Proceedings of the 28th Conference
on Decision and Control
Tampa, Florida • December 1989

NEURAL NETWORKS FOR CONTROL AND SYSTEM IDENTIFICATION

Dr. Paul J. Werbos
Room 1151, National Science Foundation*
Washington D.C. 20550

ABSTRACT

Artificial neural networks (ANNs) are now being explored for a wide variety of applications, some of which have already achieved commercial success. The National Science Foundation (NSF) program in Neuroengineering places particular stress on ANNs for control applications (neurocontrol) and for systems identification (neuroidentification) [1].

Neurocontrol is not an alternative to the broad disciplines of control theory and decision analysis. In fact, it may be understood as an extension or development of those fields to deal with a family of large, sticky problems which tend to require approximations and experiments rather than exact solutions and ironclad guarantees of success. Control theorists have much to contribute to this emerging field.

This paper first reviews the field of neuroengineering as a whole, highlighting the importance of neurocontrol and neuroidentification. Then it describes the five major architectures in use today in neurocontrol(in robotics, in particular) and a few areas for future research. It will conclude with a few comments on neuroidentification.

Most of the concepts here have been discussed already elsewhere, along with lengthy discussions of the possible applications[2,3]. This paper will cite that earlier work, but will try -- whenever possible -- to discuss the issues from the viewpoint of control theory.

A REVIEW OF NEUROENGINEERING

What is Neuroengineering?

At the National Science Foundation (NSF), the term "neuroengineering" has been given two different definitions, illustrated in Figures 1 and 2. The two definitions appear quite different, at first, but they usually end up being the same in practice. Both definitions focus on the problems which neuroengineering tries to solve; there is no claim that the mathematical concepts used to solve these problems are magically different from the concepts used to solve other, related classes of problems.

Figure 1. Conventional Definition of Neuroengineering

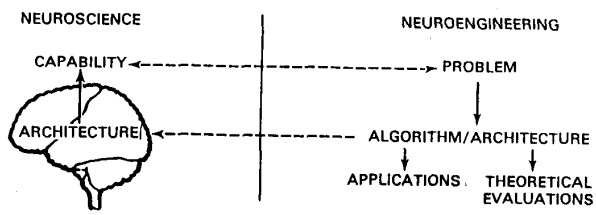


Figure 1 gives the more conventional definition. Neuroengineering is that field of research which tries to copy over known capabilities of the brain into computer hardware and software. More precisely, it tries to develop mathematical designs which could be embodied directly in hardware or simulated in software; usually, it is better to do the testing in software.

In developing these designs, neuroengineering tries to make use of what is known about how the brain achieves its capabilities. There are actually two schools of thought here. One school closely follows only what is now known about the brain, and looks for "emergent computational properties." This may be called the bottom-up or biologically-based school. The other school treats our knowledge of the brain as a very loose design constraint, and focuses on the desired capabilities, making heavy use of statistics and control theory. This may be called the "top-down" school. I will return to this distinction later on.

Neuroengineering evaluates these designs through simulation tests, practical applications, and abstract mathematical analysis. Once we know what designs really work, we can then go back to the biologists and give them a better starting point in devising theories which actually reproduce or explain the capabilities of the brain. This kind of feedback from the engineering to the biology may be crucial to our ability to understand the human brain, in the long-term future.

Figure 2. Major Areas of Neural Network Research

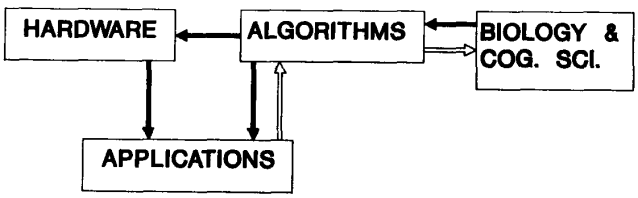


Figure 2 leads to a different definition, emphasizing computer hardware. The Neuroengineering program at NSF is actually an outgrowth of the Lightwave Technology program, which supported optical computing. Leading figures in the area of optical computing (such as Psaltis of CalTech and Caulfield of Alabama) told us that they could achieve a thousand- or million-fold improvement in the throughput of computers, compared with the best digital parallel designs. Experts in VLSI design, like Carver Meade, told us that analog parallel VLSI has a similar potential. But in both cases, there was a catch: the high throughput required highly parallel computers limited to simple repetitive operations (not 100-operator instruction sets with jumps included!).

The skeptics argued that very few computational tasks can be done with such an architecture. The advocates pointed out that the human brain can perform a very wide range of tasks, within the constraints of such an architecture. The neuroengineering program was born as an attempt to exploit our knowledge of the brain, in order to support the future of optical computing and analog VLSI. Without such support, our work on advanced hardware could fall into a trap which is very familiar to the computer business: the seduction of big, fast black boxes which perform no useful work because the software/algorithm aspect was neglected.

Even today, the algorithm development part of Figure 2 (the true neuroengineering component) receives far less support than the hardware or applications components. About 90% of the current applications are based on two algorithms, originally developed in the 1970s (backpropagation[4] and Hopfield nets[10], the latter of which was studied by Stephen Grossberg in earlier years[5]). The International Neural Network Society has over 3,000 members, most of them full-time researchers in the U.S., but the NSF Neuroengineering

* This paper reflects the personal views of the author, not the official views of NSF. Some portions have appeared elsewhere [2,4] in the public domain.

program -- the only program fully dedicated to the algorithm side -- has a budget of only $1.2 million. Similar sums of money are spent on new algorithms out of the larger applications-oriented programs at the Department of Defense, but Japan and Europe are scaling up to much larger efforts than the U.S. total. Congress is currently debating a supercomputer bill which would add an extra $4 million/year to the NSF program, among other activities.

History and Prospects of Neuroengineering

Figure 3. Partial History of Neuroengineering

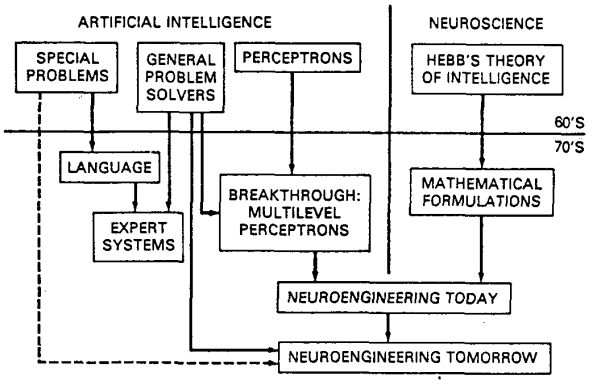


Figure 3 provides a simplified history of neuroengineering. Research on neural networks has gone on for decades, at a relatively low level. The birth of the field as a self-conscious, organized discipline should probably be dated at June, 1987, when the IEEE San Diego chapter hosted the first International Conference on Neural Networks (ICNN). Everyone was surprised when almost 2,000 people showed up. Along with the surprise came a lot of short-lived hyperbole, which then drew a negative reflex reaction from many other scientists.

The hyperbole in 1987 was actually quite tame compared with the hyperbole back in the 1960s, when another new discipline -- artificial intelligence (AI) -- was born. Many of us can remember when "Thinking Machines" showed up on the cover of Time magazine, etc. In the beginning, AI focused on one basic question: "How can we build intelligent systems?" In other words, how could any well-specified mathematical system -- made up of neurons or anything else -- achieve the kind of "intelligence" we see in human beings?

There were three different ways of trying to answer this question, which led to three different strands of research. On the one hand, some people tried to concentrate on solving specific problems which are generally considered to require intelligence. For example, they developed algorithms and computer programs to play better and better games of chess. Some of the early work on these lines -- like Samuel's famous checker-playing program[6] -- developed some very important general insights, but after awhile the field began to get cluttered with hundred-page assembly-language programs specialized to narrow applications.

On the other hand, a second school of thought -- the general problem-solving school -- argued that there is no intelligence in these kinds of specific algorithms. They argued that intelligent systems produce algorithms in much the same way that a clam grows a clamshell. The life is in the clam, not in the shell. Likewise, the intelligence is in the system which learns and develops the algorithm, not in the algorithm itself. Among the pioneers of this approach were Newell, Shaw and Simon.

The theory of this approach was very good, but in practice it ran into a very large obstacle. How do you design a generalized machine to do anything, without specifying anything at all about what it is supposed to do? So as a practical matter, the people working in this school also had to specify specific mathematical problems, problems which they hoped would be generic enough in character to approximate what the brain does. They mainly focused on two such problems: first, the problem of generalized symbolic reasoning -- a problem which people are still working on today; and second, a problem which people are now calling reinforcement learning.

In reinforcement learning, we ask a system to control some kinds of overt actions -- like physical movements -- over time, so as to maximize some measure of performance or reinforcement. All of these words could be discussed at great length, but let me focus on two of them for now: "over time." There is a lot of work in robotics which focuses on the link between actions at any time and results at the same time -- the kind of connection which you worry about in posture control or hand-eye coordination, for example; however, the hard part of this problem -- the core of the design problem -- is in accounting for the effect of actions at one time on results at a later time.

Some psychologists believe that human intelligence really is a matter of reinforcement learning, almost entirely. Other people -- like myself -- believe that the human mind does possess other attributes which go well beyond reinforcement learning in higher brain centers. In some of my papers[7,8], I have even tried to discuss some of these attributes. Nevertheless, reinforcement learning clearly is part of what the human brain does, and it is a worthwhile starting point in trying to understand the overall structure of the brain.

A third school of thought -- the perceptron/ADALINE school -- tried to copy over what was known about neurons in the 1950's and 1960's. In those days, people thought that they knew how neurons worked, more or less, following the classic model of McCulloch and Pitts. People strung together networks of McCulloch-and-Pitts neurons, and tried to develop learning rules to make these networks perform useful functions. The key innovators in this school were Widrow and Rosenblatt. Widrow is currently President of the International Neural Network Society (INNS); even his older work is still of current relevance, including the ADALINE-based "adaptive filters" used in billions of dollars worth of communications equipment[9].

None of these schools of thought were able to live up to all of the early hype about AI, especially not the part about building true intelligence within 5 to 10 years. People discovered very quickly that you can't build an intelligent system by mindless hacking or by using a few quick tricks. As a result, many of the manic-depressive types were bitterly depressed. Exposes of AI started to appear in various places, and government funding was cut. Yet steady progress was made, through quiet research. In the 1970s, for example, I developed the method now called backpropagation, which solved the previously intractable problem of adapting multilayer neural networks. About half the mathematical tools of neuroengineering originated in the AI tradition, and half in the neuroscience tradition. Physicists and cognitive scientists played a crucial role in using and popularizing these tools, at first, and then in developing them further[10,11], with acknowledgement of some of the prior work[12,13].

What are the prospects for neuroengineering over the next ten or twenty years? On the negative side, there has been some hyperbole and some overreaction to

the hyperbole, just as there was with AI. Also on the negative side, there has yet to be a full synthesis of the bottom-up and top-down traditions; as the general systems movement discovered twenty years ago, it is much easier to get two schools of thought into one room than it is to get them to really hear each other. On the positive side, neuroengineering has already developed enough mathematical tools to serve a wide variety of niche markets, comparable to the markets for expert systems. Also on the positive side, neuroengineering has inherited a majority of the people who are still addressing that fundamental question:"How can we build intelligent systems, systems which display the full range of learning and adaptation that organic brains demonstrate?" This question, though difficult, is of fundamental and enduring importance; this insures the long-term importance of neuroengineering, whatever the level of government support.

Branches of Neuroengineering

Neuroengineering is a complex, interdisciplinary field, which makes it difficult to slice up into subdivisions. In this paper, I will slice it up based on which brain capability is being copied into computers.

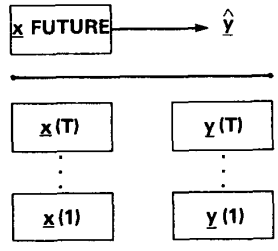
Different researchers have tried to emulate many different capabilities of different brains. For example, some researchers study the individual cells in the brain of an adult bat which give it a sonar processing capability; by duplicating the exact pattern of connections, weights and parameters, they hope to develop a robust sonar capability for military applications. Other researchers study image processing, speech processing, and motor coordination in a similar way. Jasper Lupo of DARPA has described how house flies, cockroaches and many other organisms have specific capabilities of great military interest.

The mainstream of neuroengineering, however, studies the ability of brains to learn such processing capabilities. They try to develop generalized learning systems, which can then be applied to a wide variety of specific applications in a wide variety of ways. The commonest learning systems are supervised learning systems (SLSs), which are given a history of inputs and desired outputs; they adjust their internal parameters (connection weights) so as to make their actual outputs as close as possible to the desired outputs, over the past history. SLSs can be used to classify patterns, by letting the inputs be the patterns to be classified and the desired outputs be the correct classifications. Less common as yet are reinforcement learning systems, where the system is given a gross reward or punishment for its outputs but is not told what its outputs should have been. There are also many types of unsupervised learning systems, which somehow learn something simply by observing their environment over time; such systems are typically designed to act as associative or content-addressable memories, as feature detectors for use in pattern recognition, or as forecasting networks.

In between the research into learning and the research into specific capabilities, there is a third strand of research, which grew out of the effort to understand differential equation models of neural networks. John Hopfield of CalTech[10] stimulated a large effort by physicists and others to build networks capable of minimizing a complex quadratic or similar "energy" functions. Many of the people studying these topics are still interested in the broader questions of learning and memory, but many have started to build devices to solve function minimization problems, for use in fields like combinatorial optimization or conventional image processing.

Neuroengineering in general has many important connections to statistics and control theory. For example, the problem of supervised learning is almost identical to the classic problem of nonlinear regression, as symbolized in Figure 4.

Figure 4. Supervised Learning = Nonlinear Regression?



In supervised learning, the ANN implements a mapping from a set of independent variables, or "inputs", which form a vector $\underline{X}(t)$, to a set of dependent variables or "targets," $\underline{Y}(t)$. The ANN contains a set of parameters called "connection weights," forming a vector $\underline{W}$. In supervised learning, we estimate or adapt the parameters $\underline{W}$ so as to minimize the differences between the outputs of the network, $\hat{\underline{Y}}(t)$, and the target outputs, $\underline{Y}(t)$, over the training set. The training set consists of T pairs of vectors $\underline{X}(t)$ and $\underline{Y}(t)$. We try to minimize error in such a way that the network will give good predictions of $\underline{Y}$ for new vectors, $\underline{X}$, outside of the training set.

Many people believe that the problem of learning a mapping from $\underline{X}$ to $\underline{Y}$ is so simple and so well-understood that there is little room for further research. The problem of adapting an ANN is simply a special case of the more general problem; the existing general methods can be used directly on this problem. Based on maximum likelihood theory, and a few quick assumptions, they argue that we should always try to minimize the square error between $\underline{Y}$ and $\hat{\underline{Y}}$, using numerical methods such as those reviewed by Dennis and Schnabel[14].

There are at least two reasons why this is not the full story.

First of all, minimizing square error is not a trivial mathematical problem, if one is working with a large network or if one is interested in fast real-time learning. People working in adaptive control have learned that the adaptation of parameter estimates in real time, in response to current data, is very different from the problem of analyzing a static database; their insights may be very important to the field of neuroengineering[15]. The classical methods generally require you to calculate the derivatives of error with respect to all of the parameters; backpropagation -- one of the pillars of neuro-engineering -- is essentially a method for calculating all of the required derivatives, in a single pass through the system, applicable to any differentiable network or model[17], ANN or other. Experts on numerical optimization often stress that knowledge of a particular problem is important in speeding up convergence; therefore, numerical methods specific to ANNs as such can be very valuable, even though ANNs can represent essentially any mapping (as proven by Hal White of UCSD). In summary, the effort to speed up convergence[4,16], minimize costs, and allow for parallel implementation are central to the research in neuroengineering. These may not sound like glamorous issues, but the difference between $O(N^2)$ cost and $O(N)$ cost may decide whether we really do have the capacity to handle very large models and networks, even in the age of supercomputers. Some of the problems here are very fundamental, and of real intellectual interest.

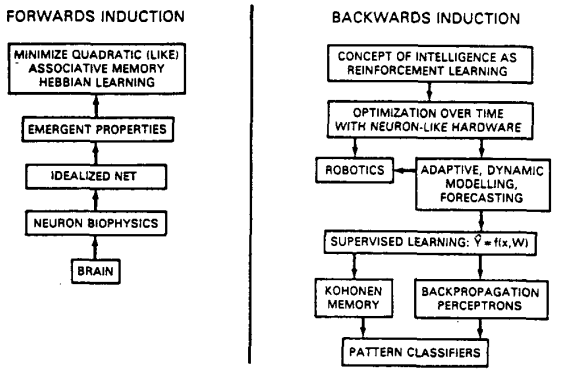
Second of all, minimizing square error is not a

universal panacea. Statisticians have known for decades that there are problems with least squares estimation, and they have developed a wide variety of methods to help cope with these problems. These problems are very important in real-world applications, although a lot of ad hoc tricks can be used to get around them[18]. In an automated learning system, ad hoc tricks and a reliance on human tweaking are not good enough.

In some applications, the best approach to the supervised learning problem is to build a "network" which matches any new vector, $\underline{X}$, to the closest vector $\underline{X}(t)$ in the training set; the network then outputs $\underline{Y}(t)$ as its prediction of $\underline{Y}$. "Heteroassociative memories" are ANNs which behave in this general sort of way -- explicitly or implicitly -- as a way of solving the supervised learning problem. Research is needed to develop systems which automatically blend this approach, the least squares approach, and the notion of interpolation, in response to the current experience of the ANN as it learns[4,8].

The links between control theory and neuro-engineering are even stronger in the case of reinforcement learns and ANNs for forecasting purposes. In fact, that very link was what led me down to the idea of backpropagation in the first place, as symbolized on the right-hand side of Figure 5. The NSF program in neuroengineering places special weight on efforts to go back up to the original goal, of building true intelligent systems, in a three-pronged effort: (1) building and demonstrating neural nets for control applications; (2) developing improved theory for better forecasting networks (stressing robustness over time, true nonlinear filtering and real-time learning); (3) improving the stock of SLS designs. The rest of this paper discusses the first of these efforts, the effort which is now receiving highest priority.

Figure 5. Conceptual Map of Neuroengineering



NEURAL NETWORKS FOR CONTROL

Overview

In October 1988, NSF sponsored a workshop on neural networks for robotics and control at the University of New Hampshire. A small number of key researchers were invited from the neural network community, the control theory community, and the applications communities. This section will summarize my views of that workshop, which is described in more detail in an earlier paper by myself[3] and in far more detail in a forthcoming book by Miller, Sutton and myself (which includes [15]).

The human brain itself -- as a whole system -- is a neurocontrol system. Its biological function is to control physical actions over time, so as to achieve results in an external environment. From an engineering point of view, the brains of mammals are living proof of the feasibility of achieving the following capabilities in an integrated control system:

- o A design which can be implemented in analog, parallel hardware
- o Adaptation in real time to new environments, without instability
- o The ability to handle severe nonlinearity and noise
- o The ability to "plan" or optimize over time, as required with complex tasks
- o The control of millions of actuators in parallel
- o Coupling between a slower, more comprehensive controller and a faster subordinate system, so as to achieve smooth high-speed control

Every one of these capabilities is addressed in the forthcoming book by Miller, Sutton and myself. Working examples of these capabilities -- not just simulations -- have been demonstrated for all but the last two, in various combinations. Methods for handling large numbers of variables have also been worked out, on the mathematical level. William Hutchinson of BehavHeuristics Inc., of College Park Maryland, has developed a commercial application involving large numbers of variables, but I have not seen a publication of the details. Given so much progress in a short period of time, it is not unreasonable to expect that these capabilities could all be combined together in working systems over the next decade or two, if we made a serious effort.

Capabilities of this sort could have enormous applications, in four general areas:

- o robotics and plant control
- o vehicle and structure control
- o teleoperation and aid to the disabled
- o management of communication networks, computer networks and social science models

Five Approaches to ANN Design

In the forthcoming book, the robotics area is used to illustrate three of the five basic designs now used in neurocontrol work.

In supervised control systems, the computer is given a history of what the robot sees (i.e., sensor readings) and of what a human tells it to do. Supervised control is a bit like the old pendant system used to train robots. In the pendant system, the pendant records what a human wants the robot to do, step by step; the robot repeats the desired sequence of actions exactly, every time. In supervised control, the neural net learns how the desired actions vary as a function of the sensor inputs; it learns the mapping from the sensor readings to the desired outputs. In general, supervised controllers are analogous to expert systems; they are a way of transferring knowledge from a human to a robot. In supervised control, the computer watches what the human does, not what he says. Widrow of Stanford was the first to use supervised control. Fujitsu has recently built real robots based on this principle, which they have begun to show at various trade shows.

In direct inverse control, the robot learns how to follow paths of motion laid down by a human being or an AI planning system. The problem is to learn the mapping from spatial coordinates (in which the trajectory is given) back to the actuator signals (like the angles of various joints). If this is a 1-to-1 mapping, then any Supervised Learning System (SLS) can

be used to learn the mapping. Miller of New Hampshire has developed a VLSI chip which embodies this design, and achieves position errors well under a percent with actual robots -- a performance good enough to be of real commercial interest.

In backpropagation through time, a neural net learns to minimize or maximize any performance criterion laid down by the user or designer, so long as the user can first develop a model of the system to be controlled. Jordan of MIT built a simulated robot arm based this design, an arm which can follow a desired path of motion as smoothly as possible, even when there is no 1-to-1 map from the spatial coordinates to the actuator coordinates. He first crafted a performance criterion representing the deviation from the desired trajectory to the actual trajectory, plus a measure of the smoothness of the motion. Then, after adapting a network to describe the dynamics of the arm, he used backpropagation through time to adapt the control network. Kawato of ATR (Japan's Bell Labs) has used a similar approach to build systems which calculate the optimal trajectory, follow it, and match concurrent biological experiments at the University of Tokyo.

Backpropagation through time as a control method is essentially equivalent to the calculus of variations, as described by Bryson and Ho. As in the calculus of variations, we must have a deterministic model of the system to be controlled. That model must be differentiable. The only new feature is that the derivatives can be calculated more efficiently, if the system to be controlled is a large, sparse structure; this allows the efficient handling of larger problems [17,19], and it avoids the need for users to understand Lagrange multipliers. Also, instead of solving for an optimal schedule of actions, we may choose to optimize the weights in an ANN which generates the actions; superficially, this sounds suboptimal, but it may lead to a more robust control strategy in practice. Because the derivative calculations go backwards from a final time, T, to an initial time, this method cannot be used directly when true real-time learning is required; however, a number of approximations have been suggested, of varying quality.

The two other major designs are neural adaptive control and adaptive critics. Neural adaptive control is like conventional adaptive control, but with neural nets replacing some of the usual linear mappings. Narendra of Yale -- a recognized expert on conventional adaptive control -- is working on this approach[15], as is Cotter of Utah. Adaptive critic systems are a more complex subject. (Franklin of GTE has applied them to robotics, as reported last year at IEEE/CDC.)

The adaptive critic class of designs is the only class which tries to perform optimization over time, allowing explicitly for the possibility of noise, and also allowing for true real-time learning. Adaptive critic designs may be defined as designs which try to approximate the Bellman equation of dynamic programming. The search for good approximations to dynamic programming is one of the key areas where neuro-engineering, control theory and numerical analysis have a lot to learn from each other. The phrase "adaptive critic" was coined by Barto, Sutton and Anderson, but closely related ideas were published in earlier years by myself, by Harry Klopf, and -- from a more biological perspective -- by Grossberg and Levine.

The Bellman equation has been written in many forms, suitable for different classes of problem. One simplified version, related to the work of Howard, is:

$$J(\underline{R}(t)) = \max_{\underline{u}(t)} \langle U(\underline{R}(t),\underline{u}(t)) + J(\underline{R}(t+1)) \rangle,$$

where the "<>" denotes expectation value, where $\underline{R}(t)$ refers to the state of the system to be controlled at time t, where $\underline{u}(t)$ refers to the actions taken immediately after $\underline{R}(t)$ is known, where U is the utility function to be maximized over time in the long-term, and J is the secondary utility function which we are trying to solve for. The key theorems in dynamic programming prove that maximizing $U(t)+J(\underline{R}(t+1))$ leads to exactly those actions which maximize $U(t)+U(t+1)+\ldots+U(\infty)$.

In neuroengineering, almost all of the successful uses of adaptive critics to date involve a 2-network design. One network, the Critic, inputs $\underline{R}$ and outputs an approximation to J. The other network, the Action network, inputs $\underline{R}$ and outputs $\underline{u}$. The Critic is adapted so as to make its output fit the Bellman equation. The Action network is adapted so as to maximize $U(t)+J(t+1)$. Several consistency proofs and working examples have come out in the past year or two.

Unfortunately, the 2-net designs have serious limitations. For example, a single, gross signal of reward or punishment -- J -- does not provide detailed feedback to the various components of action which underlie that signal. An adaptation scheme of this sort may converge to the right answer eventually, but the rate of convergence may be astronomically low if $\underline{u}$ has many components. One way to overcome this problem is to use an action-dependent critic, which estimates:

$$J'(\underline{R}(t),\underline{u}(t)) = \langle U(\underline{R}(t),\underline{u}(t)) + J(\underline{R}(t+1)) \rangle$$

(Substituting back into the Bellman equation gives a recurrence relation for J'.) One may then use the derivatives of J' with respect to $\underline{u}(t)$ -- calculated by backpropagation -- as the basis for adapting the Action network. Lukes, Thompson and myself at NSF have recently made this work in simulations. Another approach (BAC) is to build an explicit stochastic model of the system to be controlled, and use backpropagation through that model to get the derivatives of J(t+1) with respect to $\underline{u}(t)$ [3].

There are further challenges in trying to adapt the Critic itself. If the Critic has only a scalar output, then the error feedback in each time period is again just a scalar -- unless one exploits additional information. In earlier papers, summarized in [3], I have specified two designs which use additional information: (1) a network which estimates the derivatives of J rather than J itself (DHP); (2) a network which uses the derivatives of J in adapting a J-type Critic (GDHP). Considerable work will be needed to test these methods and bring them into practice. Both of these methods require the use of stochastic model of the system to be controlled.

A key assumption in dynamic programming is that $\underline{R}(t)$ is governed by a Markhov process. In other words, $\underline{R}$ must represent the actual state of the system to be controlled, and not just a current vector of sensor inputs. All of these neurocontrol methods will be affected by our ability to adapt other ANNs which output such a vector $\underline{R}$.

A Comment on Applications

The workshop in New Hampshire pointed towards many very important potential applications of neuro-control[2,3]. Unfortunately, the page limits here permit only a very quick summary. In robotics, more adaptive robots could reduce startup costs and increase flexibility, thereby helping overcome the key factors which have slowed the robotics market (and improvements in productivity) in the U.S. economy. The control of more difficult processes may be crucial to technologies in areas like biotechnology and the National Aerospace Plane, where difficulties in the control of large, noisy, nonlinear systems might possibly be a "show stopper." ANNs provide much-needed "team B" insurance.

Possibilities to aid the disabled or improve teleoperation also exist, based on efforts to imitate the human cerebellum. (The latter may be crucial to cost-effective utilization of lunar materials and the

eventual human settlement of space, according to reports from the Space Studies Institute.) Applications to economic or global modeling already exist, and can be extended further. Most of these possibilities are being actively studied by serious substantive experts in the various applications, but the area is very far from saturated. In summary, the potential for application goes well beyond the narrow niche markets one finds for many other new technologies.

COMMENTS ON NEUROIDENTIFICATION

Control theory usually requires that we understand a system before we try to control it. It requires an explicit model of the system, either deterministic or stochastic. From the above, it should be clear that the quality of neurocontrol will also be linked to the quality of our systems models. Even when an explicit model is not needed -- as in most 2-net adaptive critic designs -- the quality of results depends on developing a vector $\underline{R}$, which is basically a problem in nonlinear filtering, which comes back to system identification.

Given a vector of observables, $\underline{X}(t)$, and actions $\underline{u}(t)$, it is easy to adapt a network which inputs $\underline{X}(t)$ and $\underline{u}(t)$ and outputs a forecast of $\underline{X}(t+1)$. One can use any supervised learning system to do this. The problem with this approach is that it only works if we observe everything; in other words, the vector $\underline{X}(t)$ has to contain all the information we need in predicting $\underline{X}(t+1)$.

This year, the neuroengineering community has settled on a more sophisticated approach. In this approach, we build a two-part network, including a vector of intermediate calculations $\underline{R}(t)$. One part of the network inputs $\underline{R}(t-1)$ and $\underline{X}(t)$, and outputs $\underline{R}(t)$. The other part inputs $\underline{R}(t)$, $\underline{X}(t)$ and $\underline{u}(t)$, and outputs a prediction of $\underline{X}(t+1)$. The entire network is still adapted so as to minimize square error in predicting $\underline{X}(t+1)$. It is straightforward to work out all the derivatives required in this minimization problem, using backpropagation through time[19]. (Again, there are real-time approximations available[19].)

The two-part approach is adequate for many applications, but it has two major limitations. First, it imposes a stochastic model in which noise is assumed to be uncorrelated across different components of $\underline{X}$. Thus it cannot even represent the usual vector-ARMA processes assumed in Kalman filtering. Second, like any maximum likelihood method, it may have problems with robustness over time; it has no way of exploiting the hidden orderliness of nature, which typically makes it possible to forecast over long time-periods better than one would expect from extrapolating short-term fluctuations. These are serious problems in both theory in practice, but the details are too complex for here[8].

The problem of allocating noise -- even in a maximum likelihood context -- is daunting enough in itself. The challenge here is to approximate true nonlinear filtering, in a global way which goes beyond simple linearization. A special case of this problem is to adapt an ANN to input $\underline{X}(t)$, output $\underline{R}(t)$, and output a "forecast" or simulation of $\underline{X}(t)$ based on $\underline{R}(t)$, such that the noise on $\underline{R}$ is diagonalized. Even this problem is very tough in the nonlinear case. Foldiak (IJCNN) and Liu report results for the linear case, but the extension to nonlinear networks is far off. Schemes for "encoders" and "decoders" have been suggested by myself and others [4,8], but they seem to have deep flaws in the nonlinear maximum likelihood case. "Competitive learning" schemes may solve the problem when each possible state of $\underline{R}$ is represented by a different cell, but for complex vectors this is unrealistic. In general, this may be an excellent research topic for creative and brilliant researchers, who understand stochastic processes well; however, it may be basically an unsolvable problem, which the brain avoids by using a short cycle time (0.1 second) and other tricks. It would be dangerous to jump to conclusions either way, for years to come.

Robustness over time is even harder to analyze on a formal basis, but the empirical evidence for its importance is strong[8,18], and there are working methods -- albeit ad hoc -- to deal with it. For example, the "first pass" of Kawato's cascade method is a special case of the "pure robust" method which I described in 1977[8,18]; apparently, this general approach is often used in robotics in Japan. Related approaches -- based on filtering out high-frequency noise -- have been tried by control theorists. Here again, it is clear that the cutting edge of control theory and of neuroengineering are confronting essentially the same problems, and have much to learn from each other.

REFERENCES

1. NSF, Neuroengineering Program Announcement, NSF 89-26.
2. P.J.Werbos, Neural Networks for Robotics and Control, WESCON/89 Proceedings. Los Angeles: WESCON, 1989.
3. PJW, Backpropagation and neurocontrol: a review and prospectus. Proceedings of the International Joint Conference on Neural Networks (IJCNN). IEEE, 1989.
4. PJW, Backpropagation: past and future. ICNN Proceedings, IEEE, 1988. Script available from me.
5. Grossberg, Nonlinear neural networks, Neural Networks, Vol.1, No.1, 1988.
6. A.Samuel, Some studies in machine learning..., IBM J. Res. Develop., Vol.3, p.210-229, 1959.
7. PJW, Generalized information requirements.., SUGI-11. Cary, NC: SAS Institute, 1986 (&revision from me).
8. PJW, Learning how the world works, IEEE SMC Proceedings November 1987.
9. J.Denker (Bellcore), Rank Prize presentation (U.K.)
10. J.Hopfield&Tank, Computing with neural circuits: a model, Science, vol. 233, p.625-633.
11. D.Rumelhart&McClelland, Parallel Distributed Processing. MIT Press, 1986.
12. D.Parker, Learning Logic. MIT Center for Computational Economics and Statistics, 1985.
13. LeCun, "Une..", Proceedings of Cognitiva 85, June '85.
14. Dennis&Schnabel, Numerical Methods for Unconstrained Optimization & Nonlinear Equations. PrenticeHall, '83.
15. Narendra, Adaptive control using neural networks. In Miller, Sutton & Werbos (MSW), Neural Networks for Control. MIT Press, forthcoming (fall 1989).
16. D.Shanno, Recent advances in numerical techniques for large-scale optimization. In MSW[16].
17. PJW, Maximizing long-term gas industry profits in two minutes in Lotus. IEEE Trans. SMC. March-April 1989.
18. PJW, Econometrics: theory versus practice. Energy Journal, forthcoming special issue by EMF, Stanford.
19. PJW, Backpropagation through time. Submitted by invitation to IEEE Proceedings special issue.

BRAIN VERSUS ANN

	Many Motors	Noise	Long-Term Optimum
Supervised or Inverse Control	X	X	
Backprop Thru Time	X		X
Adaptive Critic:			
2-Net		X	X
BAC	X	X	X