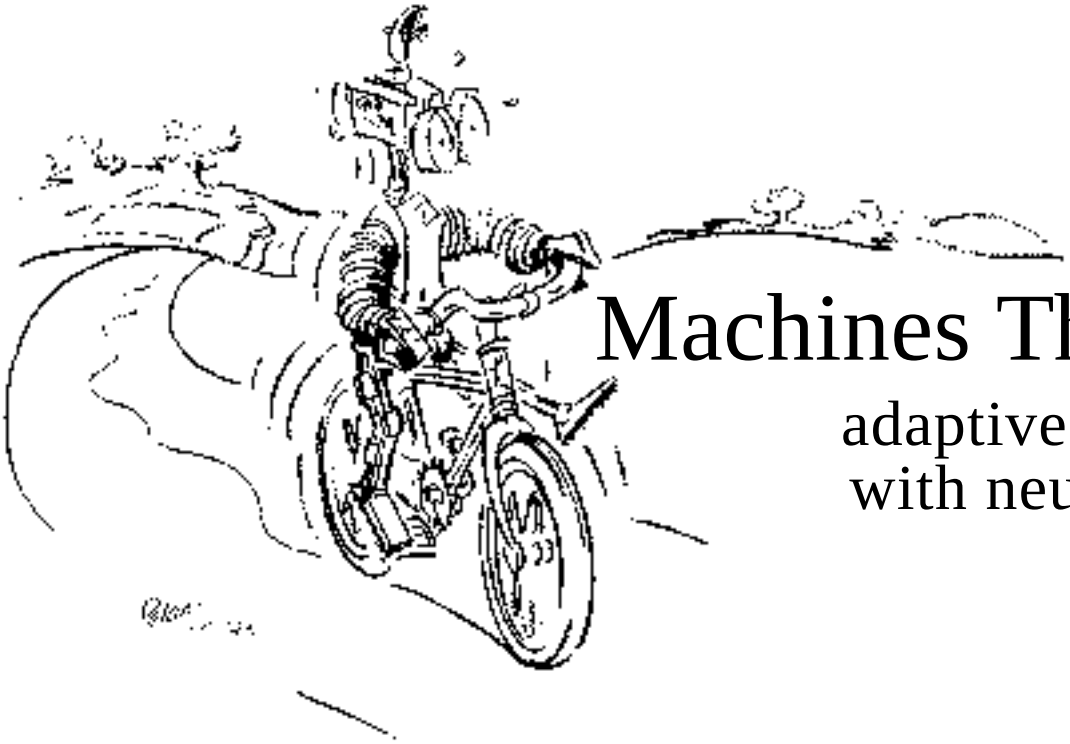

Machines That Learn

adaptive computation with neural networks

Roger D. Jones

Early in 1989 I found myself on a team that was responsible for automating the control and tuning of the source for a negative-ion accelerator. The purpose of the source was to create a rapidly pulsed ion beam to be injected into an accelerator. The pulses had to have high current and low noise and had to be reliably similar to each other.

The task presented a number of challenges. The machine was complex. The chemistry and plasma physics in the ion source were sufficiently complicated that modeling was very difficult. Moreover, at given control settings the machine's output drifted significantly in times as short as an hour. Therefore the controller had to adapt constantly to changing conditions inside the machine. Finally, since the source was ultimately intended to operate in space, humans could not be involved in adjusting it.

Earlier attempts at automating the source had failed. On the other hand, despite the complexity of the problem and the drift of the source's response to the control settings, experienced human technicians routinely tuned the machine. Their success suggested the path to the automation of the problem. We had been thinking at the time about computational algorithms, called artificial neural networks, that learn and adapt in crude imitation of the ways humans learn. The ion-source control problem seemed like an ideal test bed for some of our ideas on machine learning. In fact, Bill Mead, P. S. Bowling, and Stan Brown ultimately designed and implemented a controller that used machine-learning techniques to successfully tune and control the beam. The performance of the controller was comparable to that of a human control engineer who had a few months' training on the device.

The control of the negative-ion source is a representative of a class of technical problems that is growing at a tremendous rate. These problems are characterized by the requirement that action be taken as information flows to the controller. The amount of data available is often enormous. The controller is often required to identify the relevant facts from the fire hose of available information and to shape those facts so that meaningful control actions can be taken. Human beings accomplish this task regularly. We constantly learn, adapt, and revise our internal models of the world in order to control our environments more effectively. The challenge is to automate the capability.

In this article we will present some of the basic ideas in machine-learning research and discuss how the techniques can be used to solve practical problems such as the control of the negative-ion source. We will also discuss the direction research in this field seems to be heading.

What Is Machine Learning?

In setting up the ion-source problem for a computer, we represented the control settings by a vector, **x**, whose components corresponded to hydrogen flow rate, arc voltage, cathode temperature, and anode temperature. We defined a fitness function, f, to measure the quality of the beam. Our fitness function was a linear combination of the beam's current, its noise level, and a statistical measure of the variability of the pulses. A large value of the fitness function indicated a desirable beam. The beam quality depended on the control settings. For example, at one instant the flow rate might be set to x_1, the arc voltage set to x_2, and the cathode and anode

temperatures set to x_3 and x_4 respectively. For these settings, the beam quality would be $f(x_1,x_2,x_3,x_4)$. The problem was to find a value of the control settings that maximized $f(\mathbf{x})$.

Our method was to reconstruct $f(\mathbf{x})$ from its values at various control settings (about seventy), or more precisely to find a function $\Phi(\mathbf{x})$ that approximates $f(\mathbf{x})$ by interpolating between and extrapolating from those values. The values of $\mathbf{x}$ and $f(\mathbf{x})$ that determine $\Phi(\mathbf{x})$ are called training examples because each pair of values of $\mathbf{x}$ and $f(\mathbf{x})$ is incorporated one at a time into the formula for $\Phi(\mathbf{x})$ and the quality of the approximation should improve with each additional training example. When a good approximation $\Phi(\mathbf{x})$ has been found, standard search techniques can be used to find the value of the control settings that maximizes $\Phi(\mathbf{x})$. Those control settings are a good approximation to the desired control settings that maximize $f(\mathbf{x})$. When a computer goes through this process, we say it "learns" to control the ion source.

Simple methods for function approximation from training examples are well known. For instance, a smooth function can be approximated by a series of sinusoidal functions known as a Fourier series:

$$\Phi(x;a,\omega) = \sum_{n=0}^{N} a_n \cos(n\omega x + \delta_n).$$

The cosines are known as the *basis functions* of the Fourier series. The a_js and b_js are adjustable parameters. The frequency, ω, is determined by the spacing of the examples and by the boundary conditions. Suppose we have a number, N, of training examples, $[\mathbf{x}^p, f(\mathbf{x}^p)]$, for p=1 to N, at evenly spaced values of $\mathbf{x}$. Then the best approximation f of the form Φ is given by choosing the adjustable parameters, a and δ, according to

$$a_n = \sqrt{\alpha_n^2 + \beta_n^2}$$

$$\delta_n = \arctan(\beta_n/\alpha_n);$$

and

$$\alpha_n = \frac{1}{N} \sum_{p=1}^{N} f(x_p)\cos(n\omega x_p)$$

$$\beta_n = \frac{1}{N} \sum_{p=1}^{N} f(x_p)\sin(n\omega x_p):$$

where
and

If the function to be reconstructed varies slowly on scales of the order of the example spacing, then Φ is a good reconstruction of f from N examples. On the other hand, if f varies more rapidly, Φ can be a very poor approximation.

The Fourier series is convenient to compute. Because we can employ a separate computational hardware element to calculate the parameters associated with each sinusoid, we can perform the $2N$ computations for the evaluation of Φsimultaneously in parallel. That method is much faster than waiting for the parameters for each sinusoid to be evaluated in turn. Since each hardware element performs a similar calculation, the construction of each element can be the same. Input information is broadcast to many elements and output information is collected from many elements. Figure 1 shows the architecture of a computer built to perform those calculations; such a computer is known as a network. Network architecture is a powerful concept. It permits the construction of a large class of mappings, $\Phi(\mathbf{x})$, from a large class of examples, $\mathbf{x}$ and $f(\mathbf{x})$, using fixed arrays of similar or identical hardware elements.

Many biological systems take advantage of the network concept. In particular, neurons are simple computational elements which occur in great numbers. Neurons in the brain and spinal cord typically receive information from many other neurons, including input sensors (sensory nerves in the eyes, skin, and so forth), and pass information to many other neurons and to output transducers (muscles). Very simple systems of neurons can carry out sophisticated control actions. More complex arrangements can perform very sophisticated processing such as that involved in reading and understanding this paragraph.

The Fourier series can approximate smooth integrable functions well. It is very useful for handling many simple computational tasks, partly because it can be calculated by a network. However, Fourier approximation, like all approximation techniques based on orthogonal functions, also has three unpleasant properties that prevent its use in the control of the negative-ion source: (1) In the ion-source problem as well as in many other problems of interest, the data are not sampled evenly in the input space. Fourier approximation requires even sampling. (2) We needed to make decisions on how to adjust the controls as the input data came in, whereas Fourier methods require all the data to be in before an approximation is made. (3) We expected to have to

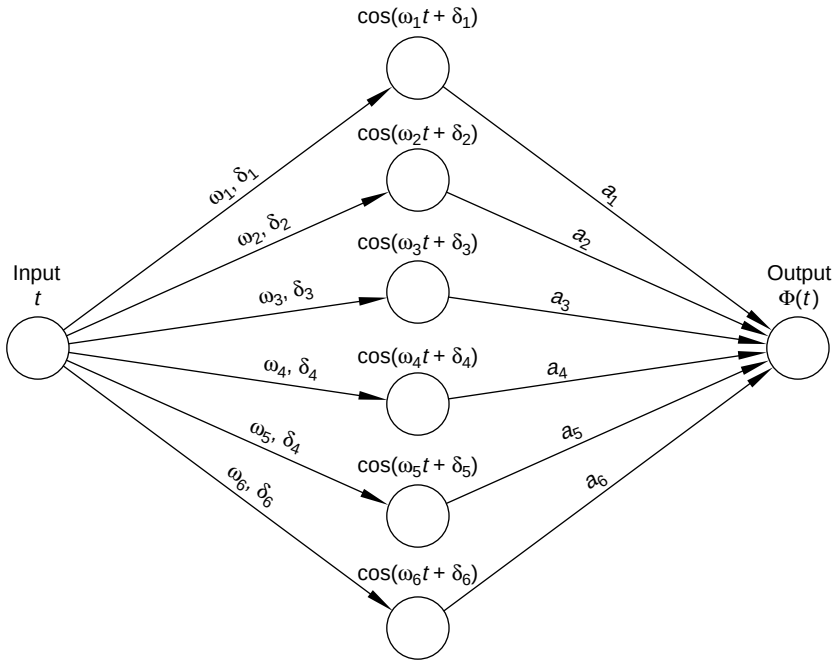

Figure 1. The Network Structure of the Fourier Series
The input and output are each associated with a single node. The sinusoids are identified with the layer of hidden nodes. The frequencies, w_j, must be calculated before the sinusoids are known, so they are associated with the links between the input node and the hidden nodes. (The frequencies are given by $!_n = n!$, where $!$ is calculated from the spacing of the examples and from the boundary conditions.) The adjustable weights of the sinusoids, a_n and $\pm_n$, are associated with the links between the hidden nodes and the output node because the weights for each sinusoid are calculated from the sinusoid and the training examples.

deal with many controls and consequently many input dimensions. The number of basis functions in the Fourier method increases exponentially with the number of dimensions, making Fourier approximation of high-dimensional functions impractical. A further disadvantage of the Fourier series is that it cannot approximate functions that are not smooth, but we would like to approximate such functions in many problems.

The CNLS Network

How do we design a computer-learning method that avoids the disadvantages of the Fourier series and still retain the advantages of multiple similar computational elements? Fourier series have the valuable properties that the basis functions are orthogonal and span the input space. The orthogonality property guarantees the existence of an algorithm to compute the Fourier coefficients in one step. The fact that the space is spanned guarantees that any smooth function can be accurately approximated. It turns out that in many real problems those guarantees are more than we require. If we choose a basis set that "mostly" spans the space and is designed to approximate functions "of the type in which we are currently interested," then we can reduce the number of basis functions in high-dimensional approximations by orders of magnitude. The penalty for using "almost" orthogonal functions is that the adaptive parameters often must be calculated by an iterative process rather than in a single step. However, in typical problems solved by computer learning, the data arrive one example at a time, so the calculation must be iterative anyway.

The use of networks that can be trained was originally suggested by the ways humans learn. The networks were inspired by the networks of nerves in the brain and the process of training from examples was inspired by human learning from examples. Such computational systems are called artificial neural networks because of their analogies to animal nervous systems; biologists insist on the word "artificial" because of the many differences.

$$f(x) = \frac{\sum_{j=1}^{M} f(x)\%_j(x)}{\sum_{j=1}^{M} \%_j(x)};$$

To find an appropriate set of nonorthogonal basis functions for an artificial neural network, we start with the identity,

$$'(x) = \sum_{j=1}^{M} [a_j + (x - c_j)^T d_j] \frac{\%_j(x)}{\sum_{k=1}^{M} \%_k(x)};$$

where M is the number of basis functions. We choose each basis function $\Omega_j(\mathbf{x})$ to be a localized function (such as a Gaussian) whose center is at $\mathbf{c}_j$; that is, each basis function is large in the neighborhood in input space around its center and is small outside the neighborhood. If $f(\mathbf{x})$ is a smooth function, it

can be approximated by its Taylor expansion about $\mathbf{c}_j$.

Approximating $f(\mathbf{x})$ by discarding the quadratic and higher-order terms of the Taylor expansion gives
The superscript T indicates a transpose in input space; in other words, the second term in brackets is a dot product or inner product. Here $\mathbf{a}_j$ corresponds to the zero-order term in the Taylor expansion and $\mathbf{d}_j$ to the gradient term. In principle these terms could be approximated directly from the data. We will, however, regard them as adjustable or adaptive parameters; that is, they will be changed in the process of training. In some cases we will also regard

$$u_j(x) \cdot \frac{\Omega_j(x)}{\sum_{k=1}^{M} \Omega_k(x)} :$$

the basis-function centers, $\mathbf{c}_j$, and the widths of the basis functions as adaptive parameters; in other cases we will simply specify them. In this approximation the basis functions are

The adjustable parameters are changed each time a training example is shown to the network. A *learning rule* computes new parameters from the previous parameters so that the new $\Phi(\mathbf{x})$ approximates $f(\mathbf{x})$ better than the previous $\Phi(\mathbf{x})$. The examples need not be evenly spaced in input space.

What is the appropriate form for Ω? All we have required so far is that the function be localized. We would like to choose Ω in a manner that maximizes the accuracy of the mapping and the ability of the network to generalize, that is, to interpolate between and extrapolate from the data points in a way that uses as much of the information they contain as possible. Information-theory arguments indicate that Gaussians optimize accuracy and generalization. Therefore our network uses Gaussians for the Ω_j functions. A diagram of the network appears in Figure 2. We call it the CNLS network, for Connectionist Normalized Local Spline (alluding to the Laboratory's Center for Nonlinear Studies, where some of the development of the network was done).

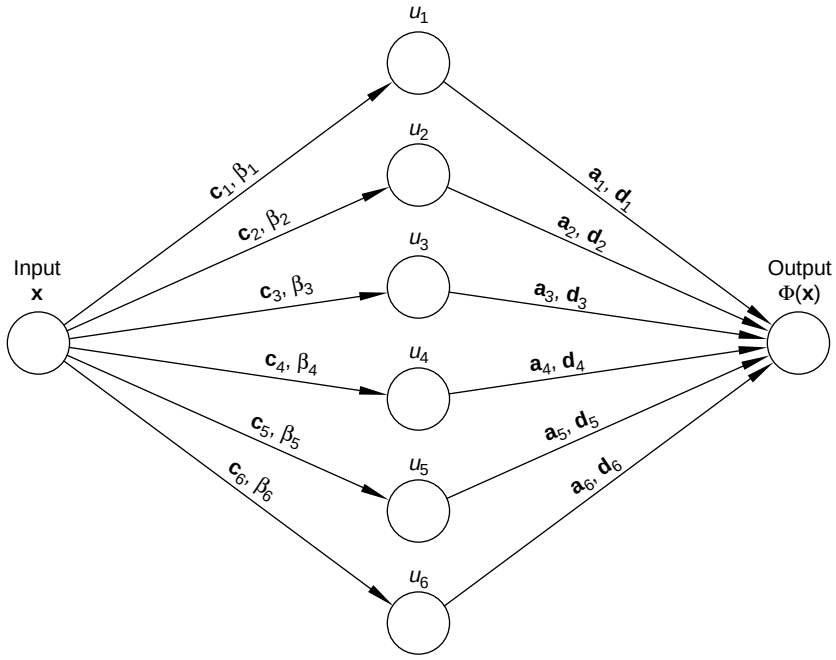

Figure 2. The Network Structure of the CNLS Net
The basis functions, u_j, are normalized Gaussians. Because the basis functions depend nonlinearly on their inverse widths, $\emptyset_j$, and centers, $\boldsymbol{c}_j$, those parameters are associated with the links between the input node and the hidden nodes. The linear weights, $\mathbf{a}_j$ and $\mathbf{d}_j$, are calculated after the basis functions are known, so they are associated with the links between the hidden nodes and the output node. In contrast to networks that calculate Fourier approximations, the CNLS net calculates adjustable parameters by an iterative process. Each step of the process alters the parameter values resulting from the previous step to produce new parameter values that better approximate the training examples. The process is complete when additional steps do not appreciably improve the approximation.

Unlike the sinusoids in the Fourier transform, the set of Gaussian basis functions does not span the input space. Therefore not every smooth function can be approximated by this network. However, simple functions (those without too many wiggles) are not a problem. Furthermore the number of basis functions required for approximation of simple functions does not explode when the number of input dimensions is high, as it does when the Fourier series is used.

Applying the CNLS net to control of the negative-ion source. Given the network architecture and the basis functions, how do we train the adjustable parameters? The best way is to define a cost function that

measures the error of the approximation and then minimize that function by adjusting the parameters. Perhaps the simplest and most common cost function is the sum of the squares of the differences between the measured output, f (of the negative-ion source in the present case), and the network output, Φ:

$$E = \frac{1}{2}\sum_{p=1}^{N} [f(x_p) - \Phi(x_p; a)]^2 ,$$

The summation is over the set of training examples and a is shorthand for the set of adjustable parameters. This least-squares cost function has a minimum value of 0 when Φ exactly matches f at the training points.

Figure 3 shows the architecture of the controller for the negative-ion source. It consists of an ion-source adjuster, a neural network, a neural-network trainer, and a comparator. Each time a new training example, $[\mathbf{x}_i, f(\mathbf{x}_i)]$, is generated by running the ion source at a particular control setting, $\mathbf{x}_i$, the training example is presented to the network. The network uses standard iterative numerical methods to find the parameter values that minimize the cost function. Because the parameters $\mathbf{a}_j$ and $\mathbf{d}_j$ appear linearly in the equation for Φ, they can be optimized by a simple parallel calculation. The centers and widths of the basis function are less easy to optimize, so in the control of the negative-ion source and many other problems, we choose those parameters from initial data (acquired in an exploratory run or previous runs) and leave them fixed. We choose the starting values for the adjustable parameters from a previous run as well. Once the network has found new $\mathbf{a}_j$'s and $\mathbf{d}_j$'s that minimize the cost function, the adjuster finds a new value of the control settings, $\mathbf{x}_{i+1}$, that maximizes Φ. The calculation then proceeds to the next iteration, in which the pair $[\mathbf{x}_{i+1}, f(\mathbf{x}_{i+1})]$ is the new training example. Thus the data are taken near the true optimum settings, where knowledge of the fitness function is most useful. The controller iterates the process of measuring the fitness function, training the network, and

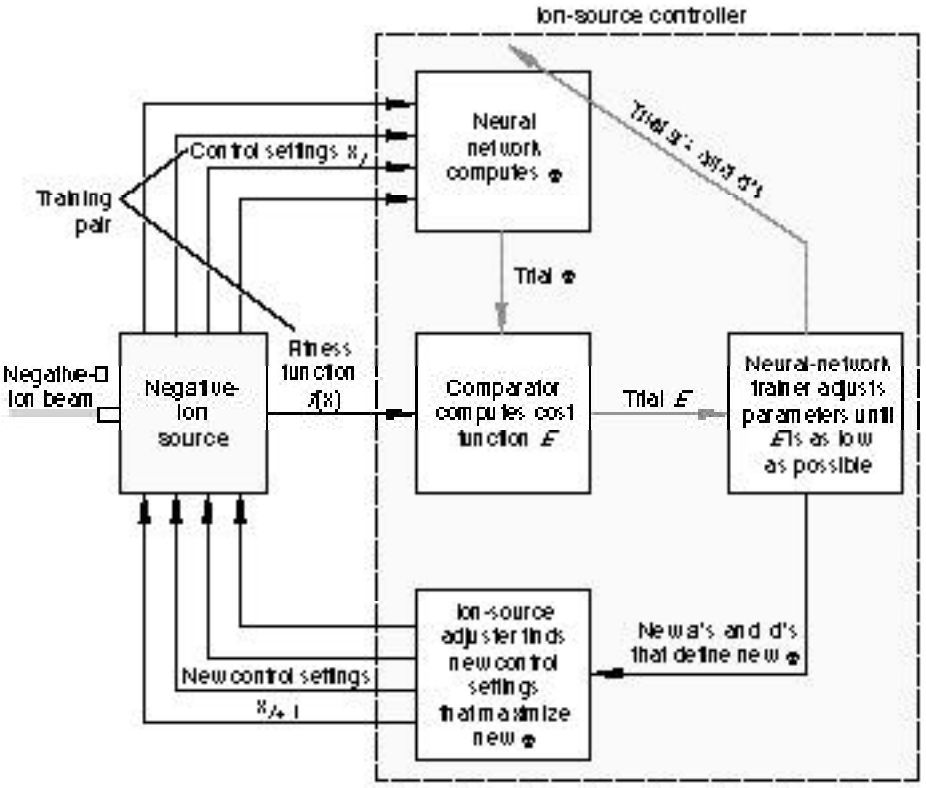

Figure 3. How the CNLS Net Controls the Negative-Ion Source
The figure shows the overall architecture of the negative-ion-source controller. When the negative-ion source runs at control settings $\mathbf{x}_i$, the fitness function (or beam quality) $f(\mathbf{x}_i)$, at those settings is measured and the pair $[\mathbf{x}_i, f(\mathbf{x}_i)]$ is sent to the controller. The neural-network parameters, **a** and **d** (the subscript is omitted for simplicity), are then adjusted by an inner loop (red arrows) as follows. First the neural network calculates $\Phi(\mathbf{x};\mathbf{a},\mathbf{d})$ at all values of **x** at which $f(\mathbf{x})$ is known blah blah blah blah blah. The comparator uses those values of Φ and all the known values of $f(\mathbf{x})$ to calculate the cost function, E. The neural-network trainer, given that value of E, generates new trial values of **a** and **d** intended to give a smaller E. (In many other applications previous training examples are not stored and the parameters are trained by changing them slightly so that Φ reproduces the latest example.) The process is iterated until the parameters minimize E; those parameters, a and d´, are sent to the ion-source adjuster. The ion-source adjuster finds the control settings, $\mathbf{x}_{i+1}$, that maximize $\Phi(\mathbf{x};\mathbf{a},\mathbf{d})$ by the straightforward method of computing Φ at all 7000 possible control settings. The adjuster then begins the next iteration of the outer loop by operating the source at the settings $\mathbf{x}_{i+1}$.

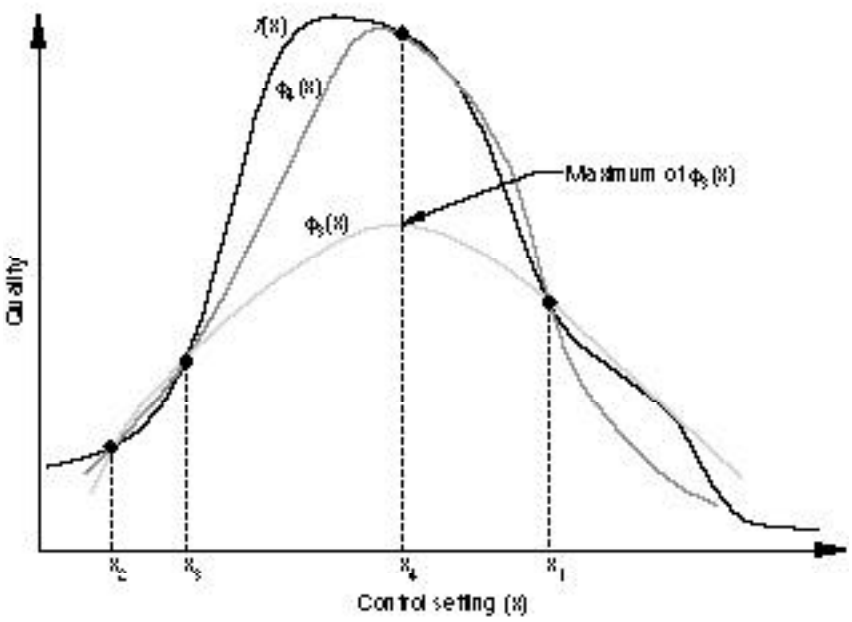

Figure 4. Convergence on the Optimum Control Settings
The graph is a mock example in one dimension of how successive approximations ¡ approach the fitness function $f(\mathbf{x})$ (black) and how the control settings approach their optimal values. Black dots represent training examples, [$\mathbf{x}$, $f(\mathbf{x})$]. The network approximation, Φ_3, produced by the third iteration is shown in gray. The next training example consists of $\mathbf{x}_4$—the control settings that maximize Φ_3—and $f(\mathbf{x}_4)$. A new approximation, Φ_4 (red), is calculated by the neural network to reproduce as well as possible all the training examples, including the latest one. The control settings for the next iteration, $\mathbf{x}_5$, will be the settings that maximize Φ_4. Thus in each step the network is trained and the true optimum value of $\mathbf{x}$ is approached.

locating the best settings at which to measure the fitness function again, approaching the true optimum settings with each step. Figure 4 illustrates the approach to the optimum. Very near the optimum the controller switches to conventional optimization techniques to fine-tune the beam.

During the experimental runs the dependence of f on the inputs continuously changed, but the network successfully tracked the optimum settings as they drifted. In a few instances the machine crashed through no fault of the controller. The controller was able to reoptimize the beam in a short amount of time using the parameter values saved from before the crash, which were generally fairly close to the optimum after the source was restarted.

The CNLS network is the workhorse for many of our applications. There are also other networks and learning algorithms in use. The most popular is known as the multilayer perceptron with backpropagation learning. It is derived from the perceptron, one of the first artificial neural networks, and is more closely based on biological networks than the CNLS network. The multilayer perceptron contains several hidden layers of neurons. When the network is trained, the adjustable parameters of the last hidden layer are adjusted according to the difference between the output and the desired function, f. The parameters of each other hidden layer are adjusted according to the adjustments to the following layer. Thus information about the difference between the network's output and the desired output is propagated from the output layer to the first hidden layer, a process called backpropagation.

We have not yet addressed the issue of function smoothness. Fortunately, this issue did not arise in the negative-ion source problem, although it has arisen in a number of other applications. We will discuss this issue in more detail in the section on future directions.

What Else Can They Do?

The potential value of neural networks is illustrated by the variety of problems that Laboratory researchers are already solving with them. For instance, the success of the negative-ion-source controller suggested that similar networks could be applied to process control in general. In fact, we have projects under way with DuPont to develop for chemical processing a set of controllers based on the ion-source controller. The economic impact of improving the efficiency and reducing the waste products from industrial processing could be tremendous. For example, a group of outside consultants performed a benchmarking effort for DuPont. Their results indicate that improved process control could result in an annual savings of $500 million for DuPont alone. These techniques can also be used within the DOE laboratories. Artificial neural networks

might improve the efficiency and materials accountability of uranium processing. The Laboratory's Nuclear Chemistry and Analysis Group has developed an adaptive scheduler based on the multilayer perceptron that can optimize the work throughput of automated radionuclide-assay equipment.

Artificial neural networks are by no means limited to process control. For instance, the External Information Technology Group has successfully used the multilayer perceptron to classify ordnance that might be found in the field. In that situation often only partial information (such as a piece of a tailfin) is available, but classification must be very reliable.

Many potential applications of adaptive computing involve predicting a point in a time series based on the previous points. In the late 1980s Alan Lapedes and Rob Farber of the Complex Systems Group demonstrated that the multilayer perceptron could predict the behavior of chaotic time series (generated by the logistic map and the Mackey-Glass equation) with an accuracy comparable to that obtained from more conventional methods. The method required significantly more computational resources, however.

In the summer of 1992, researchers in the Inertial Fusion and Plasma Physics Group and the Center for Nonlinear Studies turned to the problem of predicting highly correlated time series in which the data arrive rapidly. They devised a new learning algorithm that was sensitive to the small differences between consecutive data points. The technique was immediately combined with the CNLS net to attack a control problem of concern to everyone. Researchers in the Applied Theoretical Physics Division were already working with the automotive industry on adaptive controllers. The new algorithm allowed them to start developing a controller that can learn and adjust to road conditions in a fraction of a second. Such a controller could significantly improve automobile safety.

Another interesting application, studied in the Computer Research and Applications Group, is the prediction of tongue and mouth motions of speakers given the sounds of speech. This work could be valuable in speech therapy and possibly in speech understanding by computers. The researchers who performed that work received an R&D 100 Award in 1992. Time-series prediction has also been used to predict successfully when floods will occur in Venice Lagoon.

Database Mining. With the increase in the availability of computation has come an explosion in the amount of information that flows around the world and in near-earth space. This flow is overwhelming the humans who are tasked with extracting information from it. It is also severely taxing the communications networks that support the flow. Some projects at the Laboratory are concerned with the construction of information filters which can automatically and adaptively make some rudimentary sense of very large, possibly noisy data streams.

One industry where information filtering is especially important is the banking industry. In the electronic age money flow is reduced to information flow. There are people at the Laboratory who never touch a checkbook. Their paychecks are deposited automatically and their monthly bills are paid automatically. Goods and services are paid for by bits flowing into and out of accounts. U.S. banks could analyze the manner in which money/information flows through their nodes in order to improve products and services and consequently build up the bit counts in their own accounts instead of seeing the bit counts rise in the accounts of their foreign competitors. On a larger scale, policy makers would like to extract information from the flow in order to have better control over the economy. The Laboratory has a project with the banking industry to build adaptive computers for purposes such as forecasting and individual profitability prediction. This technology will be applied to information extraction from arms-control and nonproliferation databases to identify problems of nuclear proliferation.

Overwhelming amounts of data will also be produced by the Superconducting Super Collider. Information will be flowing from the experiments at a rate several times that at which the human visual system receives information. Researchers in the Complex Systems and Fluid Dynamics groups are developing adaptive techniques to quickly sort through this data explosion.

Another source of overwhelming information flow is visual images. Much more imagery is being generated by satellites and other sources than humans can analyze or even store in our memories. Quite often one would like information to be extracted before the data are transmitted. Researchers in the Analysis and Assessment Division and the Inertial Fusion and Plasma Physics Group are developing an adaptive tracking system which can automatically identify and track interesting items in noisy and cluttered images.

Where Are We heading?

The CNLS net approximates smooth functions reasonably well. In many problems, however, the function to be approximated contains discontinuities and singularities. A common way to calculate discontinuous functions is by iteration with feedback. Figure 5 depicts an iterated network with feedback. Such a network has two inputs, one for the input vector **x** and the other for the output of the previous iteration. After a number of iterations (usually five to twenty), the output settles down to a fixed point; that is, when the output is fed back into the network in combination with the input vector **x**, the network gives the same output that was fed in. The fixed point is the result of the entire calculation. One can think of finding a fixed point as stepping downhill on a potential function; a stable fixed point corresponds to a minimum of the function. Therefore a very small difference in the input can produce a large change in the output by affecting which side of a peak the calculation starts on and consequently which valley it ends in. Figure 6 shows that the CNLS network with feedback can closely match a discontinuous function (a fitness function for a free-electron laser beam) that the unmodified CNLS net approximates rather poorly.

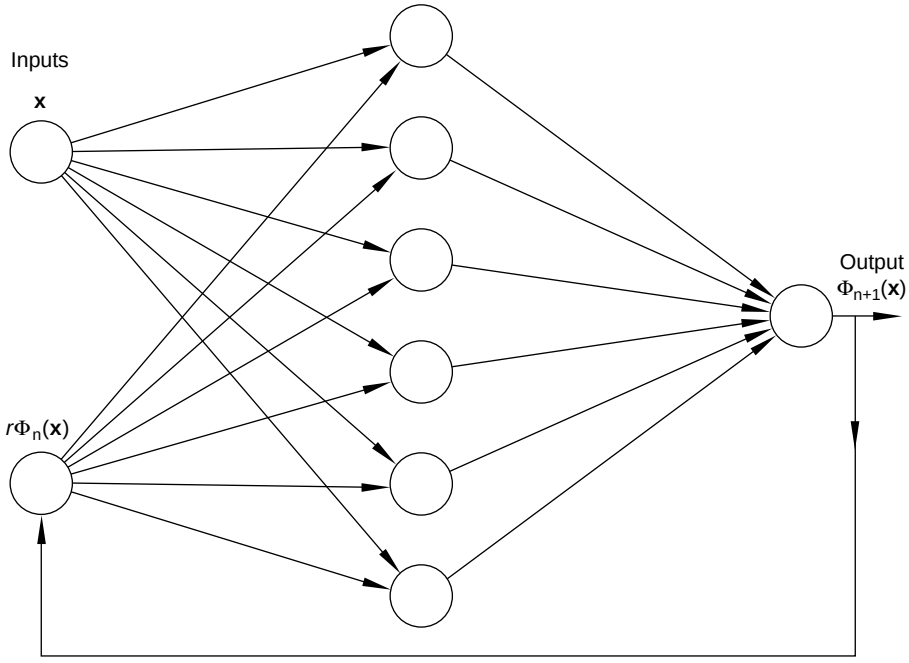

Figure 5. A Network with Feedback
To make an artificial neural network with feedback, we add an input node to the network for each output node. We run the network a number of times, each time using the input vector **x** as input to the original node and the output of the previous run, $\Phi_n(\mathbf{x})$, as input to the new node (or nodes). Before the previous output becomes the input, it is multiplied by a constant, *r*, that is between 1 and 0. We let Φ_0, the input to the first iteration, be the $\Phi(\mathbf{x})$ produced by a version of the network that does not have feedback. We train a network with feedback by giving it, for each training example [**x**, $f(\mathbf{x})$], the inputs **x** and $rf(\mathbf{x})$. The output is trained to be $f(\mathbf{x})$ using the same methods as for networks without feedback. Thus we make each training example a fixed point of the network.

In most of the applications discussed in the previous section, an artificial neural network learns a single-valued function of the inputs, **x**. Quite often, however, real systems can have several states under a single set of conditions, depending on their previous states. Networks with feedback can approximate such multivalued functions. The network is trained so that at a given value of **x**, each of the possible output values is a fixed point. In the first iteration, the two inputs are **x** and a guess supplied by the user. The calculation chooses the fixed point that is reached by going downhill from the initial guess.

To attack still more complex problems, we can generalize the idea of feedback by designing arrays of networks that communicate with each other dynamically. This method permits very sophisticated computation. For instance, a simple array of networks, developed by researchers in the Center for Nonlinear Studies, learned how to control in simulation the balancing of two inverted broomsticks, one on the end of the other. The sticks were confined to move in a vertical plane. The input to each network consisted of the angles and motions of the broomsticks and the outputs of adjacent networks in the array; the output of each network also controlled the motion of a cart to which the bottom end of one broomstick was fixed. The networks were trained only by "punishing" them

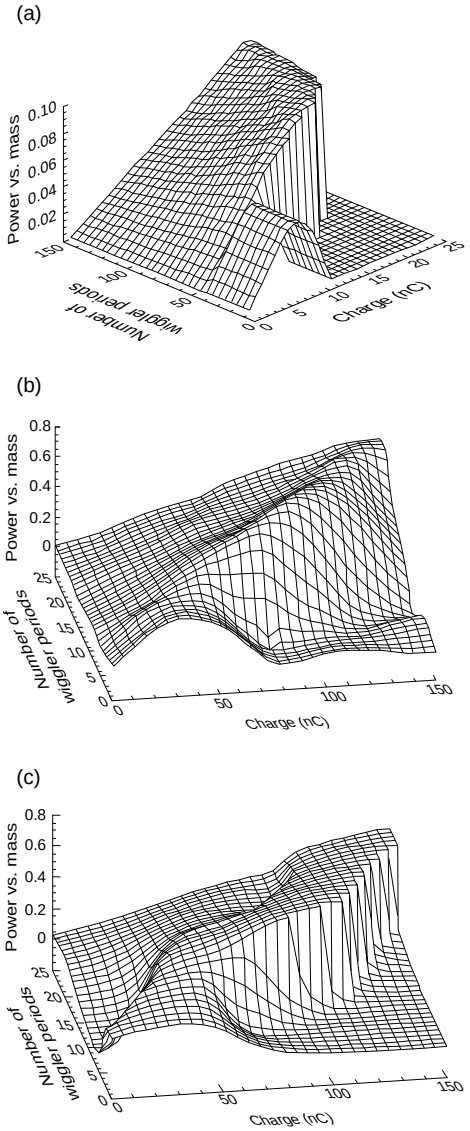

Figure 6. The Effect of Feedback on Network Approximation of a Discontinuous Function

(a) A two-dimensional cross section of a simulated fitness function for the beam of a free-electron laser. The goal is to find the control settings that maximize the fitness function. Since the maximum of the fitness function is very close to the edge of the "cliff," modeling the cliff accurately is important. (b) The output of a CNLS net without feedback trained with the function shown in (a). The control settings that maximize the approximation differ from the true optimal control settings by 20 percent. (c) The output of a CNLS net with feedback after two iterations, also trained with the function shown in (a). The cliff is much better reproduced. The control settings that maximize the approximation are very close to the true optimal settings.

every time the brooms fell. This problem is too difficult for the networks previously discussed, but the networks in the array arrived at sets of parameters that allowed them to balance the broomsticks. Similar arrays of networks will be required for more difficult learning problems such as language processing, image interpretation, large-scale simulation interpretation, and design of experiments with many parameters.

Concluding Remarks

I had an eerie feeling when I watched the network control the negative-ion source. Pumps and heaters turned off and on and voltages were adjusted while four human observers watched. Watching the beam traces appear on the screen was like watching a rat find its way through a maze. When Bill gave the controller the command to shut the machine down, I expected to see a message on the screen like those from the insubordinate computer HAL in Arthur C. Clarke and Stanley Kubrick's *2001: A Space Odyssey*: "Sorry, Bill, I cannot do that. I am in charge now." I have had similar experiences while watching the networks learn in other applications. It is hard not to think about networks anthropomorphically. The field is still in an early stage of development and it is not clear yet what the capabilities and limitations of machine learning will finally be.

Further Reading

G. M. Bruce, K. Lee, L. A. Lee, W. C. Mead, M. K. O'Rourke, L. E. Thode, Y. C. Lee, G. W. Flake, and I. J. Poli. 1990. Nonlinear adaptive networks: A little theory, a few applications. Los Alamos National Laboratory unclassified release LA-UR-90-273.

R. Hecht-Nielsen. 1989. *Neurocomputing.* Reading, Mass.: Addison-Wesley.

J. Hertz, A. Krogh, and R. G. Palmer. 1991. *Introduction to the Theory of Neural Computing.* Reading: Addison-Wesley.

R. D. Jones, Y. C. Lee, C. W. Barnes, G. W. Flake, K. Lee, P. S. Lewis, and S. Qian. 1990. Function approximation and time series prediction with neural networks. In *Proceedings of the International Joint Conference on Neural Networks, San Diego, California, June 17–21, 1990. Volume 1.* New York: IEEE.

P. D. Wasserman. 1989. *Neural Computing: Theory and Practice*. New York: Van Nostrand Reinhold.

Roger D. Jones earned a Ph. D. in physics from Dartmouth College in 1979. He has worked at the Laboratory as a staff member since then. He worked for ten years on inertial-confinement fusion as a designer and physicist. He has made various theoretical contributions to adaptive computation, including participating in the design of the CNLS network as well as studies in image processing and making inferences from databases. He is a member of the Executive Committee of the Center for Nonlinear Studies at the Laboratory. Roger lives in Española and plays bass with local jazz bands.