Visualizing processes in neural networks

by J. Wejchert
G. Tesauro

A real-time visualization toolkit has been designed to study processes in neural network learning. To date, relatively little attention has been given to visualizing these complex, nonlinear systems. Two new visualization methods are introduced and then applied. One represents synaptic weight data as "bonds" of varying length embedded in the geometrical structure of a network. The other maps the temporal trajectory of the system in a multidimensional configuration space as a two-dimensional diagram. Two-dimensional graphics were found to be sufficient for representing dynamic neural processes. As an application, the visualization tools are linked to simulations of networks learning various Boolean functions. A multiwindow environment allows different aspects of the simulation to be viewed simultaneously using real-time animations. The visualization toolkit can be used in a number of ways: to see how solutions to a particular problem are obtained; to observe how different parameters affect learning dynamics; and to identify the decision stages of learning. A demonstration videotape is provided.

Introduction

Representing scientific data by visual means is the essence of scientific visualization. It allows the human eye–brain system to perceive and infer visual information by geometrical and pictorial means rather than by linear numerical form. Enhanced with facilities for interactive manipulation of the imaging and display process, it is a highly efficient form of information transfer between simulation and user [1, 2].

Although neural networks are being widely investigated via computer simulations, the graphical display of related information has received little attention (with the exception of Hinton diagrams [3]). In other fields such as fluid dynamics and chaos theory, the development of visualization tools has proven to be a tremendously useful aid to research, development, and education. Similar benefits should result from the application of such techniques to neural network research, especially because such systems involve nonlinear, parallel, cooperative, and complex phenomena [3].

In this paper, several visualization methods are introduced to investigate learning in neural networks which use the back-propagation algorithm. A multiwindow environment is used that allows different aspects of the simulation to be displayed simultaneously in each window.

A new representation of neural network data is presented that displays synaptic weights as "bonds" of varying length embedded in the geometrical structure of a network. This description can be used statically, or as animation to depict the dynamics of learning.

A technique for representing the full temporal trajectory of the system in configuration space during the course of one or more learning runs is also introduced.

This draws multidimensional trajectories of the system as two-dimensional diagrams. Other more conventional graphical representations are used in other windows, and it is this set of visual tools that makes up our visualization toolkit.

As an application, the toolkit is used to study small networks that are learning Boolean functions. The animations are used to observe the emerging structure of connection strengths, to study temporal behavior, and to understand the relationships and effects of parameters. The simulations and graphics can run at real-time speeds.

A brief summary of the operation of neural networks is given first. Next, the motivation for the design of the graphics is discussed and the visual tools are described. Later, we present examples of the software applied to neural networks performing different operations. Finally, some general conclusions are made.

Neural networks

Neural networks offer a new approach to computation, based on a simplified model of the brain's organization. They can learn and generalize and are ideally suited to implementation on parallel architectures. The idealized models currently being examined consist of a set of elementary computational elements (neurons or nodes) and a corresponding set of interconnections (synapses). The strength of an interconnection is referred to as the weight of the connection between any two neurons.

In the majority of neural network models, neurons carry out a simple operation: All the weighted inputs from other neurons are summed, and this sum is passed through a nonlinear function; this becomes the output value of that neuron, in turn feeding to others:

$$O_j = f\left(\sum_i w_{ij} O_i + \theta_j\right).$$

The summation is taken over i neurons feeding to unit j. O_i is the output of neuron i, w_{ij} is the weight of the connection between neurons i and j, and θ is called the threshold value. The sum is then passed through a nonlinearity, usually the sigmoid function $f(x) = (1 + e^{-x})^{-1}$.

The networks considered here are of the feed-forward type, and are ordered into layers called input, hidden, and output. Each layer is fully connected to the next. The back-propagation algorithm provides a method for training such multilayer networks. During learning, the weights of the synaptic connections are adapted so that the difference between the calculated network output and desired output is minimized. Learning is a two-stage process: First the network output is calculated for each node, then the error is calculated and back-propagated through the network, and each weight is adjusted in proportion to that error. This overall procedure is iterated until the network converges and produces the required responses.

Usually the error function is given by

$$E = \sum_i^p (O_i - T_i)^2, \tag{1}$$

where O_i is the state of the output unit, T is the teacher signal (desired output), and p is the number of patterns.

The weights are changed by the gradient descent method

$$\Delta w = -\varepsilon \frac{\partial E}{\partial w}, \tag{2}$$

where ε is the learning rate, or by using an accelerating method that mixes in some of the previous iteration,

$$\Delta w(t) = -\varepsilon \frac{\partial E}{\partial w(t)} + \alpha \Delta w(t-1), \tag{3}$$

where α is referred to as the momentum parameter. The momentum and learning rate are important parameters for tuning the network to improve performance.

A useful way of interpreting the learning procedure is to consider it in terms of the movement of the point representing the system down an error hypersurface in a multidimensional weight space. Such an interpretation is commonly used in the description of physical systems, but cannot be simply visualized except for low-dimensional spaces.

Further information about neural networks may be found in [3] or [4].

Visual tools

In the visualization toolkit, different aspects of a simulation may be displayed simultaneously in a multiwindow environment, with each component of information mapped into a separate window. In designing the graphics, simple principles of displaying quantitative data were used [5]. The information in each window is as uncluttered as possible, and information is distributed in a number of windows, allowing the user to control which information should be displayed.

A principal window shows network topology and weight data as a "bond diagram"; other windows display projections of trajectories of the system in configuration space and show conventional plots of variables with time. We found that it is this integrated environment that is most effective in displaying the information.

Bond diagram

The principal visualization question is how to represent synaptic strength (weight) data. Hinton's method [3] of drawing squares of different sizes to represent different

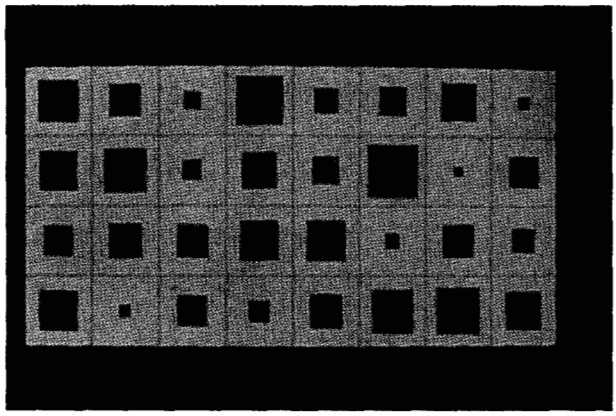


Figure 1

A Hinton diagram for a neural network with four input units, four hidden units, and four output units.

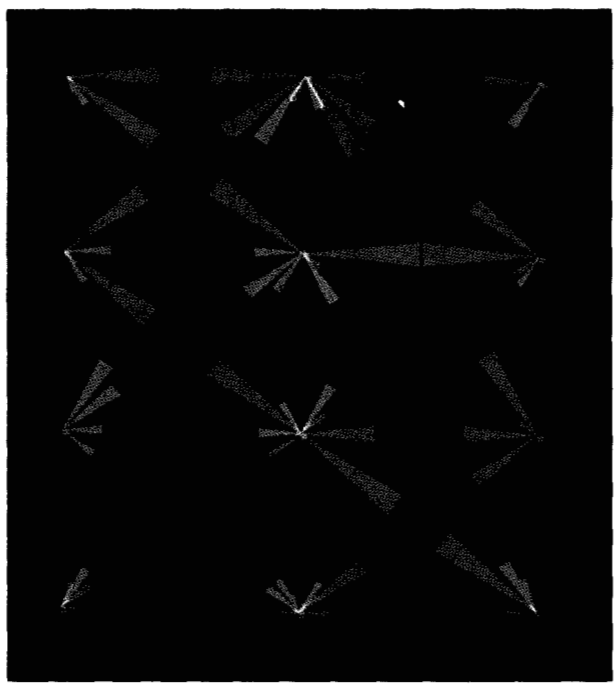


Figure 2

The "bond" diagram for a network with four input units, four hidden units, and four output units. Red and blue correspond to positive and negative weights.

weight values has been used extensively for this purpose. Typically, squares may be ordered according to the application for which the neural network is being trained, or placed in a way that suggests the network connectivity. **Figure 1** shows a Hinton diagram for a neural network with four input units, four hidden units, and four output units; positive and negative weights are displayed as red and blue squares, respectively.

The Hinton diagram does not explicitly reveal network topology in relation to the weight data, and it is not very useful in trying to understand problems relating to the internal representation of the network. Because it is of interest to try to see how the internal configuration of weights relates to the problem the network is learning, it is clearly worthwhile to have a graphical representation that explicitly includes weight information integrated with network topology. (A method used by Edelman [6] that color-codes connections between nodes has been used, but it is susceptible to color distortion and is not very suitable for animation.)

In our representation, the above objectives are met by displaying weights as "bonds" between nodes. The stronger the connection, the greater the length of the bond between any two nodes. Figure 2 shows the "bond" representation for the same network as in the Hinton diagram. As can be seen, it clearly incorporates structure with weight data. The bonds representing the weights extend from both nodes of a connection; one can look at any node and clearly see the magnitude of the weights feeding into and out of it. This allows one to detect differences in weight magnitude as differences in bond length, and is better suited to animation because changes in geometry are easier to perceive than changes in color [7]. Also, a sense of direction is built into the picture, since the bonds point to the node to which they are connected. Further, the collection of weights forms distinct patterns that can easily be perceived by the user, so that one can also infer global information from the overall patterns formed. It is probable that a different graphical design should be used for larger data-set sizes (networks with thousands of weights), where color or texture maps may be more appropriate. In such a case, the present method could be used to show portions of larger networks.

• *Trajectory diagram*

The description of a dynamic system in terms of a phase space or configuration space has been used extensively in the physical sciences. It has the advantage that the whole system can be described as one point in a space which encompasses all the possible states of the system. Information about the dynamics can be deduced from the behavior of the paths or trajectories created by the time evolution of such points. Like that of a physical system, the state of a neural network can be defined as a point in "weight space"; as the weights adapt and evolve, the dynamics of learning can then be described by trajectories of these points in weight space. Conventional 2D phase plots can be used to study the behavior of two components, but this is of little use if trajectories are

embedded in a high-dimensional space. Although attention has been given to visualizing multidimensional data [8–10], the issue of visualizing paths in such multidimensional spaces has not received much consideration. A straightforward way of reducing the trajectories in a multidimensional space into a two-dimensional picture is presented in this section.

The scheme is based on the premise that the human user has a good visual notion of vector addition. To represent an n-dimensional point in the plane, the axial components defining the point are defined as vectors and then plotted radially in the plane; the vector sum of these components is then calculated to yield the point representing the n-dimensional position. It is obvious that for $n > 2$ the resultant point is not unique; however, the method does allow one to infer information about families of similar trajectories, make comparisons between trajectories, and notice important deviations in behavior, as shown in the next section. A more formal account of this method is given in the Appendix.

Using the trajectory diagram, information can also be deduced about the error hypersurface [error landscape, as defined by Equation (2)] in the locality of a path. By assigning the current value of the global error function as the color of the current point in weight space, one obtains a sense of the contours of the error hypersurface and the dynamics of the gradient-descent evolution on this hypersurface. Further, the spacing of points gives an indication of the velocity on the surface. (An earlier attempt to obtain information about the energy landscape perturbed a system about a chosen arbitrary axis in weight space and plotted energy as a function of perturbation [11].)

• *Technical points*

The graphics software was written in C using X-Windows[1] version 11. X-Windows is portable and can be used to run graphics remotely on different machines using a local area network. The FORTRAN neural network simulator was linked and executed within the UNIX[2] environment, on the RT[3] workstation. Execution time was slow for real-time interaction except for very small networks (typically up to 30 weights). For larger networks the Stellar[4] graphics workstation was used, on which the simulator code could be vectorized and parallelized. The Stellar was used to run the graphics remotely over the local area network for display on the RT workstation. This resulted in a speed increase of a factor of ten, allowing networks of up to 200 weights to be run at real-time speeds.

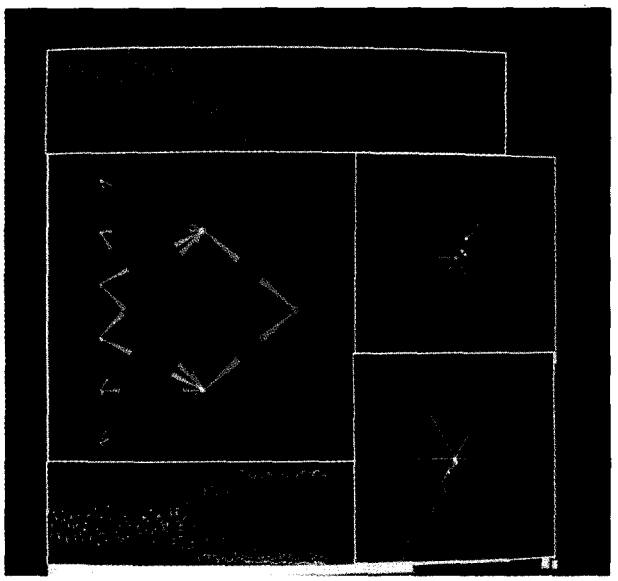


Figure 3

An example of the toolkit being used, with most of the windows active; the command line appears on the bottom.

A command-line argument is used to enter parameters such as momentum, learning rate, random seed, or the magnitude range of initial weights into a simulation. Runs can be restarted from the same random seed so as to study effects of changing parameters. Windows can be moved or closed during simulation, allowing the user to concentrate on chosen aspects of the simulation. The topology used for a problem may easily be adjusted from input files, and the graphics can be used to display neural networks of different topologies learning different tasks.

Research applications

Using the visualization toolkit, one can see how solutions to a particular problem are obtained and how different parameters affect these solutions, and observe stages at which learning decisions are made. In the examples, the bond diagram is used to represent weight values, which are colored red for positive and blue for negative; nodes are colored green. The same color code is used for the vectors representing the weights in the trajectory diagram. A gradation of red to blue is used to denote the magnitude of the error function in these plots. Two other windows are used to trace the time evolution of the total error function and the state of the output node. **Figure 3** gives an example of the toolkit in use, showing the bond diagram, the trajectory diagram, and other plots.

[1] X-Windows is a trademark of MIT.

[2] UNIX is a registered trademark of UNIX Systems Laboratories, Inc.

[3] RT is a registered trademark of International Business Machines Corporation.

[4] Stellar is a trademark of Stardent Computer, Inc.

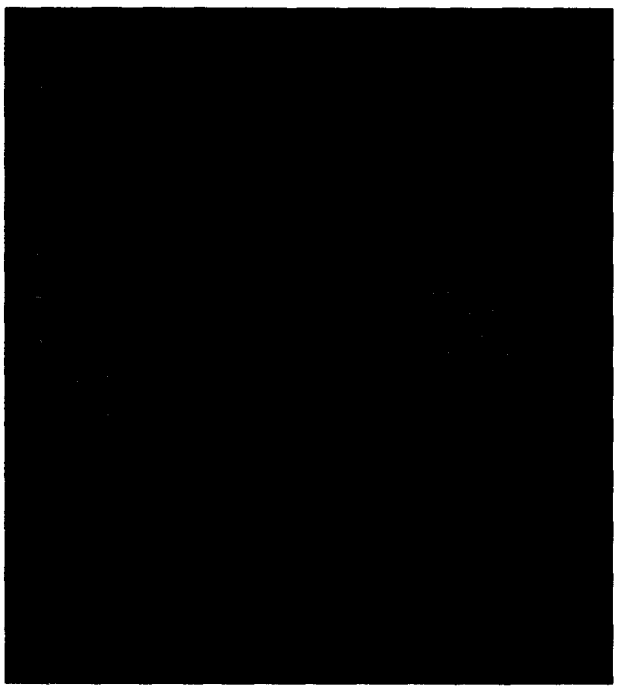


Figure 4

The final configuration of weights for the majority function. The size of the green disc corresponds to the threshold value.

As an example of the operation of the visualization toolkit, we focus on networks learning Boolean functions: Binary input vectors are presented to the network through the input nodes, and the teacher signal is set to either 1 or 0 depending on how the patterns are to be classified. Examples of such functions are majority, parity, and symmetry. The output of the majority function is 1 only if more than half of the input nodes are on; parity is an extension of the exclusive-or function; symmetry identifies input vectors that are perfectly symmetric about a central axis.

First we show how the configuration of weights corresponds to solutions for the majority and simplified symmetry functions using the bond diagram. Later, we investigate the evolution of learning for the general symmetry problem. The accompanying video shows the time evolution of the network as it learns these functions.

• *Configurations and solutions: Learning majority and simple symmetry*

Consider an input vector u_i, $i = 1 \cdots n$ presented to the network, where $u_i = \pm 1$. The majority function teacher signal is 1 if $\Sigma\, u_i > 0$, and 0 otherwise. A network topology with no hidden units can solve this classification [3]. **Figure 4** shows the final configuration of the two-layer network that has learned the majority function, from which we have an indication of how the network has solved the problem: A large output node is displayed and the magnitudes of all the weights are roughly uniform, indicating that a large bias (or threshold) is required to offset the sum of the weights. Majority is quite a simple problem for the network to learn; more complicated functions require hidden units, in which case the whole visualization toolkit becomes useful.

To see how the network solves a more complicated function, we consider a simplified symmetry function. This function discriminates between perfectly symmetric and perfectly antisymmetric patterns, and it is only given these two possibilities in the input set. This function may be defined as follows: Given an input vector u_i, $i = 1 \cdots n$, where $u_i = \pm 1$, the teacher signal is set to 1 when $u_j = u_{n+1-j}$, $j = 1, \cdots, n/2$ (symmetric), and 0 when $u_j = -u_{n+1-j}$, $j = 1, \cdots, n/2$ (antisymmetric). The network required to solve this mapping requires a topology with at least two hidden units [3].

A solution for a network with six input units, two hidden units, and one output is shown in **Figure 5**. It is interesting to note that the network has determined

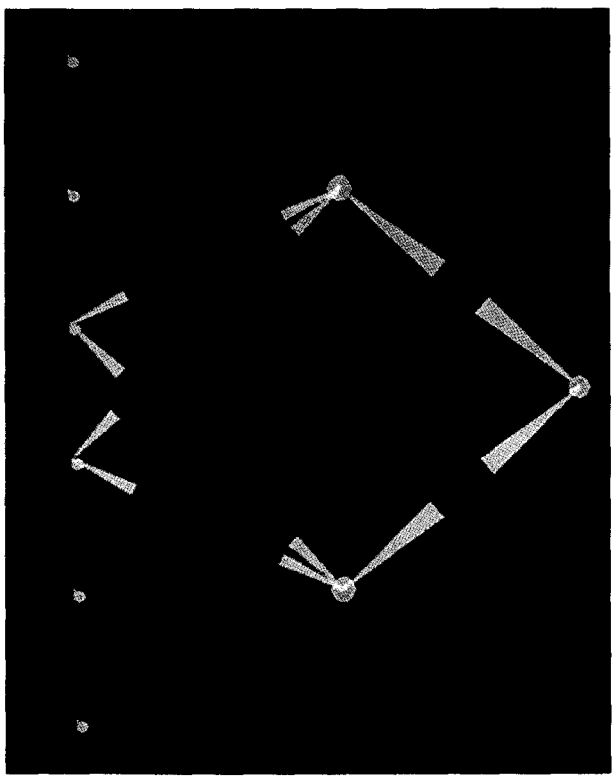
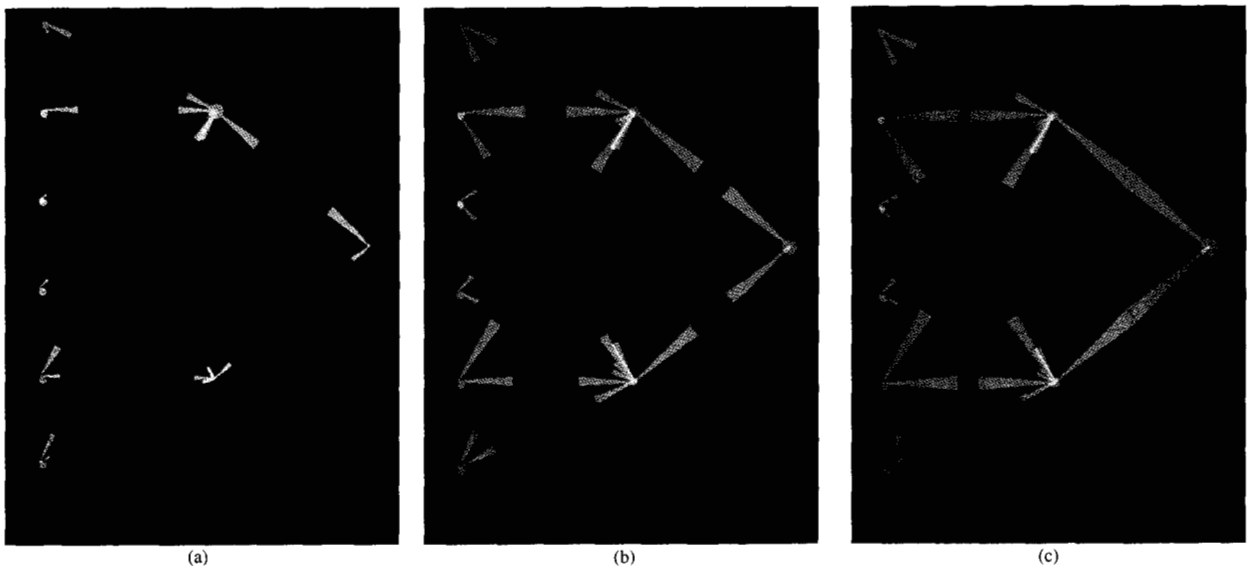


Figure 5

A solution of the simple symmetry function. As can be seen, only two input units are required to solve this problem.



(a) (b) (c)

Figure 6

An example of a network learning the general symmetry function. There are six input units, two hidden units, and one output unit. Weights are shown by bonds emanating from nodes. (a) Early in the learning process the weights are still small, and no overall structure has emerged. (b) An intermediate stage when the connections are becoming established. (c) The end of learning, showing the final pattern of weights.

(correctly) that it needs only two units to decide whether the input is totally symmetric or totally antisymmetric. In fact, solutions are not unique, as can be shown by using various random starting configurations; the network then may choose other symmetrically separated input pairs to support its decision. Note the patterns that the weights have created, as shown by the "bond" representation. As will be seen, this simple pattern carries over into the more general symmetry function, where the network must identify perfectly symmetric inputs from all the other permutations of input examples.

• *Parameters and decisions: General symmetry function*
To investigate the effects of parameters and see what decisions are made during the learning process, a more difficult learning function is required. The general symmetry function detects symmetrical patterns among all the binary permutations; i.e., it has a teacher signal of 1 when $u_j = u_{n+1-j}$, $j = 1, \cdots, n/2$ (symmetric) and 0 for all other patterns. Again, a network with six input units, two hidden units, and one output was used. Patterns were presented at random and the weights were updated every 2^{n-1} patterns. The command-line parameters were chosen to be α (momentum), r (magnitude of weights), and t (time for the simulation). The learning rate was fixed at $\varepsilon = 1.0$. To improve learning performance, the symmetric patterns must be presented with higher probability because they occur quite infrequently.

Snapshots of the evolution of a sample network are shown in **Figures 6(a)–(c)**. At the start of a simulation the weights are set to small random values [Figure 6(a)]. This can be seen as small random clusters of blue and red around each node. During learning, many example patterns of vectors are presented to the input of the network and weights are adjusted accordingly. Initially the rate of change of weights is small; later, as the simulation gets under way, the weights change rapidly, until only small changes are made as the system moves toward the final solution [Figure 6(b)]. A distinct pattern of red and blue triangles shows the configuration of weights in their final form [Figure 6(c)].

As can be seen from the bond diagram, the network has chosen a hierarchical structure of weights to solve the problem, using the same basic pattern of weights as in the simple symmetry problem. The major decision is made on the outer pair and additional decisions are made on the remaining pairs with decreasing strength. As before, the choice of pairs in the hierarchy depends on the initial random weights.

To monitor the simulation it is useful to have windows display the total error and the state of the output unit (**Figure 7**). Typically, the error initially stays high, then decreases rapidly and levels off to zero as final adjustments are made to the weights. Spikes in this curve are due to the method of presenting patterns at random. The state of the output unit initially oscillates and then

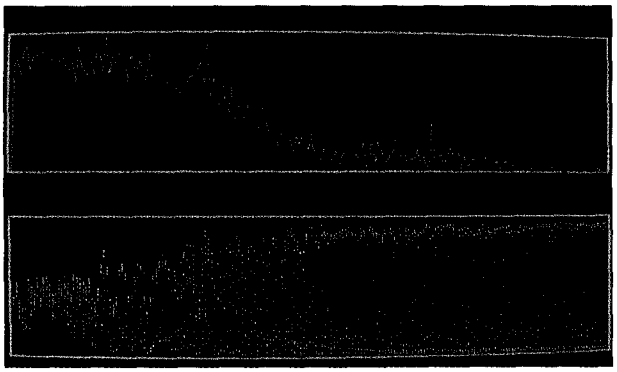


Figure 7

The total error (top) and the state of the output unit (bottom) for the learning run described in Figure 6.

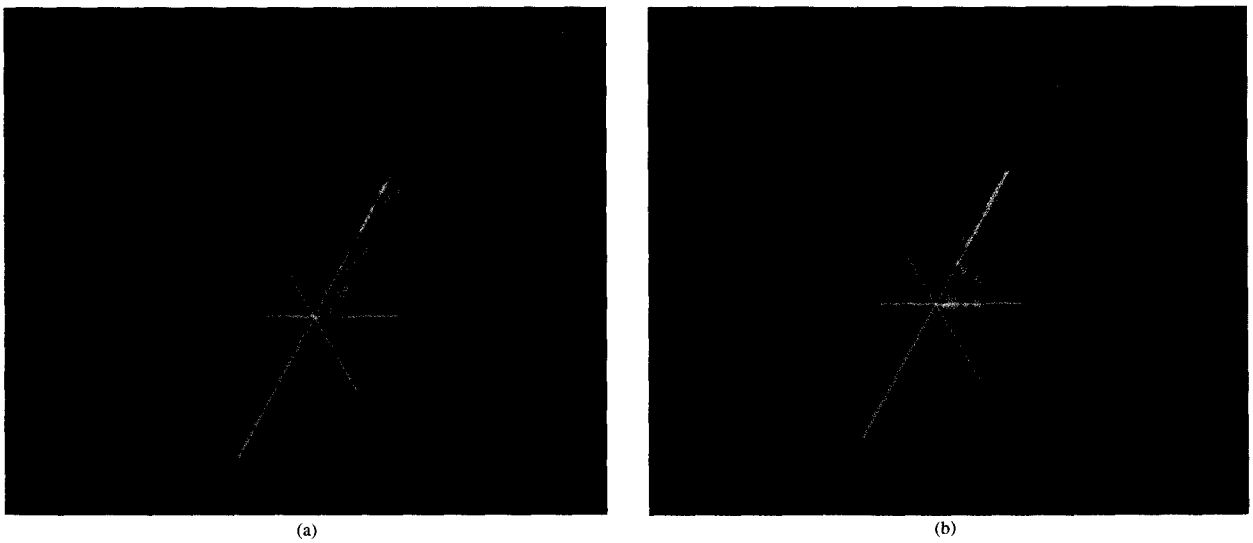
bifurcates into the two required output states. As can be seen from Figure 7, bifurcation and the rapid decrease in error occur at roughly the same time.

The temporal trajectory of the system in weight space reveals interesting information. As described in the Appendix, the components of a point in the multidimensional space are treated as vectors, plotted radially in the plane and then added to yield a point in the two-dimensional representation. In developing this method we initially experimented with conventional 2D phase trajectories, which were then compared with the vector method. We found that the qualitative features of both carried over into the higher-dimensional representations.

Instead of using the trajectory diagram to represent the whole space, we found it more useful to map trajectories into subspaces, particularly when tracking the behavior of the hidden units. Thus, for the same six-input network as used before, we displayed two diagrams (one for each hidden unit), each representing a six-dimensional path **[Figures 8(a) and 8(b)]**. In this example, where the network does converge to a correct solution, the paths of the two hidden units tried either to match each other (in which case the configurations of the units were identical) or to move in opposite directions (in which case the units were opposites). Both led to valid solutions, the former of which is displayed in Figure 8.

By contrast, for learning runs which do not converge to global optima, we found that (usually) one of the hidden units followed a normal trajectory, whereas the other unit was not able to achieve the appropriate match or anti-match. This is because the signs of the weights of the second hidden unit were not correct and the learning algorithm could not make the necessary adjustments.

By repeatedly watching the animations, we could make some qualitative observations about the stages of learning from the bond diagram. We found that the early stages were the most critical, because the signs of the weights feeding to the hidden units were determined at this time. Any problems with the gradient search always occurred at this stage. The later stages posed fewer problems because only the relative magnitudes of the weights were adapted. Once the network had sorted the signs of the



(a) (b)

Figure 8

Two reduced trajectories of the hidden units. (a) corresponds to the topmost unit in the network. Note that the axes in (a) and (b) correspond to the weights feeding into the respective units.

weights, rapid gradient descent could occur and the error quickly decreased; simultaneously the output state bifurcated.

The trajectory diagram can be used to locate the exact point during learning that determines whether the network will end up in global or local minima. To do this, one can treat the momentum parameter as a variable that affects the final solution. If momentum is increased, then trajectories tend to follow straighter lines, which speeds up learning. However, if momentum is too large, problems occur with the gradient search and the system may end up in a local minimum. **Figure 9** shows an example of trajectories that were collected for two hidden units. In this case both the trajectories are mapped onto the same diagram. These are similar to previous runs, except that the graphics are plotted on a scale that shows the initial behavior of the trajectories in greater detail. The momentum was set to $\alpha = 0.6$ and $\alpha = 0.7$; these values mark a transition between the network finding a global solution and getting stuck in a local minimum. As the trajectory diagram reveals, a definite point in configuration space marks a decision for one of the hidden units. If $\alpha > 0.7$, then the hidden unit proceeds into a completely different portion of configuration space corresponding to a local minimum on the error hypersurface. Thus the trajectory diagram was able to show that deviations in behavior are determined at the early stages of learning. Further, it could give an indication of the exact point in weight space at which such deviations occur. In this way the trajectory diagram can be used as a herald of bad learning behavior and may suggest heuristics that could improve learning by setting initial trajectories in "correct" directions.

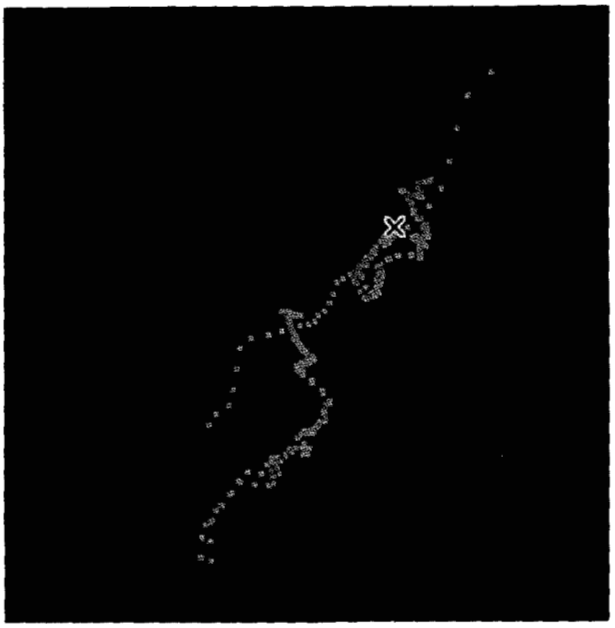


Figure 9

A trace of paths of the two hidden units for α equal to 0.6 and 0.7. The upper path deviates and ends up in a different region of weight space by trying to match the lower path. The "X" marks the transition in the upper path for the parameter change. The lower path remains the same in both cases.

Conclusions

We have created a visualization toolkit to study processes in neural networks, and have used this toolkit to gain insights into learning processes. At the design phase of the project we concentrated on creating useful graphical representations of neural networks. These are the bond diagram for representing neural weight data and a way of mapping the configuration trajectory of the system.

We have found that the bond diagram was very useful in showing how internal representations are related to network function. It was also helpful in revealing insights about the dynamic stages of learning. Similarly, we demonstrated the usefulness of the trajectory diagram in depicting multidimensional trajectories as two-dimensional pictures. This method should also be of use in other areas where similar problems must be visualized. In general, we found the use of the interactive multiwindow environment to be extremely effective for the purpose of visualization. Such an approach allows modularity so that new tools can be added to the toolkit. For example, Hinton diagrams or alternative representations could easily be included.

It is fascinating to conjecture how much can be understood from simulations and experiments by purely visual means. Indeed, we have gained many insights from the visualization toolkit, and further use of such visual tools should lead to even greater knowledge. Two-dimensional graphics were found to be sufficient for representing dynamic neural processes, and this approach allowed us to concentrate on all the aspects of visualization, rather than just on building the tools. We fear that visualization research has so far concentrated too much on techniques for producing pictures rather than on *how* to design them or on real examples of their *application* and *usefulness*. We hope to have contributed to bridging this gap by building, applying, and using visual tools for our research purposes.

Appendix

Consider a vector representation of a point in an n-dimensional space $\mathbf{r} \in \mathbb{R}^n$,

$$\mathbf{r} = (r_1, r_2, \cdots, r_n) = r_1\mathbf{k}_1 + r_2\mathbf{k}_2 + \cdots + r_n\mathbf{k}_n , \qquad \text{(A1)}$$

where $\mathbf{k}_1, \mathbf{k}_2, \cdots, \mathbf{k}_n$ are the orthonormal basis vectors.

Figure 10

An example of a collection of paths from a six-dimensional space taken from a network simulation. Paths ending in local minima and the symmetry of certain solutions can be identified.

To represent this point in a two-dimensional diagram, $\mathbf{r}$ is mapped into the point $\mathbf{a} \in \mathbb{R}^2$ defined as

$$\mathbf{a} = (a_1, a_2) = a_1\mathbf{i} + a_2\mathbf{j}, \tag{A2}$$

where $\mathbf{i}$, $\mathbf{j}$ are the usual orthonormal basis vectors. Now suppose we take n unit vectors, $\mathbf{m}_1, \mathbf{m}_2, \cdots, \mathbf{m}_n$, and plot them radially in the plane. This corresponds to taking each of the $\mathbf{k}_i$ and positioning them so as to lie in the plane. Then, by analogy with Equation (A1), we take the representation of $\mathbf{r}$ to be the sum

$$\mathbf{a} = r_1\mathbf{m}_1 + r_2\mathbf{m}_2 + \cdots + r_n\mathbf{m}_n = \sum_i \mathbf{r}_i\mathbf{m}_i . \tag{A3}$$

Now, any of the above vectors $r_i\mathbf{m}_i$ can be written in terms of components relative to the $\mathbf{i}$, $\mathbf{j}$ axes, that is,

$$r_i\mathbf{m}_i = r_i \cos(\theta_i)\mathbf{i} + r_i \sin(\theta_i)\mathbf{j}, \tag{A4}$$

where θ_i is the angle between $\mathbf{m}_i$ and the $\mathbf{i}$ unit vector.

Since $\mathbf{a}$ is defined by the sum of these vectors, its components are

$$a_1 = \sum_i r_i \cos(\theta_i), \tag{A5}$$

$$a_2 = \sum_i r_i \sin(\theta_i). \tag{A6}$$

This explicitly defines the mapping from $\mathbb{R}^n$ to $\mathbb{R}^2$. Each of the θ_i are control parameters left to the user. By changing the directions of the unit vectors, one changes the display of the trajectory. For example, if it were found that too many of the vectors $r_i\mathbf{m}_i$ were canceling each other, the unit vectors could be reorganized to yield an alternative plot. Note that when $n = 2$, the two unit vectors $\mathbf{m}_1$, $\mathbf{m}_2$ can be chosen at right angles and the mapping results in the usual geometry of an x–y plot. However, when $n > 2$ the resultant point becomes nonunique and can be produced in many ways.

In general, the mapping corresponds to making a projection in n-space onto a surface, and moving the unit vectors corresponds to rotations of such a surface in the space. However, the method described here has advantage over conventional projection techniques because the user can estimate how the vectors will add so as to ensure a good spread of points.

Figure 10 gives an example of the use of a diagram which shows a collection of trajectory traces for a network with six weights. From this one can identify which families of paths end in local minima, or observe the symmetry of certain groups of paths. The trajectory diagram has similar uses to a conventional phase plot: It can distinguish between different regions of configuration space, it can be used to detect critical stages of the dynamics of a system, and it gives a trace of its time evolution.

Acknowledgments

We wish to thank Scott Kirkpatrick for help and encouragement during the project and to thank members of the Visualization Laboratory (Yorktown Heights) for the use of their resources.

References

1. B. H. McCormick, T. A. DeFanti, and M. D. Brown, Eds., "Visualization in Scientific Computing," *Comput. Graph.* **21,** No. 6 (1987); see also "Visualization in Scientific Computing—A Synopsis," *IEEE Computer Graph. & Appl.* **7,** 38–44 (July 1987).
2. R. S. Wolff, "Visualization in the Eye of the Scientist," *Computers in Physics* **2,** No. 3, 28–35 (May/June 1988).
3. D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* Volume 1, MIT Press, Cambridge, MA, 1986.
4. R. P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Mag.* **4,** No. 2, 4–22 (April 1987).
5. E. R. Tufte, *The Visual Display of Quantitative Information,* Graphic Press, Cheshire, CT, 1983.
6. G. M. Edelman, *Neural Darwinism,* Basic Books, New York, 1987.
7. M. Livingstone, "Art Illusion and the Visual System," *Sci. Amer.* **258,** No. 1, 78–85 (January 1988).
8. D. J. Cox, "Using the Supercomputer to Visualize Higher Dimensions: An Artist's Contribution to Scientific Visualization," *Leonardo* **21,** No. 3, 233 (1988).
9. E. J. Farrell, "Visual Interpretation of Complex Data," *IBM Syst. J.* **26,** No. 2, 174–200 (1987).
10. C. A. Pickover, "DNA Vectorgrams: Representations of Cancer Genes as Movements on a 2D Cellular Lattice," *IBM J. Res. Develop.* **31,** No. 1, 111–119 (1987).
11. D. C. Plaut, S. J. Nowlan, and G. E. Hinton, "Experiments on Learning by Back Propagation," *Research Report CMU-CS-86-126,* Carnegie Mellon University, Pittsburgh, PA, 1986.

Received November 12, 1989; accepted for publication July 26, 1990

Video notes

1) Majority function

This shows a 15-input network learning the majority function. During the run many input patterns are being presented to the network, during which time the weights are changed. The weights evolve from small random values through to an almost uniform set corresponding to the solution of the problem. The threshold of the output unit (green disc) evolves to offset the sum of the weights.

2) Simple symmetry function

Here perfectly symmetric or perfectly antisymmetric patterns are presented to the network. Two examples of learning are shown leading to different but valid solutions. Positive and negative weights are colored red and blue, respectively. The upper window shows the total error, and the lower window the state of the output window.

3) General symmetry function

Here the network is required to detect symmetry among all the possible input patterns. The two extra windows on the right show the trajectory diagrams for the two hidden units. In the first example the network can find a solution to the problem. In the second example the momentum parameter has been changed, and the network gets stuck in a local minimum; the upper unit follows a different trajectory.

Jakub Wejchert *European Visualization Centre, IBM Scientific Centre, St. Clements St., Winchester, Hampshire SO23 9DR, England.* Dr. Wejchert is currently a Research Fellow working with the European Visualization Group at the IBM Scientific Centre, Winchester. He obtained his B.A. in theoretical physics from Trinity College, Dublin, Ireland (1983). He then went to University College, Dublin, to do a M.Sc. (by research) in simulational physics. Dr. Wejchert then returned to Trinity to do his Ph.D. research in simulational physics, obtaining his degree in 1988. He has worked at the Centro Commune di Ricerca, Italy (1986–1988) and at the Computer Sciences Department of the IBM Thomas J. Watson Research Center (1988–1990). Dr. Wejchert's current interests are in the areas of scientific visualization and computer animation.

Gerald Tesauro *IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.* Dr. Tesauro is a Research Staff Member in the Computer Sciences Department at the IBM Thomas J. Watson Research Center. He received a B.S. degree in physics from the University of Maryland in 1980 and a Ph.D. in physics from Princeton University in 1985. After a postdoctoral appointment at the Center for Complex Systems Research, University of Illinois, Dr. Tesauro joined IBM in 1988. His research interests include theoretical analysis of neural network learning algorithms, as well as their application to real-world problems, and modeling of learning and information processing in real biological neural networks.