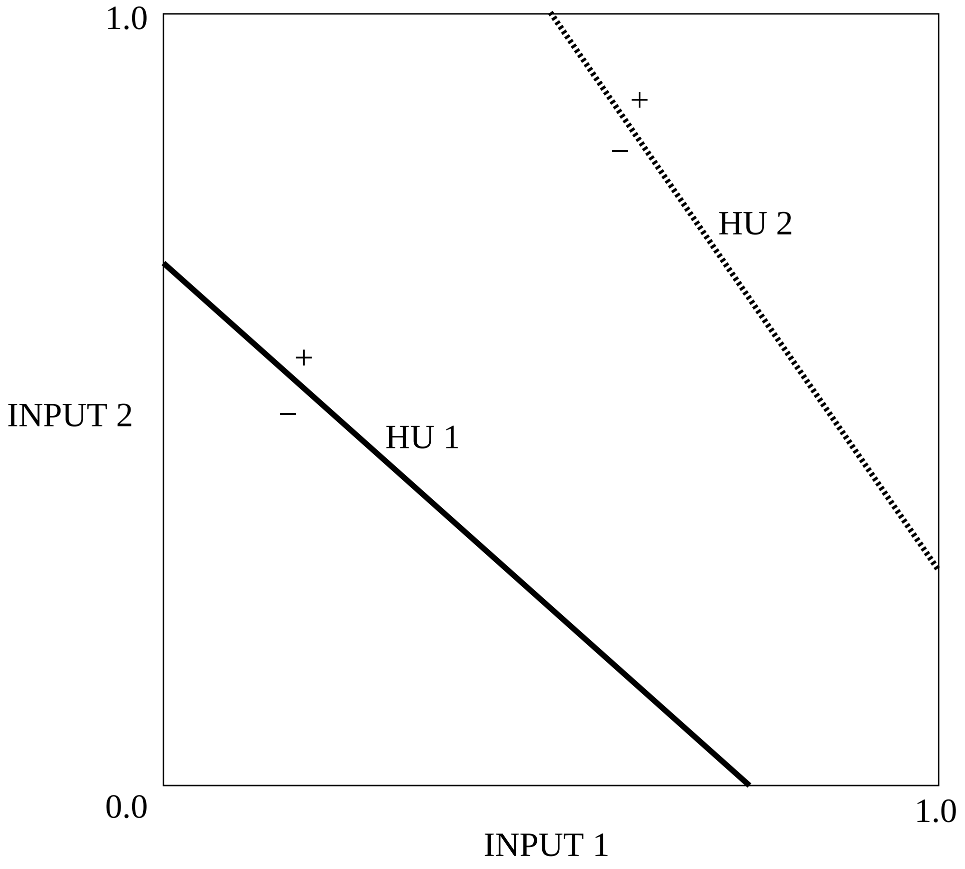


Figure 7: **A hyperplane diagram.** The diagram shows the hyperplanes defined by the two hidden units in Figure 1. HU1 refers to the leftmost hidden unit in Figure 1, HU2 refers to the rightmost hidden unit.

Units which are in the same layer of the network, and hence have the same (or nearly the same) input space, can have their hypeplanes depicted in one diagram. A *hyperplane diagram* can show how hidden units make decisions in an input space defined by input units, or it can show how output units make decisions in an input space defined by hidden units. Figure 7 shows the hyperplanes defined by the two hidden units in Figure 1. The axes of a hyperplane diagram denote the range of activations that may be propagated to the units through their incoming connections. Data points that a network is learning to classify may be plotted in the space. Each hidden unit of the network is represented by a hyperplane (or in this case a line) which indicates how the unit is partitioning its input space. The learning process is animated by showing the movement of the hyperplanes as the weights and biases of the network are changed.

It is important to note that a unit with a continuous transfer function (such as a sigmoid) does not neatly partition its input space into two regions. Instead of a sharp dividing hyperplane, there is a gradual boundary. In many cases, however, a

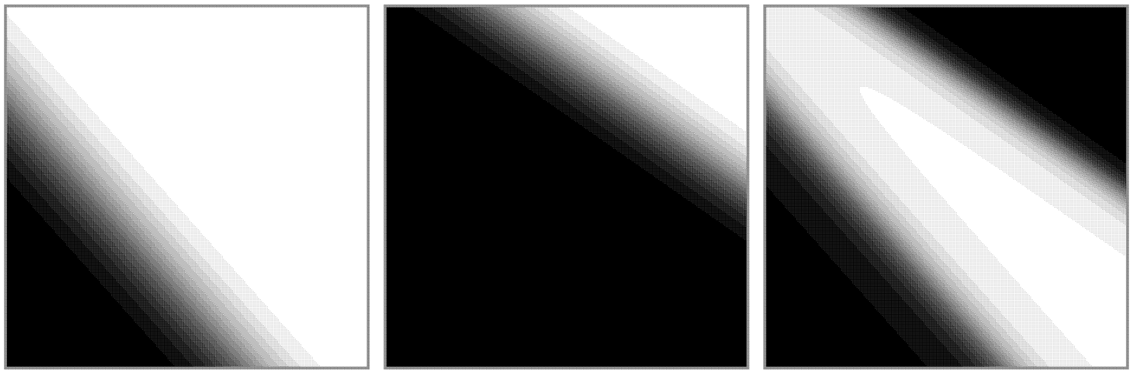


Figure 8: **Response-function plots.** The plots show the response of, from left to right, the leftmost hidden unit in Figure 1, the rightmost hidden unit, and the output unit. The vertical axes for all three plots represent the activation of INPUT 1; the horizontal axes represent the activation of INPUT 2. Lighter shades represent higher activation values than darker shades. Note that the output unit is highly active when either INPUT 1 or INPUT 2, but not both, is active.

hyperplane is a close approximation to the gradual boundary.

A fundamental limitation of the hyperplane technique is that only a two-dimensional or three-dimensional input space can be depicted. When the actual input space of a set of units has higher dimensionality, two-dimensional or three-dimensional projections of the space can be selected for viewing. Still, there is the problem of choosing which projections to view: there is an infinite number of *N*-dimensional projections of a space of dimensionality *N+1*. Furthermore, even if informative projections are selected, it is still problematic to mentally put the individual projections together to gain a unified picture of the input space. Statistical techniques, such as principal components analysis (Dennis91; Elman89) or canonical discriminant analysis (Dennis91; Wiles90) may be useful in choosing the projections that will provide the most information. These techniques are briefly described in Section 7.

5.4 Response-Function Plots

A technique which is similar to the hyperplane diagram is the *response-function plot* (Lang88). Like the hyperplane diagram, the response-function plot depicts the decision surfaces formed by hidden and output units. Unlike the hyperplane diagram, however, this technique does not require that the transfer function be approximated by a threshold function. Another difference between the two methods is that each response-function plot depicts the decision surface at only a single unit. The axes of the plot represent the range of activations for two of the network's inputs. The plot for each unit uses shading (or color) to indicate the resultant activations for each combination of input values.

Figure 8 shows response-function plots for the network shown in Figure 1. The plots display the activations of the two hidden units and the single output unit as a function of the activations that the two input units can assume. Of course it is not necessary to generate the plots for all layers of the network in terms of input-level activations. It could also be useful to generate a plot for a hidden unit in terms of inputs from the layer immediately below it. This is a general issue in ANN visualization: should (does) a given visualization technique describe things in terms of features from the previous level, or in terms of input-level features?

Lang and Witbrock show the plots the hidden and output units of a network that learns to distinguish two intertwined spirals in a two-dimensional plane. Since they are dealing with two-dimensional spatial data, the plots provide a very lucid description of the patterns being captured by each unit in the network.

A particularly nice aspect of the response-function plot is that it illustrates the gradation of activation values that result from using a continuous transfer function. The user can clearly see how close the hyperplane approximation is for a range of input values.

The primary limitation of the response-function plot is that it is able to display activation values for a unit only over a range of two or three inputs. Just as with hyperplane diagrams, to use this technique in high-dimensional spaces, it is necessary to take two-dimensional or three-dimensional projections. In the case of response-function plots, this means "clamping" the activation of some inputs and allowing only two or three inputs to vary.

5.5 Trajectory Diagrams

Another visualization method developed by Wejchert and Tesauro is the *trajectory diagram* (Wejchert90). This diagram shows the trajectory of a unit through its weight space, and the error of the unit along the trajectory. In order to fully understand this method, we first need to revisit our discussion of backpropagation.

The usual function performed by the backpropagation algorithm is to move the weights of a network toward a state in which the value of the error function (over all of the training examples) is at a local minimum. A network with a given set of weights can be thought of as a point in a multidimensional space where each dimension is defined by one of the connections in the network. The coordinates of the point are specified by the value of the weight on each connection (dimension); this is a *weight space*. If we now think of a space which has one more dimension than the weight space, we can conceptualize an *error hypersurface*; the hypersurface defined over each point in the weight space. The value of each point in the hypersurface is the error value that would result from a network with the weights given by the coordinate in the weight space. The backpropagation algorithm performs gradient descent in this weight space. In other words, when the network is in a given state (i.e., at a given point in the weight space), backpropagation changes the weights so that the

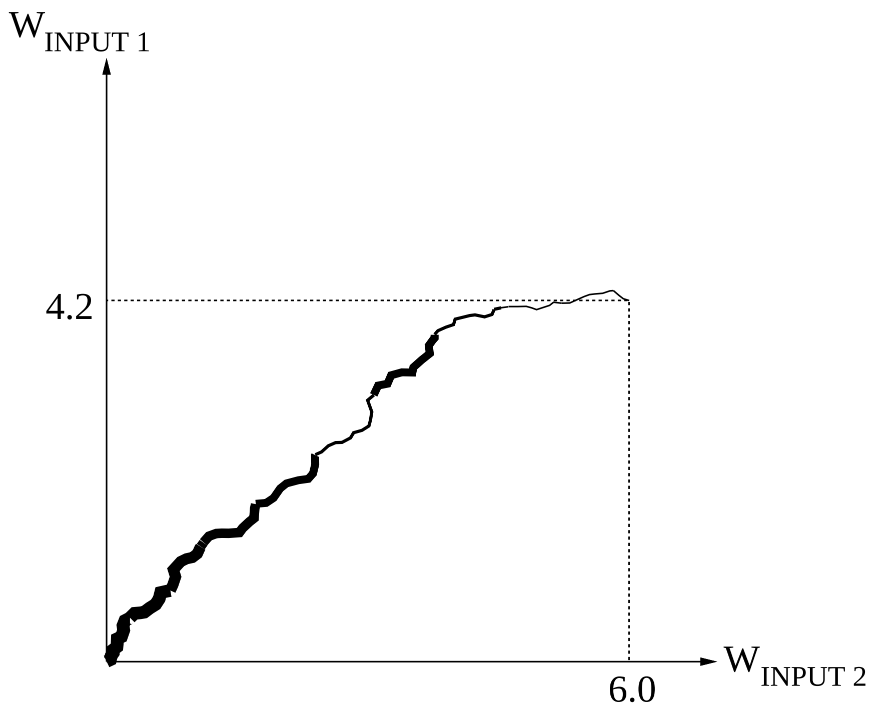


Figure 9: **A trajectory diagram.** The diagram depicts the trajectory, over a hypothetical training sesssion, of the rightmost hidden unit in Figure 1. The trajectory is plotted in the space defined by the two weights impinging on this hidden unit. The thickness of the trajectory line indicates the network error along the trajectory.

movement in weight space is in the direction in which the error is decreasing the fastest.

The trajectory diagram is designed to provide some insight into the nature of the weight space for a given problem. A trajectory diagram depicts the movement of a given unit through its weight space. Figure 9 shows a trajectory diagram for the rightmost hidden unit in Figure 1. The axes of the trajectory diagram are defined by the incoming weights to a unit; this is the weight space of the unit. A network unit at a given point in time is plotted as a point in the diagram, where the coordinates of the point are specified by the values of the weights feeding into the unit. As learning progresses, the point is replotted to reflect the updated values of its incoming weights.

Additionally, the color of each point in the trajectory diagram shows the error of the network during learning. A common technique in scientific visualization is to simultaneously depict several aspects of a problem by using a different graphical attribute for each aspect. By relating several different parameters in one representation, the relationships among the parameters are made more apparent. This technique is employed in the trajectory diagram, where color is used to indicate error values and position is used to indicate weight values. The motivation for incorporating color into the trajectory diagram is that the color values along the trajectory provide some indication of the contours present in the error hypersurface over the weight space. That is, the color values indicate the steepness of the error hypersurface at various points in the weight space, as well as the presence of local minima in the surface. [2]

As with the hyperplane technique, an inherent weakness of the trajectory diagram is the inability to visualize high-dimensional weight spaces. Wejchert and Tesauro provide some examples where they radially project [3] the axes of a six-dimensional weight space onto a two-dimensional plane. The problem with this approach is that the resultant projections of six-dimensional points are not unique. It is not clear what kind of useful information can be inferred from such diagrams. Furthermore, many networks for real-world problems have units with tens or hundreds of incoming connections. Radial projections are simply not feasible in these cases.

Another problem with the trajectory diagram arises in trying to interpret the error values of a trajectory. The error of the network is a function of all of the weights in a network, but a trajectory diagram represents error as a function of only some of the weights. Therefore error values along a given trajectory for a unit might be significantly different depending upon the trajectories of the other units in the network.

6 Visualizing Knowledge-Based Neural Networks

At the University of Wisconsin, we have developed a graphical interface for visualizing neural networks. We are using this tool to support our research in *knowledge-based* neural networks. In this section we first describe the KBANN algorithm for initializing knowledge-based neural networks. We then describe the functionality provided by our visualization tool, and discuss how we have used it to gain a better understanding of knowledge-based neural networks.

We have successfully applied knowledge-based neural networks to a several real-world problems including: finding promoters in DNA (Towell90), determining splice-junctions in DNA (Noordewier91), predicting the secondary structure of proteins (Maclin91), and refining process controllers (Scott91). In this section, however, we will discuss knowledge-based neural networks and our visualization system in the context of a "toy" problem – learning to recognize cups.

[2]Although the gradient descent process does not itself avoid local minima, the backpropagation algorithm is commonly augmented with a *momentum* term which helps the network to avoid them.

[3]A radial projection of an N-dimensional space involves a space in which the N axes all lie in the same plane, and emanate from the origin such that the separations between adjacent axes are equiangular.

Table 1: **An approximately-correct domain theory for recognizing cups.**

cup	: -	stable, liftable, open-vessel.
stable	:-	flat-bottom.
liftable	:-	graspable, light.
graspable	:-	has-handle.
open-vessel	:-	has-concavity, concavity-up.

6.1 Knowledge-based Neural Networks

Although neural network learning has proven to be an effective and general method, it suffers from several weaknesses. One weakness is that conventional neural networks do not provide a way to exploit existing knowledge about the problem to be solved. The KBANN algorithm (Towell90) provides an approach to incorporating existing knowledge into a neural network.

The KBANN algorithm uses a knowledge base of domain-specific inference rules (a *domain theory*), in the form of PROLOG-like clauses, to determine the topology and initial weights of a neural network. The domain theory need be neither complete nor correct; it need only support approximately-correct reasoning. KBANN translates a domain theory into a neural network in which units and links correspond to parts of the domain theory.

A brief explanation of the procedure used by KBANN to translate rules into an ANN follows; a detailed description can be found in (Towell90).

Consider the Prolog rules in Table 6.1, which define a roughly-correct domain theory for recognizing cups. Figure 10 depicts the hierarchical structure of this domain theory. The hierarchical structure of the domain theory determines the topology of the knowledge-based neural network generated by KBANN: the input units of the network represent the base-level facts of the domain theory, hidden units represent intermediate conclusions, and the output unit represents the final conclusion (whether or not an object is a cup). The units representing the antecedents of a rule are connected to the unit representing the consequent by heavily-weighted connections. Lightly-weighted connections are added to the network to facilitate refinement of the domain theory. Additional input units may be added to the network to incorporate features which do not appear in the domain theory, but nevertheless may be relevant.

The KBANN algorithm sets link weights and unit biases so that units have significant activation only when the corresponding deduction could be made using the knowledge base. For example, assume there exists a rule in the knowledge base with n positive antecedents (i.e., antecedents which must be true in order to derive the consequent using this rule), and m negative antecedents (i.e., antecedents which must not be true to derive the consequent using this rule). KBANN sets the weights on links corresponding to positive and negative antecedents to ω and $-\omega$, respectively. The bias on the unit corresponding to the rule's consequent is set to $(n - 1/2) * \omega$.

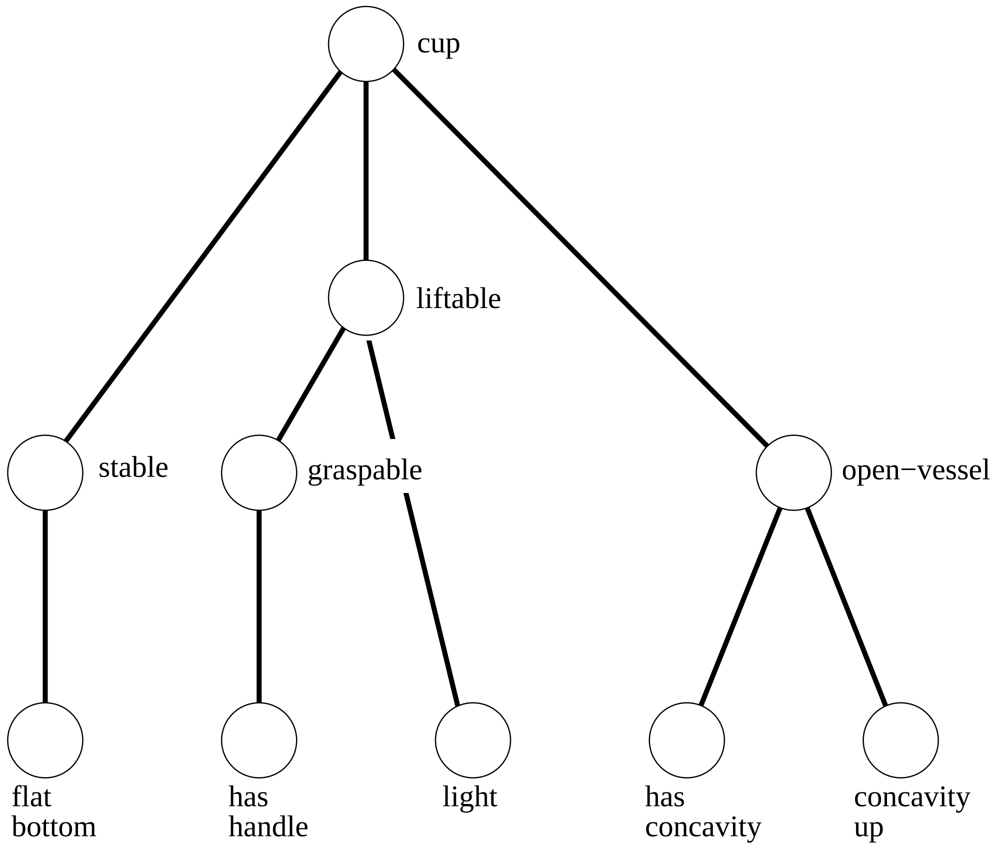


Figure 10: **The hierarchical structure of the cup domain theory.** This structure is mapped to a neural network by the KBANN algorithm.

To handle disjunctions (i.e., multiple rules with the same consequent), KBANN maps each disjunct separately as described above, and then introduces another unit to represent the disjunction. The links which connect the units representing the disjuncts to the unit representing the disjunction are given a weight of ω, and the bias of the disjunction unit is set to $\omega - 1/2$. The disjuncts cannot be represented by a single unit for their shared consequent because there is no way to set the bias of the consequent unit such that unintended combinations of antecedents cannot make the unit significantly active.

After the network topology and initial weights have been determined by KBANN, the network is trained using the backpropagation algorithm and a set of training examples. The effect of this training phase is to refine the approximately correct domain theory so that it is consistent with empirical evidence (i.e., , the training examples). After training, refined rules can be extracted from the network (Towell91).

6.2 *Lascaux* : A Tool for Visualizing Neural Networks

In order to support our research in knowledge-based neural networks, we have developed a neural network visualization tool, called *Lascaux*, [4] that enables us to view certain aspects of networks both during and after learning. In this section, we describe some of the capabilities of *Lascaux*.

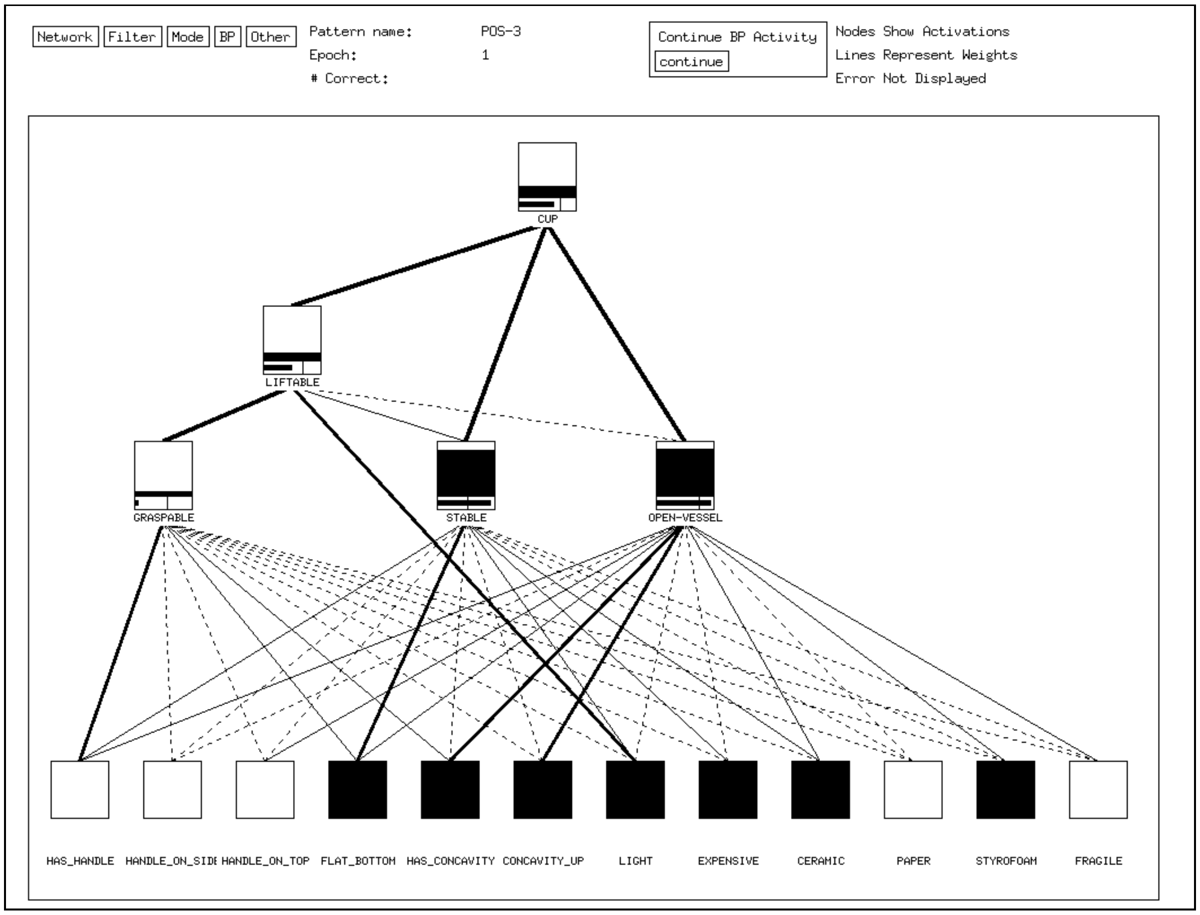
6.2.1 Depicting A Network

Figure 11 shows the interface provided by *Lascaux*. Like Wejchert and Tesauro's bond diagram, the interface clearly displays the topology of the network. Each network unit is represented by a box. A label below each hidden and output unit indicates the concept putatively represented by the unit (if the putative concept is not known, then the label may simply be something like "hidden5"). Labels below the input units indicate the problem-domain features that they measure. Network weights are represented by lines which connect the units. The thickness of each line indicates its magnitude. Positive weights are drawn as solid lines and negative weights are drawn as dashed lines. In Figure 11 it can be seen that the weights that are specified by the domain theory are drawn as thick lines, whereas the lightly-weighted links that are added to the network to facilitate refinement are drawn as thin lines. Some of these added weights are positive and hence drawn as solid lines, while some are negative and are drawn as dashed lines.

6.2.2 Visualizing Learning

Lascaux enables visualization of the learning process by depicting the forward propagation of activations, the backward propagation of error, and changes to the weights and biases of the network. The activation of each unit is illustrated by a thermometer-like display. In Figure 11 these *activation meters* occupy the top portion of each box representing a hidden or output unit, and the entirety of each box representing an input unit. The level to which a meter is filled with black indicates the activation at the corresponding unit. A unit that is completely active (i.e., activation = 1) will be completely filled. A unit that is completely inactive (activation = 0) will not be filled at all. Thus, in Figure 11 the activation at the output unit indicates that the network has decided that the current example is not a cup. Although the STABLE and OPEN-VESSEL units have fairly high activations, the unit representing the other necessary condition for being a cup, LIFTABLE, has a small activation value. The activations of the hidden units can, in turn, be explained in terms of the activations of the input units that feed into the hidden units.

[4]Our visualization tool is named after the caves in Southern France which contain some of the earliest known paintings. The paintings at Lascaux, which depict animal and human figures as well as geometric signs, represent state-of-the-art visualization in the Paleolithic period.



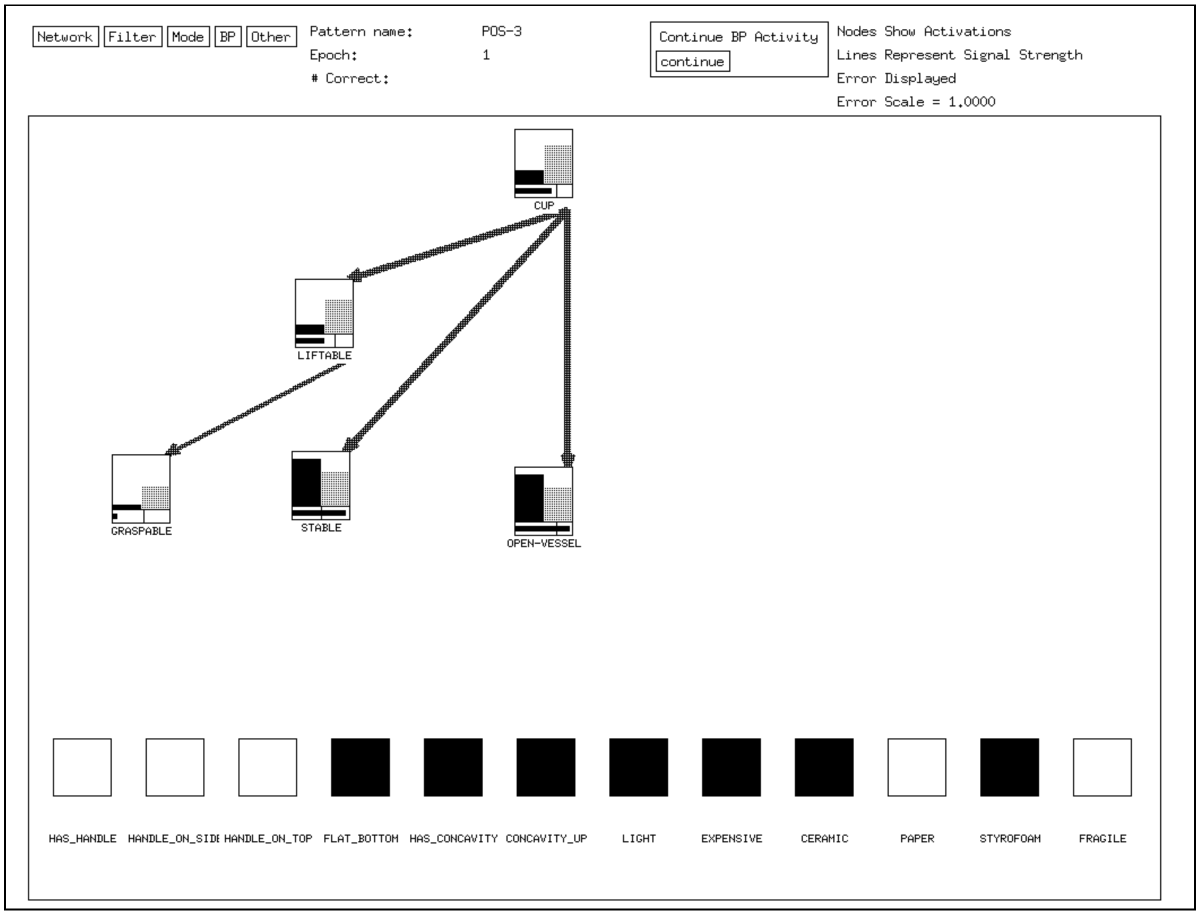
```
Pattern name:     POS-3
Epoch:            1
# Correct:
```

Nodes Show Activations
Lines Represent Weights
Error Not Displayed

Figure 11: **A knowledge-based neural network as depicted by *Lascaux*.** Units are represented by boxes and weights are represented by lines connecting the boxes. The top part of each hidden and output unit box shows the activation of the unit. The lower part of each box shows the net input (horizontal bar) relative to the "threshold" (vertical bar) of the unit. For input units, all of the box is used to show the activation.

In addition to showing the activation of each unit, *Lascaux* can display the error at each hidden or output unit for a particular pattern during learning. When error values are being shown, the activation-meter for each hidden or output unit is halved so that it occupies only the top-left portion of the boxes representing these units, and a similar thermometer-like display is located in the top-right part of each box to depict error. When the error value at a unit is positive, the error bar is drawn emanating from the bottom of the error display upward; when the error value is negative, the bar is drawn emanating downward from the top of the display. Sample error bars are shown in Figure 12. Backward-propagated error signals can be displayed as lines. As with weights, the thickness of the lines indicates the magnitude of the signals.

Lascaux includes mechanisms for filtering the information that is to be displayed.



```
Pattern name:        POS-3
Epoch:               1
# Correct:
```

```
Nodes Show Activations
Lines Represent Signal Strength
Error Displayed
Error Scale = 1.0000
```

Figure 12: **The backward-propagation of error signals as depicted by** *Lascaux***.** The error at each output and hidden unit is shown on the left side of the unit. The interface allows the available information to be filtered. For example, none of the weights or forward-propagated signals are shown here.

For example, a user-settable threshold enables the user to view only a subset of the weights in the network – weights with magnitudes less than the threshold are not displayed. During learning, weights are drawn and undrawn as their magnitudes cross this threshold. Another mechanism allows the user to select (unselect) units for which the incoming weights are to be displayed. Mechanisms such as these are important for visualization tools because they allow the user to inspect selected portions and aspects of the system of interest, without being overwhelmed by the quantity of data available for display.

6.2.3 Visualizing Computing

In addition to displaying the activation of each unit, the interface can depict the forward propagation of *signals* from unit to unit. A signal from unit j to unit i on pattern p is defined as:

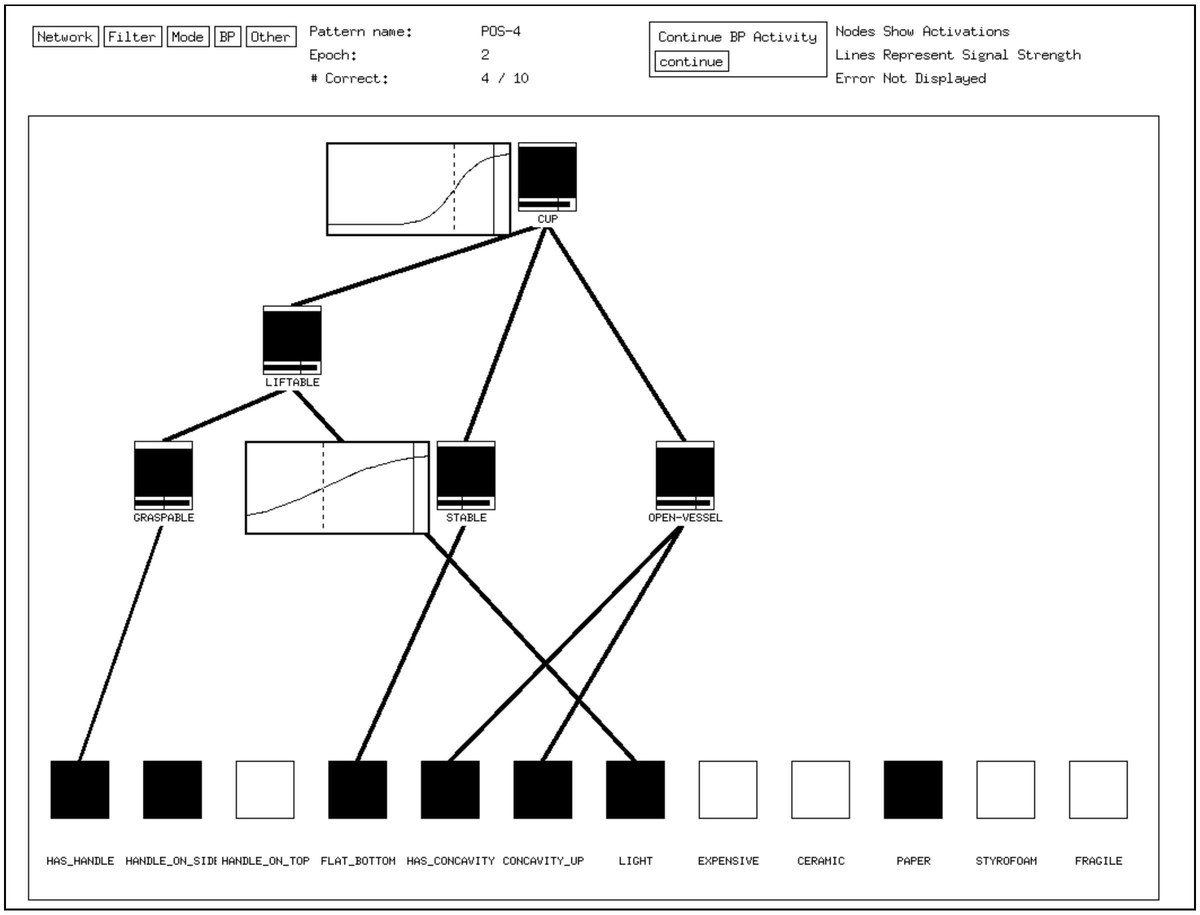
$$signal_{pij} = w_{ij} o_{pj} \tag{7}$$

The activation signals are viewed by changing the mode of the interface. In the *signal mode*, the lines connecting units represent activation signals, instead of weights. As with the weight mode, the thickness of a line indicates the magnitude of the signal. The value of the signal mode is that it can provide an explanation for a particular training example. The activation of each unit in the network can be explained by noting the signals that are being propagated to the unit. This explanation capability can be seen in Figure 13.

Another feature of the interface which assists in understanding the propagation of activations is a diagram that relates the net input and activation of a unit (i.e., it depicts the activation function). This diagram is shown for both a hidden unit and the output unit in Figure 13. The diagram plots the activation function for a unit on a scale that is defined by the range of net input values that the unit could have. Thus, the rightmost edge of the diagram shows the activation value that would result if the unit were to receive its maximum net input. The leftmost edge shows the activation that would result if the unit were to receive its minimum net input. The diagram can be thought of as plotting the *effective activation function.* The actual net input that results for a given pattern is displayed as a solid vertical line in the diagram. The dashed line marks the point where the net input is 0.

The *effective activation diagram* is valuable for two reasons. First, it describes the nature of the activation function relative to its weight space. The degree of nonlinearity and the likelihood of various activation values for a unit are made apparent. When the weights impinging on a unit are small, the activation function is relatively linear, whereas when the weights are large, it approximates a threshold function. Second, the relative influence of the weights and the bias can be determined from the diagram. When the bias is large, the unit takes much more net input to "turn on", and likewise when the bias is small it takes much less net input. Below the activation meter of each hidden and output unit there is a compact version of this diagram. This compact version is also scaled by the range of possible net input values for a given unit. The vertical line represents the value of the bias. The horizontal line shows the actual net input for a given pattern. Thus, when the net input exceeds the bias, the horizontal line will cross the vertical line.

In order to control the rate at which information is displayed by *Lascaux* there is a mechanism that enables the user to specify when to "freeze" the display. This mechanism is a threshold against which the activations of the output units are compared. Whenever, the activation of one of the output units is equal to or exceeds this thresh-


Pattern name: POS-4
Epoch: 2
Correct: 4 / 10

Nodes Show Activations
Lines Represent Signal Strength
Error Not Displayed

Figure 13: **Activation signals and** *effective activation functions* **as depicted by** ***Lascaux*.** The lines connecting units represent forward-propagated signals. The *effective activation functions* are shown for the STABLE and CUP units.

old, the interface stops drawing newly-generated information until the user presses a "continue" button. Thus, by setting this threshold to zero, the user can single-step through a set of input patterns. When the threshold is set to a value greater than one, *Lascaux* will continuously update the display without stopping (or until the threshold is changed to a lower value).

6.2.4 Implementation

Lascaux is implemented using the X Window System (Gettys89), the Athena Widget set (Peterson89), and the C programming language. It runs in concert with a separate program which implements a neural network and the backpropagation algorithm (McClelland89). The *Lascaux* process and the network process communicate via UNIX sockets. The visualization process sends user-selected commands to the backpropagation process. The backpropagation process sends back such information as activations and weights, sending information as fast as it generates it. The user controls, through the interface, the rate at which the information is displayed. By separating the network implementation from the interface, we were able to keep *Lascaux* highly responsive to user interaction without commingling user-event handling code with the backpropagation routines. Additionally, this architecture enables *Lascaux* to be extended to maintain simultaneous connections to more than one network process.

6.3 Visualizing Knowledge-Based Neural Networks

Although *Lascaux* provides the same functionality whether it is used with conventional ANNs or with knowledge-based neural networks, we believe that there is a synergistic relationship that arises between knowledge-based neural networks and our visualization techniques. This relationship makes it much more practicable to understand the classifications made by trained networks, and to understand the changes to the networks that occur during learning. With a knowledge-based neural network, the network is initially in a comprehensible state; the graph-like structure inherent in a domain theory is mapped into a neural network. Therefore, when *Lascaux* depicts a network before learning, it is essentially depicting the domain theory in a graph-like representation. Each of the units corresponds to an antecedent or a consequent in one or more of the rules of the domain theory. The heavily-weighted links indicate how the antecedents relate to the consequents.

The process of learning in a knowledge-based neural network is essentially a process of theory refinement. The initial domain theory is adjusted to account for the training data. *Lascaux* enables us to understand the refinements that occur during learning by animating the weight changes. If the revision process results in a particular antecedent of a rule gaining (losing) importance, then the weight representing the antecedent's role in a rule will increase (decrease) and this change is made visible by *Lascaux*. By setting the weight visibility threshold to an appropriate value, the refinement process can even be visualized as the addition and deletion of antecedents.

Another valuable result of the synergy between knowledge-based neural network and visualization is that the activation patterns shown by *Lascaux* can provide an explanation of why the network has made a particular decision. This is illustrated in Figure 13. An explanation is described by a combination of unit activations and forward-propagated signals. The explanation can be elicited by tracing signals from the output unit of interest back to the hidden units that propagated these signals. The activation of each hidden unit indicates the degree to which the concept represented by the hidden unit was satisfied. An explanation for each hidden unit can be constructed in the same way, by following forward-propagated signals backward. Relatively small signals can be filtered from the display so that only the salient parts of an explanation are recovered.

7 Other Approaches to Understanding ANNs

There are several varied approaches to understanding the representations formed in ANNs in addition to visualization. They include: statistical analysis; interpretation of ANNs as finite-state automata; and extraction of symbolic rules. These approaches have different strengths, weaknesses, and ranges of applicability. The task of understanding ANNs is sufficiently difficult that no single approach is usually adequate. In our research, we have found a number of these approaches, in addition to visualization, to be useful in understanding networks. We provide an overview of the approaches below.

A variety of statistical techniques can provide insight into how a neural network classifies its input. Hierarchical cluster analysis (Elman89; Sejnowski87), principal components analysis (Dennis91; Elman89), and canonical discriminant analysis (Dennis91; Wiles90) have been employed to gain insight into the classifications made by ANNs. All of these methods take a set of data points as input. In order to understand the discriminations made by hidden units, each data point can be a vector of hidden unit activations for a particular training example. Hierarchical clustering involves constructing a tree that relates vectors of hidden unit activations. Similar vectors are closely related in the tree, while dissimilar vectors are distantly related. Each leaf of the tree is labelled with the name of the input pattern that generated the vector represented by the leaf. The value of this approach is that it provides some sense of the discriminations being performed by the hidden units. Principal components analysis (PCA) calculates the major directions of variance for the given data points. Canonical discriminant analysis (CDA) requires that the category of each data point be supplied along with the values defining the point. CDA then finds the directions along which the points within a group are tightly clustered. Both PCA and CDA can be used to derive a useful low-dimensional projection of a high-dimensional input space.

Another approach to understanding ANNs involves interpreting networks as finite-state automata (Giles90). This approach is applicable to a class of networks, called *recurrent networks* (Jordan86; Williams89), which have connections from output units to input units. These connections serve to supply the previous state of the network, as expressed by the output units, as an input to be used in computing the next state of the network.

A recent approach to understanding ANNs is to translate the representations formed by a network into symbolic rules (Fu91; Towell91). A decided advantage of many symbolic machine learning algorithms is that the representations they form are often humanly-comprehensible. Artificial neural networks, on the other hand, are better at learning some problems, but the representations they form are impenetrable. With a rule-translation mechanism, however, one can employ the learning power of an ANN, and in the end still have a comprehensible representation for the knowledge learned. The challenge of this approach is to extract rules which are comprehensible while still accurately modelling the network.

8 Conclusions

In this article, we have described a number of visualization techniques that have been applied to the problem of understanding neural networks. Visualization can provide insight into the workings of a network by transforming the parameters into more easily understood visual representations. We have described our system for network visualization and its application to knowledge-based neural networks. When applied to knowledge-based neural networks, visualization techniques can help to understand how neural learning refines a domain theory, and can assist in providing explanations of the decisions made by a network.

Although visualization techniques can help to clarify the workings of neural networks, the problem of visually inspecting a network is still problematic. One of the most difficult aspects of visualizing neural networks is the high-dimensionality of the spaces that need to be understood. One of the challenges of network visualization work in the future is to develop methods that are able to succinctly compress these high-dimensional spaces into easily understood representations.

Another important issue for visualization tools is user control over the data to be visualized. There are at least three aspects of user control that should be incorporated into visualization tools. First, the user should be able to change important network parameters and immediately see the effects of such a change. Second, the user should be able to perform projections on and select portions of the available data in order to view selected aspects of a network at a time. Third, the user should be able to control the rate at which information is depicted.

Although artificial neural networks have demonstrated impressive performance in a wide variety of machine learning tasks, they possess a significant limitation in that the representations they form are difficult to understand. An important area for research is the development of techniques to aid in the comprehension of neural networks. Visualization has proven to be one such technique; it has been useful for understanding both the dynamics of neural networks, and the representations they form. There are still many open problems in neural network visualization, however, and it should continue be an important field of study.

9 Acknowledgements

Lori Pratt and Abbey Dy of Rutgers University provided us with hyperplane visualization code, and also a helpful description of their work in this area. Derek Zahn and Andy Whitsitt assisted in the development of *Lascaux*. Rich Maclin contributed many valuable ideas to the *Lascaux* effort. The work of our research group has been partially supported by Office of Naval Research Grant N00014-90-J-1941, National Science Foundation Grant IRI-9002413, and Department of Energy Grant DE-FG02-91ER61129.

References

Atlas, L., Cole, R., Connor, J., El-Sharkawi, M., Marks II, R. J., Muthusamy, Y., and Barnard, E. (1989). Performance comparisons between backpropagation networks and classification trees on three real-world applications. In *Advances in Neural Information Processing Systems*, volume 2, pages 622–629, Denver, CO. Morgan Kaufmann.

Barnard, E. and Cole, R. A. (1989). A neural-net training program based on conjugate-gradient optimization. Technical Report CSE 89-014, Oregon Graduate Center.

Brown, M. H., DeFanti, T. A., and McCormick, B. H. (1987). Visualization in scientific computing. *Computer Graphics*, 21(6).

DeFanti, T. A., Brown, M. D., and McCormick, B. H. (1989). Visualization: Expanding scientific and engineering research opportunities. *Computer*, 22:12–25.

Dennis, S. and Phillips, S. (1991). Analysis tools for neural networks. Technical Report 207, Department of Computer Science, University of Queensland, Queensland, Australia.

Elman, J. L. (1989). Representation and structure in connectionist models. Technical Report 8903, Center for Research in Language, University of California, San Diego.

Fisher, D. H. and McKusick, K. B. (1989). An empirical comparison of ID3 and back-propagation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 788–793, Detroit, MI.

Fu, L. (1991). Rule learning by searching on adapted nets. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 590–595, Anaheim, CA.

Gettys, J., Scheifler, R. W., and Newman, R. (1989). Xlib – C language X interface, MIT X consortium standard. Technical report, Massachusetts Institute of Technology.

Giles, C. L., Sun, G. Z., Chen, H. H., Lee, Y. C., and Chen, D. (1990). Higher order recurrent networks and grammatical inference. In *Advances in Neural Information Processing Systems*, volume 2, pages 380–387, San Mateo, CA. Morgan Kaufmann.

Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the Theory of Neural Computation*. Addison Wesley, Redwood City, CA.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 77–109. MIT Press, Cambridge, MA.

Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Conference of the Cognitive Science Society*, pages 531–546, Amherst, MA.

Knight, K. (1990). Connectionist ideas and algorithms. *Communications of the ACM*, 33(11):59–74.

Lang, K. J. and Witbrock, M. J. (1988). Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 52–59. Morgan Kaufmann.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, volume 2, pages 396–404, San Mateo, CA. Morgan Kaufmann.

Maclin, R. and Shavlik, J. W. (1991). Refining domain theories expressed as finite-state automata. In *Machine Learning: Proceedings of the Seventh International Workshop*, pages 524–528, Evanston, IL. Morgan Kaufmann.

McClelland, J. L. and Rumelhart, D. E. (1989). *Explorations in Parallel Distributed Processing*. MIT Press, Cambridge, MA.

Minsky, M. and Papert, S. (1969). *Perceptrons*. MIT Press, Cambridge, MA.

Moody, J. and Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294.

Munro, P. (1991). Visualizations of 2-D hidden unit space. Technical Report LIS035/IS91003, School of Library and Information Science, University of Pittsburgh, Pittsburgh, PA.

Noordewier, M. O., Towell, G. G., and Shavlik, J. W. (1991). Training knowledge-based neural networks to recognize genes in DNA sequences. In *Advances in Neural Information Processing Systems*, volume 3, pages 530–536, San Mateo, CA. Morgan Kaufmann.

Peterson, C. (1989). Athena widget set – C language X interface. Technical report, MIT X Consortium.

Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, volume 1, pages 305–313, San Mateo, CA. Morgan Kaufmann.

Pratt, L. Y. and Mostow, J. (1991). Direct transfer of learned information among neural networks. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 584–589, Anaheim, CA.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–363. MIT Press, Cambridge, MA.

Scott, G. M., Shavlik, J. W., and Ray, W. H. (1991). Refining PID controllers using neural networks. In *Advances in Neural Information Processing Systems*, volume 4, Denver, CO. Morgan Kaufmann.

Sejnowski, T. and Rosenberg, C. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168.

Shavlik, J. W., Mooney, R. J., and Towell, G. G. (1991). Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143.

Tesauro, G. and Sejnowski, T. J. (1989). A parallel network that learns to play backgammon. *Artificial Intelligence*, 39(3):357–390.

Towell, G. G., Craven, M. W., and Shavlik, J. W. (1991). Constructive induction in knowledge-based neural networks. In *Machine Learning: Proceedings of the Seventh International Workshop*, pages 213–217, Evanston, IL. Morgan Kaufmann.

Towell, G. G., Shavlik, J. W., and Noordewier, M. O. (1990). Refinement of approximately correct domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866, Boston, MA. MIT Press.

Tufte, E. R. (1983). *The Visual Display of Quantitative Information.* Princeton University: Graphics Press, Princeton, NJ.

Weiss, S. M. and Kapouleas, I. (1989). An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 688–693, Detroit, MI.

Wejchert, J. and Tesauro, G. (1990). Neural network visualization. In *Advances in Neural Information Processing Systems*, volume 2, pages 465–472, San Mateo, CA. Morgan Kaufmann.

Wiles, J., Stewart, J., and Bloesch, A. (1990). Patterns of activations are operators in recurrent networks. Technical Report 189, Department of Computer Science, University of Queensland, Queensland, Australia.

Williams, R. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.