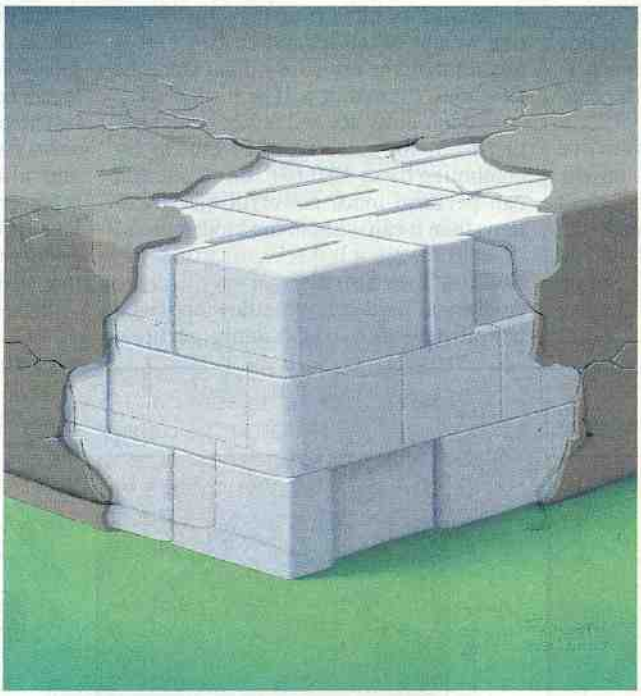
What's Hidden in the Hidden Layers?

The contents can be easy to find with a geometrical problem, but the hidden layers have yet to give up all their secrets

David S. Touretzky and Dean A. Pomerleau



Much of the current fascination with neural networks has to do with their ability to learn. The most popular learning algorithm today is back-propagation, which can be implemented rather easily on a microcomputer (see "Back-Propagation," October 1987 BYTE).

To solve a problem with a back-propagation network, you show it sample inputs with the desired outputs, over and over, while the network learns by adjusting its weights. If it solves the problem, it will have found a set of weights that produce the correct output for every input.

But what has the network learned? Unlike an expert system, neural networks do not automatically explain their reasoning. Whatever knowledge the network acquires is encoded in its numerical weights. It's not easy to decipher the network's solution to a problem when all you have to look at is a set of floating-point numbers.

In the past, the difficulty in interpreting weight patterns contributed to the neural-network mystique. Networks were sometimes billed as magic boxes whose learning algorithms produced solutions unintelligible to mere humans.

Today, we have a better understanding of neural-network learning procedures like back-propagation, and we can analyze, to some extent, the representations that develop. Back-propagation consists of two passes. In the forward pass, inputs proceed through the network and generate a certain output. Then, in the backward pass, the difference between the actual and desired outputs generates an error signal that is propagated back through the network to teach it to come closer to producing the desired output.

Between the input and output layers, there may be additional layers of units, called *hidden units*. When analyzing a network, we study two kinds of hidden-unit representations. First, we want to understand what the weights mean. Second, we want to look at the patterns of activation of units in the hidden layer in response to particular inputs.

Hidden units should really be called "learned-feature detectors" or "re-representation units," because the activity pattern in the hidden layer is an encoding of what the network thinks are the significant features of the input. The two representations (weights and activity patterns) are closely related, but, for some problems, looking at one is more informative than looking at the other.

To understand the hidden-layer representations that real networks develop, look at two examples of geometric problems that have recently been solved by back-propagation. The first is a highly

continued

nonlinear binary classification problem; the second involves driving a robotic vehicle along a road through a park.

The Unit Square

In two-layer networks, input units connect directly to output units, and each connection has a number, or weight, attached to it. One widely known limitation of these networks is that they cannot compute the XOR function. Introducing a third, hidden layer of units between the input and output layers provides the necessary computational power for XOR.

You can view XOR as a special case of a more general problem: classifying points in the *unit square* (as in figures 1a and 1b). Each point in the unit square is either in class 0 or class 1. In the case of XOR, you consider only the four corners of the square: the points (0,0), (0,1), (1,0), and (1,1). The first and fourth points are in class 0 (0 XOR 0 = 0, and 1 XOR 1 = 0); the second and third are in class 1 (0 XOR 1 = 1, and 1 XOR 0 = 1).

A single artificial neuron computes a linear sum of its inputs and produces either a 0 or a 1 as output. This in effect draws a line that partitions this square into two regions. For all points on one side of the line, the neuron outputs a 1; for all points on the other side, the neuron outputs a 0.

The position and orientation of the line are determined by the weights on the neuron's input connections. You can't draw a single straight line through the unit square so that (0,1) and (1,0) end up in one region and (0,0) and (1,1) end up in the other. Therefore, you can't solve XOR with a two-layer network.

Introducing a layer of hidden units increases the power of the network, since each hidden unit can partition the input space in a different way. The output unit then computes a linear combination of these partitionings to solve the problem.

In the XOR example, a hidden layer containing two units is adequate (see figure 1c). The first unit partitions the space so that it is activated when either input, (0,1) or (1,0), or both, (1,1), are active, as in figure 1a. It has an *excitatory* connection (a positive weight) to the output unit. The network sets the second hidden unit's weights so that it becomes active only when both inputs, (1,1), are active, as in figure 1b. It has a stronger *inhibitory* (negative) influence on the output unit than the excitatory influence of the first hidden unit.

This network correctly solves the XOR problem. When neither input is active, (0,0), neither hidden unit is active, so the output unit remains off. When a single input unit is on, (0,1) or (1,0), the first hidden unit turns on, activating the output unit. If both input units are active, (1,1), both hidden units turn on. Since the inhibitory input from the larger negative weight of the second hidden unit is greater than the excitatory input from the first, the output unit will be turned off.

Hidden units act as feature detectors, or *filters*, for some types of inputs. By combining these features, the output unit can perform more powerful classifications than it can without the hidden units.

Solving Two Spirals

Additional hidden layers allow artificial neural networks to efficiently partition the input space into arbitrary regions and perform complex tasks. One such task is the two-spirals problem, originally posed by Alexis Wieland of the Mitre Corp. in Cambridge, Massachusetts. In this problem, the network must distinguish between points on two intertwined spirals in the unit square (see figure 2).

The black dots are all in class 0, the white dots in class 1. Like XOR, this problem is not linearly separable. There is no way to draw a single straight line through the unit square so that all the black dots end up on one side and all the white dots on the other.

Two of our colleagues at Carnegie Mellon University, Kevin Lang and Michael Witbrock, recently taught a neural network to solve the two-spirals problem and analyzed the hidden-layer representations that developed (see reference 1). Their network, shown in figure 3, has two input units, representing the x and y coordinates of the point, and one output unit. The activation levels of the input units are not restricted to binary values, but they can take on any value between 0.0 and 1.0.

This network has two hidden layers of five units each. The units in each layer receive connections from the units in all layers below it. The connections that skip layers provide direct information pathways from lower layers in the network and allow more flexible hidden-layer representations. Unlike the XOR problem, however, it's not obvious what a good set of hidden-layer feature detectors would look like for this task.

Back-propagation develops the feature

continued

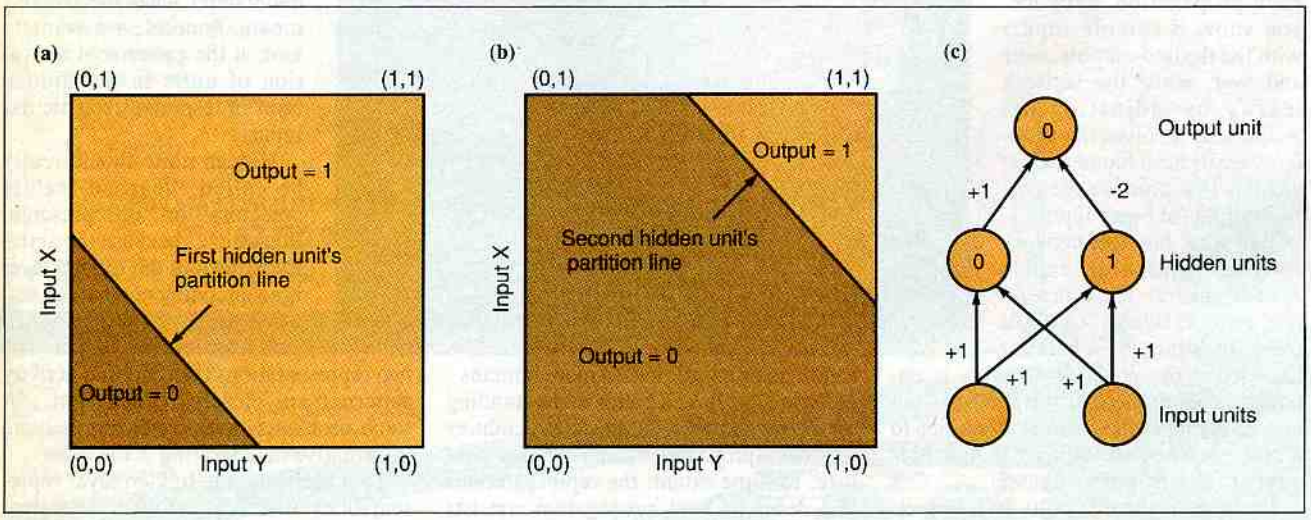


Figure 1: *A network designed to solve the XOR problem.* **(a)** *and* **(b)** *The regions of input space for which the two hidden units are active.* **(c)** *The number inside each unit is its threshold. A unit turns on when its total input exceeds its threshold. The total input is equal to the sum of its input values (each input multiplied by the weight on the line).*

detectors in figure 4. Each square in the figure graphs the response of a single unit to points at various positions in the interior of the unit square. (The squares in figure 4 correspond directly to the circles in the same positions in figure 3.) The brightness of each point in a square indicates the activation level of that hidden unit when the network is shown an input point at that position.

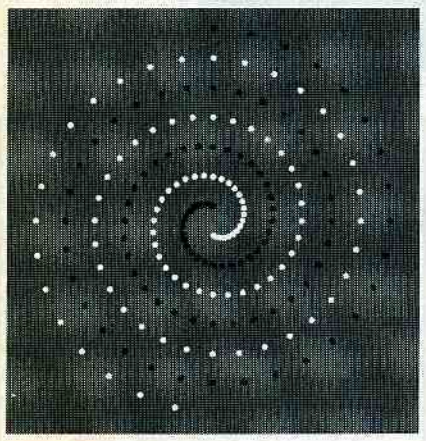


Figure 2: *Training points for the two-spirals problem. Black points should produce an output of 0; white points should produce an output of 1.*

Units in the first hidden layer divide the input space into two regions along various angles. Units in the second layer use combinations of these first-layer features to produce curved response patterns. The output unit then uses these curved patterns to form successive turns of the spiral.

The imperfections of the solution are an interesting aspect of the way back-propagation works. Notice the bumps and gaps in the spirals that the output unit forms. The network learns to classify all the points in the training set in figure 2 correctly, but it is *underconstrained*: It is not told how to respond to the remaining points in the unit square. Given this kind of freedom, back-propagation almost never develops a perfect solution.

One of the most difficult parts of training neural networks is choosing the training set. You want back-propagation to develop a network that classifies patterns in the training set correctly and also generalizes to new patterns correctly. Providing additional training data and constraining the network architecture are two techniques that reduce excess freedom and clean up the network's representations.

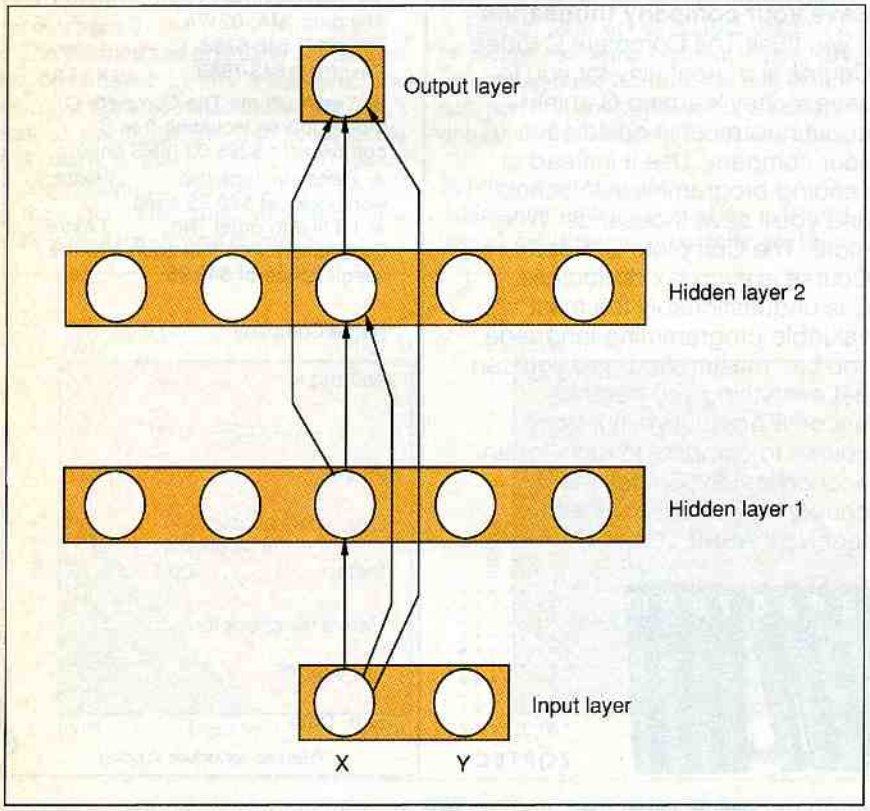


Figure 3: *Lang and Witbrock's network for learning the two-spirals problem. Each unit receives input from all the units in all the layers below it.*

A Road Tracker

Proper generalization is particularly crucial in real-world problems where you can't train a network in advance for every circumstance it might encounter in the field. One such problem we have been working on at Carnegie Mellon is autonomous vehicle navigation (see photo 1 and reference 2).

The goal of ALVINN (for autonomous land vehicle in a neural network; see reference 3) is to drive the NAVLAB vehicle along a winding road. The inputs to ALVINN are more complex than the coordinates of a single point in the unit square, but they are geometrical in nature.

ALVINN receives two types of sensor inputs from the NAVLAB (see figure 5). One is a 30- by 32-pixel image from a video camera mounted on the roof of the vehicle. (Each pixel in the video image corresponds to an input unit in the video retina.) The activation level of each unit in the video retina indicates the brightness of the corresponding pixel in the video image.

The other input is an 8- by 32-pixel image from a laser range finder. The activation levels of units in the range finder's retina represent its distance from the corresponding area in the image. The darker the color, the closer the object is. A stylized input sample is shown in figure 5. Notice that the tree to the left of the road in the video image shows up as an area of constant brightness in the range finder image. This is because the tree surface is essentially perpendicular to the horizontal range finder beam and, therefore, at a constant distance away.

The two input retinas are connected to a single layer of hidden units, which are in turn connected to the output units. (In other words, all input units are connected to all hidden units, and all hidden units are connected to all output units.) The response of the output layer is a linear representation of the direction in which the vehicle should travel to head toward the center of the road. The centermost output unit represents the "travel straight ahead" condition, while units to the left and right of center represent successively sharper left and right turns.

To drive the NAVLAB vehicle, video and range finder data from the on-board sensors are injected into the input layer. After completing a forward pass, the network reads a steering command from the output layer. The output unit with the highest output value determines the direction in which the vehicle will head.

Training the network is difficult. To develop a hidden-layer representation that generalizes correctly to new situa-

tions, we fed the network road images taken under a wide variety of viewing angles and lighting conditions. It would be impractical to try to collect thousands of real road images for such a data set. Instead, we developed a synthetic road-image generator that can create as many training examples as we need.

To train the network, 1200 simulated road images are presented 40 times each, while the weights are adjusted using the back-propagation learning algorithm. This takes about 30 minutes on Carnegie Mellon's Warp systolic-array supercomputer. (This machine was designed at Carnegie Mellon and is built by General Electric. It has a peak rate of 100 million floating-point operations per second and can compute weight adjustments for back-propagation networks at a rate of 20 million connections per second.)

Once it is trained, ALVINN can accurately drive the NAVLAB vehicle at about 3½ miles per hour along a path through a wooded area adjoining the Carnegie Mellon campus, under a variety of weather and lighting conditions. This speed is nearly twice as fast as that achieved by non-neural-network algorithms running on the same vehicle. Part of the reason for this is that the forward pass of a back-propagation network can be computed quickly. It takes about 200 milliseconds on the Sun-3/160 workstation installed on the NAVLAB.

The hidden-layer representations ALVINN develops are interesting. When trained on roads of a fixed width, the network chooses a representation in which hidden units act as detectors for complete roads at various positions and orientations. When trained on roads of variable

continued

Photo 1: *The NAVLAB autonomous navigation test-bed vehicle and the road used for trial runs.*

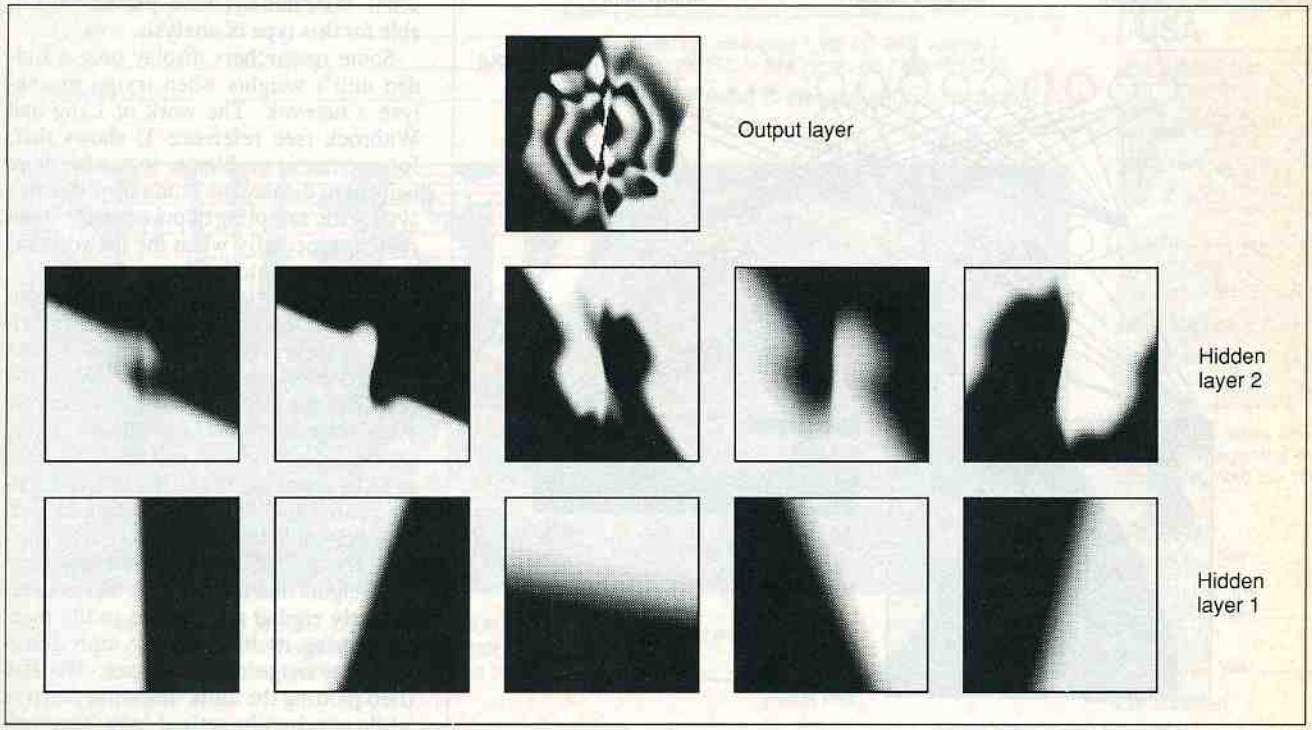


Figure 4: *Response function plots for the units in the two-spirals network. Each plot shows the activation level of a single unit as the* x,y *input to the network ranges over the interior of the unit square. The topmost plot is for the output unit, and the plots below are for the five units in each of the two hidden layers. (Figure courtesy of Kevin Lang and Michael Witbrock)*

widths, the hidden units turn into road-edge detectors, sensitive to only one of the two road edges. (Some look for left edges, and some for right edges.)

Figure 6 shows the weights to and from a single hidden unit after ALVINN was trained on roads of a fixed width. White squares represent positive values; black squares represent negative values. This hidden unit acts as a filter for two types of roads, one slightly to the left of center and one slightly to the right.

The weights from the video camera retina, along with the explanatory schematic, show the positions and orientations of the two road types that activate the hidden unit. Notice that the road specifications overlap: The large white region in the center of the weight diagram is a merger of the weights for the left edge of the rightmost road with the weights for the right edge of the leftmost road.

This hidden unit is also excited by obstacles in the periphery of the image and inhibited by obstacles in the center of the image where it expects the road to be. By fusing data from the video-camera and range finder sensors, hidden units can determine the position and orientation of the road more accurately than they could with either sensor alone.

This hidden unit makes excitatory connections to two sets of output units, dictating a slight left or right turn. Since it provides support for two turn directions, it must work with other hidden units to pin down the correct steering direction. Double-duty hidden units like this provide a compact representation. They allow a network with a small hidden layer to perform a complex task, like following a road, accurately.

Reducing the size of the hidden layer not only increases the rate at which a computer can simulate the network, it can also improve the network's performance. With too many hidden units, a network can simply memorize the correct response to each pattern in its training set instead of learning a general solution.

By limiting the size of the hidden layers, the network is forced to develop appropriate feature detectors to efficiently classify large sets of input patterns. These general-purpose feature detectors are more likely to be relevant to novel inputs, so the network performs better. In one experiment, we drove the NAVLAB vehicle using a network trained with only nine hidden units without any significant loss in driving proficiency.

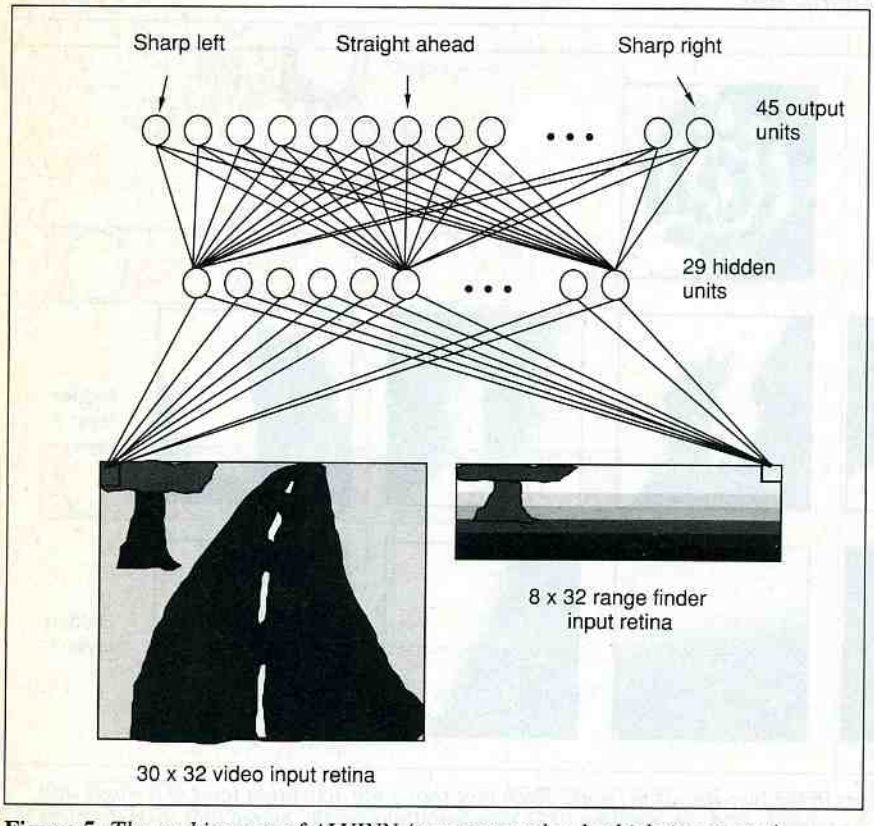


Figure 5: *The architecture of ALVINN (autonomous land vehicle in a neural network).*

Hidden units that act as filters for one to three roads are the most common result when ALVINN is trained on roads of a fixed width. The network develops a different representation when trained on images with varying road widths. Instead of developing into detectors for entire roads, the hidden units learn to look for a single road edge at a particular position and orientation.

The units support a wide range of travel directions. The correct travel direction for a road with an edge at a particular location varies substantially depending on the road's width. The hidden units cooperate with each other to determine the correct travel direction in any situation.

It's important to understand that no single hidden unit can perform the task alone; the collective activity of all the hidden units determines how the network behaves. Through this kind of cooperation, the network can use relatively coarse feature detectors and still maintain performance accuracy.

Hidden Units Demystified

It's easy to uncover what's in the hidden layers when you apply a neural network to a geometrical problem, as illustrated by the two-spirals and road-tracking examples. The visualization tools made practical by microcomputers and personal workstations have proved invaluable for this type of analysis.

Some researchers display only a hidden unit's weights when trying to analyze a network. The work of Lang and Witbrock (see reference 1) shows that, for geometric problems, it can be more helpful to display the unit's response to a systematic sampling of points in the input region, especially when the network has more than one hidden layer.

This practice is also common in classical neuroscience investigations of the visual system. You can't measure the weights between living neurons in the cortex of the brain, but you can measure their response to various inputs. Many studies of the visual system have been done by graphing the firing rate of cortical neurons while varying a stimulus pattern presented to the retina.

In the case of ALVINN, we saw from the weights that the network learns to efficiently exploit regularities in the input by making its hidden-layer units sensitive to a range of road types. We also tried plotting the units' response patterns while varying the retinal input (presenting roads at various positions and orientations); this confirmed our interpretation of what the hidden layer was doing.

Training ALVINN is time-consuming

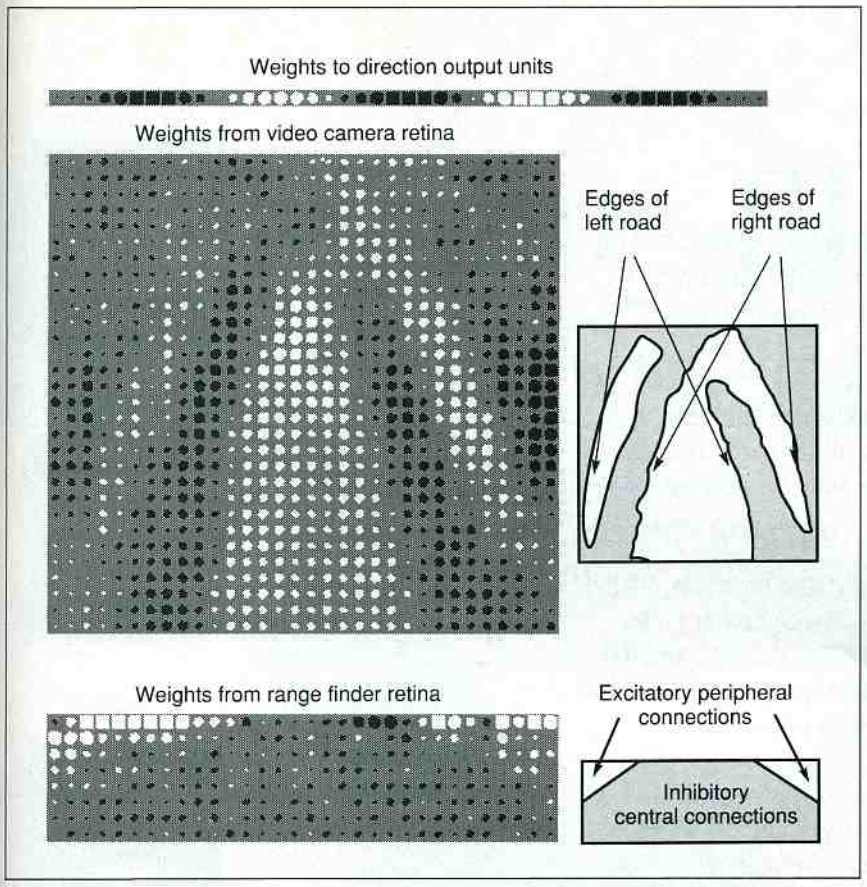


Figure 6: *Pattern of weights projecting to and from a single ALVINN hidden unit after training on roads with a fixed width. This hidden unit acts as a filter for two types of road, one slightly to the left of center and one slightly to the right. The explanatory schematics on the right side of the figure highlight our interpretation of these weights.*

and requires serious computing power, but you can implement the resulting network on a personal computer or workstation. We see this as a developing trend in neural computing: Training for real-world applications will be expensive, but delivery will be cheap. Analysis of networks through visualization is also easily done on personal workstations.

While we have removed some of the mystery concerning the representations that neural networks develop, the hidden layers have yet to give up all their secrets. One question still to be answered is how ALVINN accomplishes "sensor fusion," combining inputs from its video-camera and range finder retinas to arrive at the best steering direction. Experiments are under way to answer this. ■

REFERENCES

1. Lang, K. J., and M. J. Witbrock. "Learning to Tell Two Spirals Apart." In *Proceedings of the 1988 Connectionist Models Summer School*, D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski, eds. San Mateo, CA: Morgan Kaufmann Publishers, 1988.

2. Thorpe, C., M. Herbert, T. Kanade, S. Shafer, and the members of the Strategic Computing Vision Lab. "Vision and Navigation for the Carnegie Mellon NAVLAB." *Annual Review of Computer Science Vol. II*, Joseph Traub, ed. Palo Alto, CA: Annual Reviews, Inc., 1987.

3. Pomerleau, D. A. "ALVINN: An Autonomous Land Vehicle in a Neural Network." In *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, ed. San Mateo, CA: Morgan Kaufmann Publishers, 1989.

David S. Touretzky is a research computer scientist at Carnegie Mellon University in Pittsburgh, Pennsylvania. He has a Ph.D. in computer science from Carnegie Mellon. Dean A. Pomerleau is a doctoral student in computer science at Carnegie Mellon. They can be reached on BIX c/o "editors."