Neurocontrol and Fuzzy Logic: Connections and Designs

Paul J. Werbos
National Science Foundation,[*]
Washington, D.C.

ABSTRACT

Artificial neural networks (ANNs) and fuzzy logic are complementary technologies. ANNs extract information from systems to be learned or controlled, while fuzzy techniques most often use verbal information from experts. Ideally, the two sources of information should be combined. For example, one can learn rules in a hybrid fashion and then calibrate them for better whole-system performance. ANNs offer universal approximation theorems, pedagogical advantages, very high throughput hardware, and links to neurophysiology. Neurocontrol – the use of ANNs to directly control motors, actuators, etc. – uses five generalized designs, related to control theory, that can work on fuzzy logic systems as well as ANNs. These designs can copy what experts do instead of what they say, learn to track trajectories, generalize adaptive control, and maximize performance or minimize cost over time, even in noisy environments. Design trade-offs and future directions are discussed throughout. The final section mentions a few new ideas regarding reasoning, planning, and chunking, with biological parallels.

KEYWORDS: ***fuzzy control, neurocontrol, planning, adaptive critic, reinforcement learning, neural networks, intelligent control, adaptive control, stochastic systems, basal ganglia, backpropagation***

1. INTRODUCTION

This paper mainly discusses neurocontrol—the use of neural networks (artificial or neural) to *directly* control motors, actuators, muscles, or other kinds of overt physical action. It also discusses the relation between artificial neural networks (ANNs) and fuzzy logic and how best to combine them. It

[*]The views expressed here are personal views and do not reflect official views of NSF; however, as government work, this paper is in the public domain.

Address correspondence to Paul J. Werbos, National Science Foundation, Room 1151, 1800 G. St., NW, Washington, D.C. 20550.

begins by discussing the most basic and most popular application of ANNs—learning a mapping from a vector **X** to a vector **Y**. Then it discusses neurocontrol and the central importance of neurocontrol to understanding intelligence. The final section includes some thoughts about reasoning and planning, directed more toward future research.

In this paper I will take the position that fuzzy logic and neurocontrol are complementary technologies. In many applications, the best approach is to use the two together, rather than decide which technology is "best." This complementarity is based in part on their common emphasis on the use of continuous variables, which also allows a high degree of complementarity with nonlinear control theory and a new generation of analog computer hardware.

Precisely because they are complementary technologies, there are certain semantic problems that arise in defining which technology is which (i.e., in defining the *boundaries* between neural nets and fuzzy logic). There are cases where neural networkers and fuzzy logicians would use exactly the same mathematics to solve a specific problem but would give the mathematics different names and would refer back to different sources. In cases like this, it is particularly absurd to try to decide which technology is "better," even for a specific problem; it is more realistic to lay out a diverse inventory of techniques in concrete terms while trying to exploit *both* traditions. In this paper I survey the techniques that have been developed in neurocontrol, in the hope that this will be useful to both communities.

In Section 3 I will make a crucial point that underlines the relevance of Sections 5–8 to fuzzy control: that the learning methods described in these sections can all be applied to fuzzy inference structures (or directly to fuzzy rules, in some cases). Even though those later sections talk about block diagrams filled in with neural networks, one can use the same block diagrams by plugging in fuzzy structures instead, using the various options discussed in Section 2. Some readers might prefer that I actually work out this substitution explicitly, across the entire range of options, and label them explicitly as "fuzzy learning control designs"; however, by describing neurocontrol on its own terms, I hope to make this paper accessible to a wider audience, starting with Section 3.

I cite a few examples and surveys of learning methods developed within the fuzzy logic community; however, there is no claim that these surveys are complete, and I make no effort to provide a complete crosswalk between those learning methods and neurocontrol. Such a crosswalk would be useful, but it could probably be presented better by someone more familiar with all the many strands of thought within fuzzy control in addition to the material described here.

I do try to be relatively comprehensive in laying out the inventory of designs used in the neurocontrol field. I do not describe any one application in extensive detail; however, the references point to papers that do this for a wide variety of designs and applications. In Section 9 I discuss one example of a hybrid fuzzy/neural system now being worked on by a group in Washington, D.C.

2. ANNs AND FUZZY LOGIC IN SUPERVISED LEARNING

Neurocontrol is still a small part of the greater neural network community. Most people use ANNs for applications such as pattern recognition, diagnostics, risk analysis, and so on. They mostly use ANNs to learn static mappings from an "input vector" **X** to a "target vector" **Y**. For example, **X** might represent the pixels that make up an image, while **Y** might represent a classification of that vector. Given a *training set* made up of pairs of **X** and **Y**, the network can "learn" the mapping, by adjusting its weights so as to perform well on the training set. In the example just given, it would learn to input the image and output the classification.

This kind of learning is called "supervised learning." There are many forms of supervised learning used by different researchers, but the most popular is basic propagation (Werbos [1]). Basic backpropagation is simply a unique implementation of least squares estimation. In basic backpropagation, one uses a special, efficient technique to calculate the derivatives of square error with respect to all the weights or parameters in an ANN; then one adjusts the weights in proportion to these derivatives, iteratively, until the derivatives go to zero. The components of **X** and **Y** may be 1's and 0's, or they may be continuous variables in some finite range.

Fuzzy logic is also used, at times, to infer well-defined mappings. For example, if **X** is a set of data characterizing the state of a factory, and **Y** represents the presence or absence of various breakdowns in the factory, then fuzzy rules and fuzzy inference may be used to decide on the likelihood that one of the breakdowns may be present, as a function of **X**.

Which method is better to use, when?

The simplest answer to this question is that since ANNs extract knowledge from empirical databases used as "training sets," and fuzzy logic usually extracts rules from human experts, we should simply decide which source of knowledge we trust more in the *particular* application. (When in doubt, we can try both and try for an evaluation after the fact.) In principle, empirical data represents the real bottom line and expert judgment is only a secondary source; however, when the empirical data are too limited to allow us to learn complex relations, expert judgment may be all we have.

In many applications, there are some *parts* of the problem for which we have adequate data and others for which we do not. In that case, the practical approach is to divide the problem up and use ANNs for one part and fuzzy logic for another part. For example, there may be an intermediate proposition

R that has an important influence on $\mathbf{Y}$; we may build a neural net to map from $\mathbf{X}$ to R, and a fuzzy logic system to map $\mathbf{X}$ and R into $\mathbf{Y}$, or vice versa. Amano et al. [2], for example, have built a speech recognition system in which ANNs detect the features and a fuzzy logic system goes on to perform the classification. Many people building diagnostic systems have taken similar approaches (Shreinemakers and Touretzky [3]).

In the current literature, many people are using fuzzy logic as a kind of organizing framework to help them subdivide a mapping from $\mathbf{X}$ to $\mathbf{Y}$ into simpler partial mappings. Each one of the simple mappings is associated with a fuzzy "rule" or "membership function." ANNs or neural network learning rules are used to actually learn all of these mappings. There are a large number of papers on this approach, reviewed by Takagi [4]. Kosko's work in this area is particularly famous. Because these are typically very simple mappings—with only one or two layers of neurons—we can choose from a wide variety of neural network methods to learn the mappings; however, since the ANNs only minimize error in learning the individual rules, there is no guarantee that they will minimize error in making the overall inference from $\mathbf{X}$ to $\mathbf{Y}$. This approach also requires the availability of *data* in the training set for all of the intermediate variables (little R) used in the partial mappings. Strictly speaking, this approach is a special case of the previous paragraph; in the general case, some rules can be learned while others come from experts.

Many people in fuzzy logic might say that fuzzy logic is more than just rules and inference. There is also such a thing as fuzzy learning. In fact, much of the neural network literature on learning [like backpropagation (Werbos [1]] applies directly to *any* well-behaved nonlinear network. It can be applied directly to the inference structures used in fuzzy logic. We could easily get into a situation where fuzzy logic people and neural network people use exactly the same mathematical recipe for how to adapt a particular network and use different names for the same thing. I would prefer to focus on the *generalized* mathematical learning rules so that we can speak a more universal language and avoid distinctions without a difference.

Some problems cannot be easily subdivided into expert-based parts and learning-based parts. For example, there are theories of international conflict that involve a rich structure, containing a large number of parameters known with varying degrees of confidence; it is important to expose the *entire structure* to the discipline of historical testing ("backcasting" and "calibration"). In situations like that, the best procedure is to *combine* fuzzy logic and learning. (In Bayesian terms, one would regard this as a *convolution* of prior and posterior knowledge, to determine the correct conditional probabilities, conditional upon all available information.) For example, we can use fuzzy logic and interviews with experts to derive an initial structure, and estimates of uncertainty. Then, we can use generalized backpropagation directly to adjust the weights (or uncertainty levels or other parameters) in that network. We can even use backpropagation to minimize an error measure like

$$E = \sum_i (Y_i - \hat{Y}_i)^2 + \sum_j C_j(W_j - W_j^{(0)})^2 \tag{1}$$

where C_j is the prior degree of certainty about parameter W_j, and $W_j^{(0)}$ is the prior estimate of the parameter. This kind of convolution approach could also be applied, of course, to the learning of independent rules or membership functions, as described by Takagi in [4]. In a recent meeting to discuss long-term strategic planning issues, I suggested a two-stage approach: (1) Build up an initial inference system or model using conventional techniques, which adapt individual rules or equations; (2) then, after assessing degrees of certainty, adjust all of the weights in a "calibration" phase, using backpropagation to make sure that the overall structure adequately fits the overall structure in historical data.

As far as I know, the idea of applying backpropagation to a fuzzy logic network was first published in 1988 (Werbos [5]). Matsuba of Hitachi, in unpublished work, first proposed the use of Equation (1). Backpropagation is important in this application because it can adapt multilayer structures.

Backpropagation *cannot* be used to adapt the weights in a more conventional, Boolean logic network. However, since fuzzy logic rules are differentiable, fuzzy logic and backpropagation are more compatible. Strictly speaking, it is not necessary that a function be everywhere differentiable to use backpropagation; it is enough that it be continuous and be differentiable almost everywhere. Still, one might expect better results from using backpropagation with modified fuzzy logics, which avoid rigid sharp corners like those of the minimization operator.

One reason for liking fuzzy logic, after all, is that it can do a better job than Boolean logic in representing what *actually exists in the mind of a human expert*. This being so, modified fuzzy logics, which are even smoother, may be even better. Fu [6] has gotten good results by applying backpropagation to simple fuzzy logic structures (using special rules to handle the corner points), while Hsu et al. [7] have proposed a modified logic. Presumably the fuzzy logic literature itself includes many examples of smooth, modified fuzzy logics. Among the obvious possibilities are (1) to use simple ANNs themselves in knowledge representation and (2) to use functional forms similar to those used by economists, in production functions and cost functions, with parameters to reflect the importance, the complementarity, and the substitutability of different inputs.

Fuzzy logic has the advantage that it can be applied in a flexible way, using a different inference structure for each case in the training set. This inference structure may contain logic loops that go beyond the capability of what ANN people call "feed-forward" networks. The inference structure may be a "simultaneously recurrent" network. Nevertheless, backpropagation can be used on such inference structures (using the memory-saving methods of Werbos [8]*) to calculate the derivatives of error with respect to every parameter, at a cost less than the cost of invoking the inference structure a single time. Thus one can use backpropagation here as well. Hybrid systems like this may be too expensive to justify for unique applications, but they make considerable sense in generalized software systems.

When complex inference is required, in fuzzy logic as in conventional logic, the design of an inference engine can be very tricky. Neurocontrol systems may be used, in essence, as inference engines. In fact, I would argue that this is precisely how the human brain does inference—that the true "deep structure" of language is a collection of neural nets that learn, through experience, how to perform more and more effective inference (in a non-Boolean environment). Inference may be more difficult than other forms of control problem; however, there are parallels between neurocontrol systems and existing inference engines that suggest some real possibilities here.

Stinchcombe and White proved (IJCNN 1989) that conventional ANNs can represent essentially any well-behaved nonlinear mapping. Sontag [9] extended this result to some of the ill-behaved mappings one sometimes encounters in control designs. Nevertheless, in applications of ANNs, many researchers have begun to encounter the limitations of *any* static mapping. In recognizing dynamic patterns (Werbos [1]) such as speech or moving targets, or in real-world diagnostics (Werbos [10]), it is often necessary to add memory of the past. As one adds such memory, it becomes more and more important to build up robust dynamic models of the system to be analyzed or controlled. Neural networks can do this (Werbos [11]), in part by adapting intermediate features and developing representations that an expert might not have thought of.

3. NEUROCONTROL IN GENERAL

In 1988, neurocontrol was just beginning a major period of growth. At that time, NSF sponsored a workshop on neurocontrol at the University of New Hampshire, chaired by W. Thomas Miller (Miller et al. [12]), who brought together a small mixed group of neural network people, control theorists, and experts in substantive application areas. In the very early part of that workshop, a few people echoed the old arguments about who is better, control

*When simultaneous recurrence is not present, the calculations are much simpler, as in Jordan [22].

theorists or neural networkers. Within a very short time, however, it became apparent that this issue was utterly meaningless. It was meaningless because it revolved about a distinction without a difference. The reason for this is illustrated in Figure 1.

Figure 1 is a Venn diagram, telling us that neurocontrol is a subset both of neural network research and of control theory. In the course of the workshop, it became apparent that the existing work in neurocontrol could be reduced to five fundamental design strategies, each of which occurred over and over again, with variations, in numerous papers. (Individual papers tend to highlight their unique aspects, of course.) All five turned out to be *generic* approaches that could be applied to any large, sparse network of differentiable functions or to an even larger class of networks. One may call these ''functional networks,'' as opposed to neural networks. All five methods could be fully understood as generic methods within control theory. By remembering that neurocontrol is a subset of both disciplines, we are in a position to draw upon both disciplines in developing more advanced design and applications.

This situation is particularly important to fuzzy logicians, because the inference structures of fuzzy logic are themselves functional networks. In this paper, I present numerous boxes labeled as ''neural networks,'' but every such box could just as easily be filled in with a fuzzy inference structure varying over time. In other words, every one of the five ''neurocontrol'' methods can also be applied directly to fuzzy learning as well. In practice, one would want to fill in different boxes with different things—perhaps an ANN for one box, a hybrid neural/fuzzy map (as described in the previous section) for another, and a conventional fixed algorithm for a third. This kind of mixing and matching is quite straightforward once one understands the basic principles.

Why should we be interested at all in the special case where the functional network is built up from the traditional kinds of artificial neurons? Why should

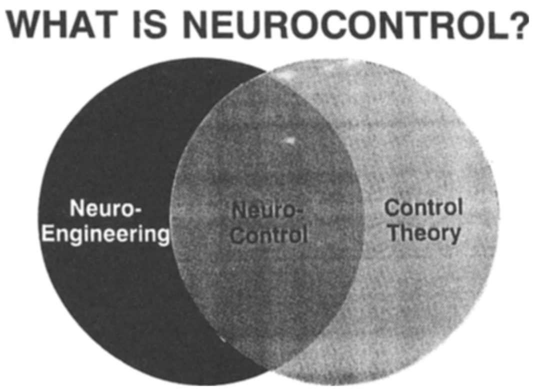

Figure 1. Neurocontrol is a subset.

we be interested in functional forms close to the conventional form used in ANNs (Werbos [1]),

$$x_i = s(\sum W_{ij} x_j) \tag{2}$$

where

$$s(z) = 1/(1 + e^{-z})? \tag{3}$$

(Here, x_i represents the "output" or "activation" of a model neuron, while W_{ij} represents a "weight" or "parameter" or "connection strength" or "synapse strength.")

There are at least four reasons for paying attention to the special case represented by neural networks: (1) the universal mapping theorems of Sontag [9] and White and others, (2) the availability of special-purpose computer hardware, (3) the pedagogical value of the special case, and (4) the link to the brain.

The theorems of White and others have excited great interest in the control community because they show that conventional ANNs do something very similar to what Taylor series do: provide a basis for approximating an arbitrary nonlinear function. As with Taylor series, the nonlinearity is very simple, offering a hope of workable practical tools.

The availability of special-purpose computer hardware is a decisive factor in favor of ANNs. There are many cases where a task can be done equally well using conventional sequential methods of neural nets and where both approaches involve a similar degree of computational complexity. (For examole, there are cases where an ANN can simply be trained to mimic the input-output behavior of an existing algorithm.) In such cases, ANNs may have a decisive advantage in real-world implementation, because of the hardware.

Intel, for example, recently produced a neural net chip, now publicly available, in cooperation with the U.S. Navy at China Lake. David Andes of China Lake has stated that one handful of these chips has more computational power than all of the Crays in the world put together. This is critical in applications where it is acceptable to add on a few extra chips but not haul along a Cray. Other companies, such as Syntonics in the United States and Oxford Computing in England, have also come up with impressive chips. Users without the technical knowledge (or clients) to wire up chips have reported that the neural board by Vision Harvest, Inc. (which includes a special-purpose chip) offers some of the same advantages. More and more products of this sort may be expected, especially if the optical approach reaches maturity.

Fuzzy logic chips have also been developed. However, because of the complexity of fuzzy logic, as normally practiced, these chips cannot take advantage of parallel distributed architecture as much as neural chips do. At the May 1990 conference in Houston on neural nets and fuzzy logic, the Japanese developer of one of the leading fuzzy chips stated unequivocally that one could expect far more computational throughput from a neural chip than from a fuzzy chip.

Harold Szu of the Naval Surface Warfare Center has often argued that digital parallel computers constitute the real "fifth generation" of computers, as far beyond current PCs as the PCs are beyond the old LSI mainframes. In a similar vein, he argues that fixed-function, analog distributed hardware—either VLSI or optical—represents a sixth generation. The NSF program in neuroengineering got its start when people like Carver Mead [13]—often viewed as the father of all VLSI—and people like Psaltis and Farhat and Caulfield (famous in optical computing) argued that this sixth generation could achieve a thousandfold or millionfold improvement in throughput over even the fifth generation. The challenge was to find a way to *use* this hardware in a truly general-purpose way. That is the goal that led to the neuroengineering program at NSF. Similar considerations have been crucial to the neural networks program at the Air Force Office of Scientific Research, which has also begun to stress neurocontrol. Some engineers would simply *define* an ANN as a general-purpose system capable (in principle) of efficient implementation in such hardware.

A third reason for being interested in neural networks as such is their pedagogical value. The importance of this should not be underestimated. For example, when I first published backpropagation as a *generalized* method for use with *any* functional network, it received relatively little attention, in part because the mathematics was unfamiliar and difficult. Later, when several authors (including myself) presented it as a method for use with simplified ANNs—with interesting interpretations, with nice flow charts using circles and lines, and with easy-to-use software packages (exploiting the simplicity that comes from giving the user no choice of functional form)—the method became much better known (Werbos [14]). Even now, for many people, it is easier to learn how to use a new design in the ANN special case and then generalize this knowledge than it is to start with the purest, most general mathematics. The explosion of interest in neural networks has also been very useful in motivating a new generation of graduate students, with diverse backgrounds, to learn the relevant mathematics. The effort to attract graduate students from diverse and nontraditional backgrounds—especially women and minorities—is now a major concern in Congress and the White House because of the changing composition of the young adult population in the United States. For example, from fiscal year 1990 to fiscal year 1991, Congress told NSF to increase its spending on education and human resources by about 50%, while single-investigator grants in engineering were held to something more like the rate of inflation.

A fourth reason for being interested in neurocontrol is the desire to be explicit about the link to the human brain. This link can be useful in both directions—from engineering to biology, and from biology to engineering.

The output of the human brain as a *whole system* is the control over muscles (and other actuators), as illustrated in Figure 2. Therefore the function of the brain as a whole system is control, over time, so as to influence the physical environment in a desired direction. Control is not *part* of what goes on in the brain, it is the function of the whole system. Even though lots of pattern recognition and reasoning and so on occur within the brain, they are best understood as subsystems or phenomena *within* a neurocontroller. To understand the subsystems and phenomena, it is most important to understand their function within the larger system. In short, a better understanding of neurocontrol will be crucial, in the long term, to a real understanding of what happens in the brain. (For a more concrete discussion of this, see Werbos [11].) Because the mathematics involved is general mathematics, it should be applicable to chips, to neurons, and to any other substrate we are capable of imagining to sustain intelligence.

The brain is living proof that it is *possible* to build an analog distributed controller that is capable of effective planning (long-term optimization) under conditions of noise, qualitative uncertainty, nonlinearity, and millions of variables to be controlled at once, all with a very low incidence of falling down or instability. Control at such a high level necessarily includes pattern recognition and systems identification as subsystems. Table 1 compares the five major design strategies now used in neurocontrol against the four most challenging capabilities of the brain that are of engineering importance.

Table 1 was developed two years ago (Miller et al. [12]), but it still applies to all the recent research that I am aware of (except that a few clever researchers like Narendra have developed interesting ways to combine some of these approaches). *Supervised control* is the strategy of building a neural network that imitates a preexisting control system; this is like expert systems, except that we copy what a person *says* instead of what he *does*, and can

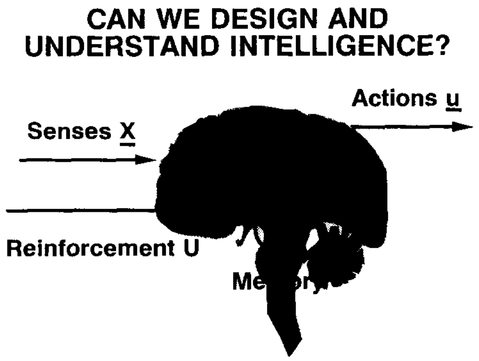

Figure 2. The brain as a whole system.

Table 1. Neurocontrol Versus Brain Capabilities

	Many Motors	Noise	Long-Term Optimization, Planning	Real-Time Learning
Supervised control	X	X		X
Direct inverse control	(X)	X		X
Neural adaptive control	X	?		?
Backpropagating utility	X		X	
Adaptive critics				
Two-net		X	X	X
BAC + DHP, etc.	X	X	X	X

operate at higher speed. *Direct inverse control* builds neural nets that can follow a trajectory specified by a user or a higher-level system. *Neural adaptive control* does what conventional adaptive control does, but it uses neural networks for the sake of nonlinearity and robustness; for example, an ANN may learn how to track an external reference model (as in conventional MRAC design). *Backpropagating utility* and *adaptive critics* are two techniques for optimal control over time—to maximize utility or performance, or to minimize cost, over time. All five will be discussed in more detail in later sections. (A similar taxonomy has been published by Sugeno [15] for fuzzy control approaches.)

Table 1 does suggest that we are now on a well-defined path to duplicating the most important capabilities of the human brain. However, the human brain is more than just a set of cells and learning rules. It is also a very *large* mass of cells. For the next few years, it may be better to think of ANNs as artificial mice (at best) rather than artificial humans. Mice are magnificent at some very difficult control and even planning tasks, but they are not very good at calculus (or is it that they don't pay attention?) Artificial humans are certainly possible, in my view, but there are many reasons to move ahead one step at a time. Personally, I find myself most interested in the last group of methods, because of its importance to understanding true intelligence; however, there are many engineering applications where it pays to use a simpler approach, and the brain itself may be a hybrid of many approaches.

4. AREAS OF APPLICATION

Four major areas have been discussed at length (Werbos [11], Miller et al. [12]) for possible applications of neurocontrol:

- Vehicles and structures
- Robots and manufacturing (especially of chemicals)

- Teleoperation and aid to the disabled
- Communications, computation, and general-purpose modeling (e.g., economics)

This paper cannot describe all these areas in depth, but a few words may be in order.

In vehicles and structures, the aerospace industry has been a leader in applying these concepts. Unfortunately, the most exciting applications remain proprietary. NSF has been mainly interested in sponsoring high-risk applications, which in turn serve as *risk reducers* in high-risk projects of economic importance. Risk reduction comes from providing an alternative, backup approach to solving very difficult problems that conventional techniques may or may not be adequate to solve. The National Aerospace Plane is a prime example. The goal is *not* to replace humans in space but to improve the economics required to make the human settlement of a space a realistic possibility. In October 1990, NSF and McDonnell-Douglas jointly sponsored a technical workshop on aerospace applications of neurocontrol, which will hopefully serve to advance this area. Barhen of the Jet Propulsion Laboratory has discussed a possible $15 million per year initiative on neural networks from NASA, with a control component. Ideally, there should be NSF/NASA cooperation here, so as to stimulate the development and testing of the most advanced forms of neurocontrol.

As this paper goes to press, McDonnell-Douglas (Sofge and White [16]) has revealed some of the details of one important application of neurocontrol, the manufacture of thermoplastic composite materials. Composite materials are both lighter and stronger than metals and would have enormous benefits throughout the economy if only they could be manufactured more cheaply. McDonnell-Douglas has used neurocontrol to solve problems in the continuous, lower cost manufacture of these materials, problems that had previously proved resistant to conventional methods (including expert systems). Test problems involving the integration of propulsion, steering, and thermal control will be published in the proceedings of the October 1990 workshop. It is hoped that successful solutions of those problems, incorporating noise and uncertainty, will produce greater confidence that a vehicle like NASP is actually feasible, even without allowing for future progress in areas like propulsion technology.

The chemical industry has also been quite active in neurocontrol. Major sessions have been held at the American Control Conference and at the annual meetings of the chemical societies on this topic. The Chemical Reaction Processes program at NSF held a workshop in January 1991, focusing on neurocontrol and laying the groundwork for expanded activity. The Bioengineering and Aid to the Disabled program has recently held a broad workshop to prepare for its approved initiative in this general area.

All of these new activities were motivated by interests expressed in the engineering community itself. There are many cases where industry or industry-oriented researchers are coping with fundamental issues that mainstream academics are barely beginning to address.

5. SUPERVISED CONTROL AND CONVENTIONAL FUZZY CONTROL

In the usual expert systems approach, a control strategy is developed by asking a human expert how to control something. Supervised control is essentially the ANN equivalent of that approach.

In supervised control, the first task is to build up a *training set*—a database—that consists of sensor inputs (**X**) and desired actions (**u**). Once this training set is available, there are many neural network designs and learning rules (like basic backpropagation) that can learn the mapping from **X** to **u**. Once the training set has been set up, the rest of this method is extremely simple.

Usually, the training set is built up by asking a human expert to perform the desired task and recording what the human sees (**X**) and what the human does (**u**). There are many variations of this, of course, depending on the task to be performed. (Sometimes the input to the human, **X**, comes from electronic sensors, which are easily monitored; at other times, it may be necessary to develop an instrumented version of the task, using teleoperation technology, as a prelude to building the database.) The goal is essentially to "clone" a human expert.

Supervised control has two other applications besides cloning a human expert. First, it can generate a controller that is faster than the expert. For example, a human might be asked to fly a slowed-down *simulated* version of a new aircraft. The ANN could then be implemented on a neural net chip that allows it to operate at a higher speed—higher than a human could keep up with. Second, it can be used to create a compact, fast version of an existing automated controller, developed from expert systems or control theory, that was too expensive or too slow to use in real-time, onboard applications. Supervised control is similar, in a way, to the old "pendant" system used to train robots; however, unlike the pendant system, it learns how to *respond* to different situations, based on different sensor input.

When should we use supervised control, with ANNs (or other networks), and when should we use fuzzy knowledge-based control?

Knowledge-based control is like following what a person *says*, while supervised control is like copying what the person *does*. Parents of small children may remember the famous plea: "Do what I say, not what I do." Knowledge-based systems obey this injunction. Supervised controllers do not.

There are many tasks where it is not good enough to ask people what they do and follow those rules. For example, if someone asked you how to ride a bicycle and coded those rules into a fuzzy controller, the controller would probably fall down a lot. Your system would be like a child who just *started* riding a bicycle based on rules he learned from his mother. The problem is that your knowledge of how to ride a bicycle is stored "in your wrists," in your cerebellum, and in other parts of your brain that you can't download directly into words. A supervised controller can imitate what you do and thereby achieve a more mature, complete, and stable level of performance. (This may be one reason why children have evolved to be so imitative, whether their parents like it or not.) Other forms of ANN control can go further and learn to do *better* than the human expert; however, it may be best to *initialize them* by copying the human expert, as a starting point, in applications where one can afford to do so.

The example here does *not* tell us that neurocontrol should be preferred over fuzzy logic in all cases. As with the problem of learning a mapping, discussed above, the theoretical optimum is to *combine* knowledge-based approaches and ANN approaches. As a practical matter, the theoretical optimum is often unnecessary and too expensive to implement. However, there are tasks that are too difficult to do in any other way.

As an example, consider the problem of learning how to do touch typing. Even a human being cannot learn to do touch typing simply by hunting and pecking and gradually increasing speed. In a technical sense, we would say that the problem of touch typing is fraught with "local minima," such that even the very best neural network—the human brain—can get stuck in a suboptimal pattern of behavior. To learn touch typing, one begins with a *teacher*, who explicitly conveys rules using words. *Then* one fine tunes the behavior, using neural learning. Then one learns *additional* rules. Only after one has *initialized* the system properly—by learning all the rules—can one rely solely on practice to improve the skill. Morita et al. [17] showed how a *two-stage* approach—knowledge-based control followed by backpropagation-based learning—can improve performance in certain supervised control problems. There are other ways to deal with local minima, but they *complement* the use of symbolic reasoning rather than compete with it.

In actuality, practical users of fuzzy control often tweak their rules and assumptions to get good control after the fact (e.g., see Kosko [18]). Morita's approach may be seen as a way of replacing that tweaking stage with something more objective, more automatic, and more suitable for larger and more confusing problems. At the October 1990 workshop mentioned above, Robicon Systems of Princeton, New Jersey, reported successful results using a *four-stage* strategy that carries Morita's approach still further, consistent with my ideas in [11].

Advanced practitioners of supervised control no longer think of supervised control as a simple matter of mapping $\mathbf{X}(t)$, at time t, onto $\mathbf{u}(t)$. Instead, they use past information as well to predict $\mathbf{u}(t)$. They think of supervised control as an exercise in "modeling the human operator." The best way to do this is by using neural nets designed for robust modeling, or "system identification," over time. There is a hierarchy of such ANN designs, the most robust of which has yet to be applied to supervised control (Werbos [11]).

Supervised control with an ANN was first performed by Widrow and Smith [19]. Kawato, in conversation, he stated that Fuji has widely demonstrated working robots based on supervised control. Many other applications have been published.

6. DIRECT INVERSE CONTROL

Direct inverse control is a highly specialized method used to make a plant (like a robot arm) follow a desired trajectory, a trajectory specified by a human being or by a higher-order planning system. The underlying idea is illustrated in Figure 3.

Let us suppose, for example, that we had a simple robot arm, controlled by two joints. One joint controls the angle θ_1, and the other determines θ_2. Our goal is to move the robot hand to a point in two-dimensional space with coordinates X_1 and X_2. We know that X_1 and X_2 are functions of θ_1 and θ_2. Our job here is to go ***backwards***: for ***given*** (***desired***) X_1 and X_2, we want to calculate the θ_1 and θ_2 that move the hand to that point. *If* the original mapping from $\boldsymbol{\theta}$ to $\mathbf{X}$ were invertible (i.e., if a unique solution always exists for θ_1 and θ_2), then we can try to learn this inverse mapping directly.

To do this, we simply wiggle the robot arm about for a while, to get examples of θ_1, θ_2, and the resulting X_1 and X_2. Then we adapt a neural network to input X_1 and X_2 and output θ_1 and θ_2. To *use* the system, we plug in the *desired* X_1 and X_2 as input.

Miller (Miller et al. [12]) has used direct inverse control to achieve great

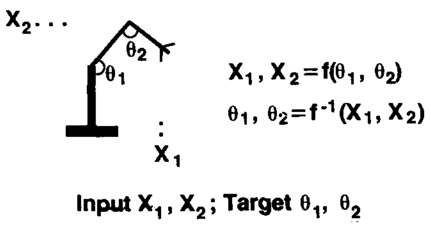

Figure 3. Direct inverse control.

accuracy (error less than 0.1%) in controlling an actual, physical Puma robot. Morita et al. [17] used direct inverse control with a fuzzy network but with an ANN learning rule, and claim that this is better than supervised control for the same problem. Grossberg and Bullock and Grossberg and Kuperstein have given many talks arguing that neural networks that implement direct inverse control are good models for biological phenomena like hand-arm coordination and visual tracking.

In direct inverse control, as in supervised control, it works better to think of the mapping problem in a dynamic context (Werbos [11]) to get better results. This may explain why Miller has gotten better accuracy than many other researchers using this method. (For example, some authors report positioning errors of 4% of the work space. Miller's method may be like getting 4% error in *reducing* the remaining gap between the desired position and the actual position; as that gap is reduced from one time step to the next, it should go to zero quite rapidly.) Because he uses a highly appropriate supervised learning rule (Narendra [20]), Miller reports that he can get a robot to adapt in real time to changing parameters. For example, in pushing an unstable cart around a figure 8 track, his robot arm demonstrates highly accurate tracking after three loops around the track, after the weight of the cart is changed. If Miller went further, by using a full-fledged system identification network with memory (Werbos [11]), using fast learning in the upper layer of the network, I would expect that his robot could adapt to changing weights even more rapidly than it now does.

Direct inverse control does not work when the original map from $\boldsymbol{\theta}$ to $\mathbf{X}$ is not invertible. For example, if the degrees of freedom of the control variables (like $\boldsymbol{\theta}$) are more or less than the degrees of freedom of the observables (like $\mathbf{X}$), there is a problem. Eckmiller et al. [21] found a way to break the tie, in cases where there are excess control variables; however, methods of this sort do not fully exploit the value of additional motors in achieving other desirable goals such as smooth motion and low energy consumption.

Kawato's "cascade method" (in Werbos [11]) and Jordan [22] describe more general ways of following trajectories, which *do* achieve these other goals, by rephrasing the problem as one of *optimal* control. They define a cost function as the error in trajectory following *plus* a term for jerkiness or torque change. Then they adapt a neural network to minimize this cost function. To do this, they use the backpropagation of utility—a different ANN design, to be discussed later on. Kawato also argues that optimizing networks of this kind fit more recent experiments better than direct inverse control can.

Earlier, Kawato developed a special-purpose inverse control design called "feedback error learning" (also in [11]), which requires starting off from a known feedback controller of adequate quality. SAIC (San Diego, California) has widely distributed a videotape demonstration of a working vibration suppressor (applied to glasses on a table), which may be seen as a special case of feedback error learning when the known feedback controller happens to be an identity map.

7. NEURAL ADAPTIVE CONTROL

Neural adaptive control ties to do what conventional adaptive control does, using ANNs instead of the usual linear mappings. Because there are many tools used in conventional adaptive control, this is a complex subject (Werbos [11], Narendra [20], Narendra and Annaswamy [23], Narendra and Parthasarathy [24]).

One common tool in adaptive control is model reference adaptive control, where a controller tries to make a system follow specifications laid down in a reference model. In the conference on neural networks and fuzzy logic in Houston this year, Narendra described a straightforward way to do this with ANNs. One can simply define a cost function to equal the *gap* between the output of the reference model and the actual trajectory, and then minimize this cost function exactly as Jordan and Kawato did, by backpropagating utility. In actuality, one does not *have* to use the backpropagation of utility to minimize this cost function; one could also use adaptive critic methods here (Werbos [11]).

In adaptive control, the goal is often to cope with slowly varying hidden parameters. There are *two* different ways of doing this with ANNs, which are complementary. One is by *real-time* learning, where an ANN, like a biological neural network, adapts its weights in real time in response to experience. Another is by adapting *memory* units that are capable of estimating the hidden parameters. Even without real-time learning, it is possible to train an ANN *offline* so that it will be adaptive *in real time* because of this memory (Werbos [11]). Ideally, one would want to combine both kinds of adaptation, but there is a price to be paid in so doing. The main price is that backpropagation through time must be replaced by adaptive critics (Werbos [11]) both in control and in system identification; the trade-offs involved will be discussed in the next section.

In conventional, linear adaptive control it is often possible to prove stability algebraically in advance by specifying a Lyapunov function (Narendra and Annaswamy [23]). In nonlinear adaptive control, it is far more difficult (Narendra [20]). In actuality, however, the "critic" networks to be discussed below function very much like Lyapunov functions (especially in the BAC design). For many complex, nonlinear problems, it may be necessary to *adapt* a Lyapunov function after the fact and verify its properties after the fact rather than specify it in advance.

8. BACKPROPAGATING UTILITY AND ADAPTIVE CRITICS

General Concepts

Backpropagating utility and adaptive critics are two general-purpose designs for *optimal* control using neural networks. In both cases, the user specifies a utility function or performance index to be maximized or a cost function to be minimized. In both cases, these designs will always have *more than one* ANN component. Different components are adapted by different learning rules, aimed at minimizing or maximizing different things.

There will always be an Action network, which inputs current state information (and perhaps other information) and outputs the actual vector of controls, $\mathbf{u}(t)$. The utility function itself can also be thought of as a network (the Utility network), even though it is not adapted. [Some earlier papers talked about "reinforcement learning," which is logically a special case of utility maximization (Werbos [11], Miller et al. [12])]. In most cases, there will also be a model network, which inputs a current description of reality, $\mathbf{R}(t)$, and the action vector $\mathbf{u}(t)$; it outputs a forecast of $\mathbf{R}(t + 1)$ and of $\mathbf{X}(t + 1)$, the vector of sensor inputs at time $t + 1$. (In some cases, the Model network can be a stochastic network, which outputs simulated values rather than forecasts.) Finally, in the case of critic designs, there will be a Critic network, which inputs $\mathbf{R}(t)$ and possibly $\mathbf{u}(t)$ and outputs something like an estimate of the sum of future utility across all future times.

The real challenge in maximizing utility over time lies in the problem of linking *present* action to *future* payoffs, across all future time periods. There are really only two ways to address this problem, in the general case. One is to take a proposed Action network and *explicitly* work out its future consequences for *every* future time period. This is exactly what the calculus of variations does in conventional control theory, and it is also what the backpropagation of utility does. The backpropagation of utility is equivalent to the calculus of variations, but, because derivatives are calculated efficiently through large sparse nonlinear structures, one may hope for less expensive implementation. A second approach is to adapt a network that *predicts* the optimal future payoff (over all future times) starting from a given value for $\mathbf{R}(t + 1)$, and to use that network as the basis for choosing $\mathbf{u}(t)$. This requires that we *approximate* the payoff function, J^0, of dynamic programming. This is the *adaptive critic* approach.

Backpropagating Utility

The backpropagation of utility through time is illustrated in Figure 4. In the backpropagation of utility, we must start with a Model network that has

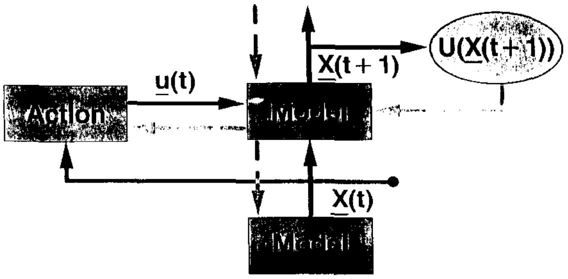

Figure 4. Backpropagation of utility through time.

already been adapted and a Utility network that has already been specified. Our goal is to *adapt* the weights in the Action network. (In practice, of course, we can adapt both the Action net and Model net concurrently; however, when we adapt the Action net, we treat the Model net *as if* it were fixed.) To do this, we start from the initial conditions, **X**(0), and use the initial weights in the Action network to predict **X**(t) at all future times t. Then we use generalized backpropagation to calculate the derivatives of *total* utility, across all future time, with respect to all of the weights in the Action network. This involves backwards calculations, following the dashed lines in Figure 4. Then we adjust the weights in the Action network in response to these derivatives and start all over again. We iterate until we are satisfied. The mechanics are described in more detail in [1], but Figure 4 really tells the whole story.

The backpropagation of utility was first proposed in 1974 (Werbos [25]). By 1988 there were four working examples. There was the truck-backer-upper of Nguyen and Widrow and the "cascade" robot arm controller of Kawato, both published in [11]. There was Jordan's robot arm controller (Jordan [22]) and my own official DOE model of the natural gas industry (Werbos [26]). Recently, Narendra and Hwang reported success with this method. Widrow recently showed videotapes of trucks with double trailers doing complex loops to avoid obstacles while backing up, all based on the same methodology. McAvoy has developed variations that may also be seen as generalizations of control methods that have been widely used in the chemical industry (Donat et al. [27]).

The backpropagation of utility is a very straightforward and exact method. Unfortunately, there have been few reported successes this past year. This may be due in part to a lack of straightforward tutorials (though [1] and [26] should help). The biggest problem in practical applications may be the difficulty of adapting a good Model network. In some applications, it may be good enough to build a Model network that inputs **X**(t) and **u**(t), that uses **X**(t + 1) as its target and contains time-lagged memory units (as described in [1]) to complete the state vector description; however, in some applications, it is crucial to go beyond this and insert special "sticky" neurons—designed to represent slowly varying hidden parameters—and elements of robust estimation (Werbos [11]).

Kosko [18], in comparing a fuzzy truck-backer-upper against a truck-backer-upper based on backpropagating utility, has reported results with the latter quite inferior to what Widrow claimed. Clearly there must be some difference in the two implementations here, which puts into doubt any conclusions about fuzzy logic versus neurocontrol and the like. The most obvious explanation would be differences in how the Model network is adapted—something quite crucial to success with this method—but there are other possibilities as well, which need to be investigated (differences verified as this goes to press).

The biggest limitation of backpropagating utility is the need for a *forecasting* model, which cannot be a true *stochastic* model. In fuzzy logic, this is not so bad, because the variable being forecast may itself be a measure of likelihood or probability. In some applications, however, like stock market portfolio optimization, a more explicit treatment of probabilities and scenarios may be important. There are tricks that can be used to represent noise, even when backpropagating utility, but they are somewhat ad hoc and inefficient (Werbos [11]).

Another problem in backpropagating utility is the need to learn in an *offline* mode. The calculations *backwards* through time require this. Various authors have devised ways to do backpropagation through time in a time-*forwards* direction (e.g., Werbos [28]), but those techniques are either very approximate or do not scale well with large problems or both; in any case, Narendra and Parthasarathy [24] questioned the stability of such methods. Nevertheless, even if we backpropagate utility in an offline mode, we can still develop a network that adapts in real time to changes in slowly varying parameters; we can "learn offline to be adaptive online" (Werbos [11]). This should be very attractive in many applications, because true real-time learning is more difficult.

Adaptive Critics

Adaptive critic methods, by contrast, do permit true real-time learning and stochastic models, but only at a price; they lack the exactness and simplicity of backpropagating utility. One reason for their lack of simplicity is the wide variety of designs available—from simple two-net structures, which work well on small problems, through to complex hybrids, which hopefully encompass what goes on in the human brain (Werbos [11], Miller et al. [12]).

Adaptive critic methods may be defined, in broad terms, as methods that attempt to *approximate* dynamic programming as I first described in [29]. Dynamic programming is the *only* exact and efficient method available to control actions or movements over time so as to maximize a utility function in a noisy, nonlinear environment, without making highly specialized assumptions about the nature of that environment. Figure 5 illustrates the trick used by dynamic programming to solve this very difficult problem.

Dynamic programming requires as its input a utility function U and a model of the external environment, $\mathbf{F}$. Dynamic programming produces, as its major output, another function, J, which I like to call a secondary or *strategic* utility function. The key insight in dynamic programming is that you can maximize the function U, in the *long term*, *over time*, simply by maximizing this function J in the immediate future. Once you know the function J and the model $\mathbf{F}$, it is then a simple problem in function maximization to pick the actions that maximize J. The notation here is taken from Raiffa [30], whose books on decision analysis can be viewed as a highly practical and intuitive introduction to the ideas underlying dynamic programming.

Unfortunately, we cannot use dynamic programming *exactly* on complicated problems, because the calculations become hopelessly complex. (Bayesian inference sometimes entails similar complexities.) However, it is possible to *approximate* these calculations by using a *model* or *network* to estimate the J function or its derivatives (or something quite close to the J function, like the J' function of Werbos [31] and Lukes et al. [32].) Adaptive critic methods can be defined more precisely as methods that take this approach.

If this kind of design were truly fundamental to human intelligence, as I would claim, one might expect to find it reflected in a wide variety of fields. In fact, notions like U and J do reappear in a wide variety of fields, as illustrated in Table 2 (taken from Werbos [33]). Note that the last entry in Table 2, the entry for Lagrange multipliers, corresponds to the *derivative* of J rather than the value of J itself. In economic theory, the prices of goods are supposed to reflect the *change* in overall utility that would result from *changing* your level of consumption of a particular good. Likewise, in Freudian psychology, the notion of emotional charge associated with a *particular object* corresponds more to the *derivatives* of J; in fact, the original inspiration for backpropagation (Werbos [34]) came from Freud's theory that emotional

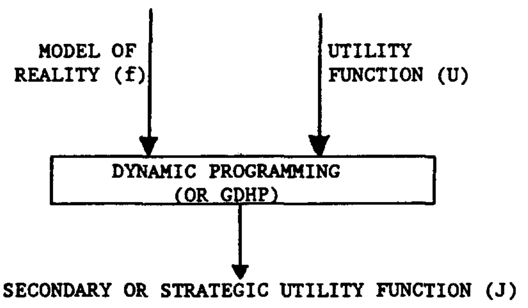
MODEL OF REALITY (f)
UTILITY FUNCTION (U)
DYNAMIC PROGRAMMING (OR GDHP)
SECONDARY OR STRATEGIC UTILITY FUNCTION (J)

Figure 5. Inputs and outputs of dynamic programming.

Table 2. Examples of J and U

Domain	Basic Utility (U)	Strategic Utility (J)
Chess	Win/lose	Queen = 9 points, etc.
Business theory	Current profit, cash flow	Present value of all assets (performance measures)
Human thought	Pleasure/pain	Hope/fear
	Hunger	Reaction to job loss
Behavioral psychology	Primary reinforcement	Second reinforcement
Artificial intelligence	Utility function	Statistic position evaluator (Simon)
		Evaluation function (Hayes-Roth)
Government finance	National values, long-term goals	Cost/benefit measures
Physics	Lagrangian	Action function
Economics	Current value of product to you	Market price or shadow price (''Lagrange multipliers'')

charge is passed *backwards* from object to object, with a strength proportionate to the usual *forwards* association between the two objects (Yankelovitch and Barrett [35]). The backpropagated adaptive critic (BAC) design reflects that theory very closely. The word "pleasure" in Table 2 should not be interpreted in a narrow way; for example, it could include such things as parental pleasure in experiencing happy children.

In order to build an adaptive critic controller, we need to specify two things: (1) how to adapt the Action network in response to the critic; (2) how to adapt the Critic network.

The most popular adaptive critic design by far is the two-network arrangement of Barto et al. [36] illustrated in Figure 6. In this design, there is no need for a model of the process to be controlled. The estimate of J is treated as a gross reward or punishment signal. This design has worked well on a wide variety of real-world problems, including robotics (Franklin [37]), autonomous vehicles, and fuzzy logic systems. Williams [20] reported some interesting new results on convergence. Unfortunately, this approach becomes very slow as the number of control variables or state variables grows to 10 or 100. The reason for this is very straightforward: Knowing J is not enough to tell us *which* actions were responsible for success or failure, and it does not tell us whether we need *more* or *less* of any component of the action vector. This design is like telling a student that he or she did "well" or "poorly" on an exam without pinpointing which answers were right or wrong; it is a lot harder for a student to improve performance when he or she has no specific idea of what to work on.

Fortunately, there are alternative designs that can overcome this problem. Note that it is critical to modify *both* the Action network *and* the Critic network, to permit learning at an acceptable speed when the number of variables is large (as in the human brain). There are also some other tricks that can help, discussed by myself, by Barto, and by Sutton [11, 12, 20].

To speed up learning in the Action network, for *large* problems, there are now two major alternatives: (1) the backpropagated adaptive critic (BAC), shown in Figure 7, and (2) the action-dependent adaptive critic (ADAC), shown in Figure 8. The BAC design is closer to dynamic programming than is

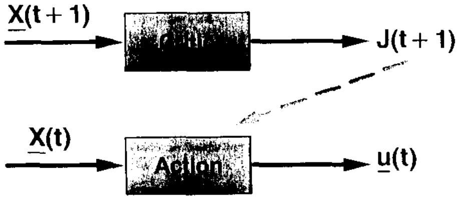

Figure 6. Two-Net design of Barto, Sutton, and Anderson.

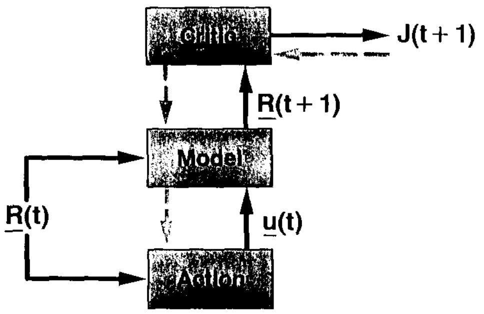

Figure 7. Backpropagated adaptive critic (BAC).

the two-net design because there is a more explicit attempt to pick **u**(t) so as to maximize $J(t + 1)$, based on the use of generalized backpropagation to calculate the derivatives of $J(t + 1)$ with respect to the components of **u**(t). The dashed lines in Figure 7 represent the calculation of derivatives. [Usually we adapt the *weights* in the action network in proportion to these derivatives rather than adapting **u**(t) itself.] The cost of BAC is that we need to develop a Model network, as we do when backpropagating utility. The adaptation of a good dynamic model can be a challenging task at times (Werbos [11]).

ADAC (Werbos [31], Lukes et al. [32]) avoids the need for an explicit model, but the Critic network in Figure 8 would have to represent the *combination* of the Critic and Model networks in Figure 7. Jordan, in conversation, has stated that he adapted an action-dependent Critic network in 1989, based on an independent paper by Watkins on "Q learning" (discussed in Narendra and Parasarathy [24]), but found the resulting Critic network to be rather complex. A variety of ADAC, with a few additional features proposed in Miller et al. [12], was the basis for the McDonnell-Douglas success with composite materials (Sugeno [15]) discussed in Section 3.

In an ideal world, one would want to combine the BAC and ADAC approaches, so as to combine the modularity and cleanliness of BAC with the

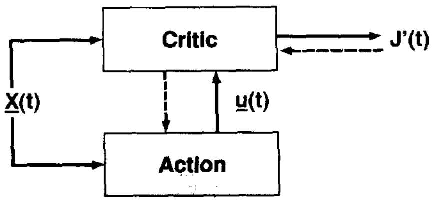

Figure 8. Action-dependent adaptive critic (ADAC).

model-independent robustness of ADAC; however, BAC may be good enough by itself in many applications. Jameson [38] has reported some preliminary results with BAC, and other aerospace-oriented researchers may have dealt with larger applications; however, more work is needed. Whatever the details, the adaptation of the Action network in large-scale problems is clearly central to the future of this discipline and of our ability to understand organic intelligence.

In adapting the Critic networks, few people have gone beyond simple, scalar methods that are more or less equivalent (Werbos [39]) and have severe scaling problems. There are two alternatives that should scale much better: (1) dual heuristic programming (DHP), which outputs estimates of the *derivatives* of J; and (2) globalized DHP (GDHP), which outputs an estimate of J (or its components) but adapts the critic by minimizing error in the implied derivatives as well as the estimate of J. These methods were first proposed in the 1970s (Werbos [28, 29]) but are described in more modern language in Werbos [11] and Miller et al. [12]. Both methods *require* the existence of a Model network. Hutchinson of BehavHeuristics has claimed real-world commercial success in applying such methods, but many of the details are proprietary.

Most neural network researchers have adapted Model networks to *predict* rather than *simulate* the plant to be controlled. One can build a stochastic simulation model from a prediction model, simply by measuring the errors in prediction and generating random numbers to simulate errors of the measured magnitude; however, this assumes that these errors at the point of prediction are uncorrelated with each other. It now seems possible to develop neural networks capable of simulating an unknown plant, in a way that fully accounts for correlations between errors across time and space (Werbos [40]). To prove that this can work is an area for future research. It is not obvious, a priori, that human brains have this kind of capability at the neuronal level; in other words, it is conceivable that human brains use fuzziness rather than true probabilities to handle uncertainty.

9. EXAMPLE OF A HYBRID SYSTEM

In 1988, a friend of mine asked how I would use these methods to assist in some very complex social decision problems, well beyond the scope of this paper. Given the nature of his application, I recommend a very conservative approach for the time being. As a first stage, I would obtain a conventional sort of modeling system, capable of storing and analyzing time-series data and capable of manipulating forecasting models built up from any of three methodologies: (1) econometric style equations, (2) fuzzy logic, and (3) ANNs. I would look for a *linkage* capability, so that models of specific sectors (built up from different methodologies and often revised) could be combined to yield composite streams of forecasts. Then I would build a general-purpose "dual compiler." The dual compiler would input a sectoral model (in text form or parsed into a tree) and output a "dual subroutine" (like those in [1]), so as to facilitate the use of generalized backpropagation. Then I would implement a whole set of tools using backpropagation.

Tool number 1 would be a simple sensitivity analysis tool. The user would type in a utility function or target function. The tool would then calculate the derivatives of utility with respect to *all* of the inputs—initial values, policy variables, and parameters—that affected the original forecast, in one quick sweep through the process. It would report back the 10 or 100 most important inputs. (There is a scaling problem here in deciding which input is most important; the user could be given a choice, for example, between looking for the biggest derivatives, the biggest elasticities, or the biggest derivatives weighted by some other variables.) The user could go on to make plans to *change* these inputs so as to increase utility, *or* he could first evaluate in detail whether he believes that the inputs are really important. (Tests of this sort can in fact be very useful in pinpointing *weaknesses* of an integrated modeling system (Werbos [8]) or real-world uncertainties, which require more analysis). The *cost* of a comprehensive sensitivity analysis is the key issue here; using more conventional tools, one must often wait a long time and spend a lot of money to get even a partial sensitivity analysis, and the results are usually out of date.

Tool number 2 would help in reassessing the importance of the key inputs. For any given input, it would use the *intermediate* information generated by backpropagation (as in Werbos [8]) to identify the path of connections that really made that input important. It could even display this information as a kind of tree or flow chart. This would be similar in purpose to the inference sequences printed out as "explanations" by many expert systems.

Tool number 3 would be an extended version of tool 1 and 2. Instead of first derivatives, it would provide information based on low-cost second derivatives [as I described in [28], based on calculations like those in Werbos [5] and Miller et al. [12]). For example, the sensitivity of utility to dollars spent in 1992 may be a key measure of policy effectiveness; it may be useful to see how *that* measure, in turn, would be changed by other factors (such as diminishing returns or complementary variables). At the optimum, the first derivative of utility with respect to any policy variable will be zero; the derivatives of *that* derivative give information about why the policy variable should be set at a particular level.

Tool number 4 would be a full-fledged version of backpropagating utility. The user could flag certain variables or parameters as policy variables, and the computer would be asked to suggest an optimal *improvement* upon current plans, so as to maximize utility. The resulting suggestion may be a local minimum, but it should at least be better than the starting plans.

Tool number 5 would be a model calibration tool, based on the backpropagation of error, and robust estimation concepts like those of Werbos [11, 40]. At a minimum, this would be a relatively quick and objective way to calibrate a model as a whole system to fit the past; it could replace the rather elaborate and ad hoc "tweaking" that usually goes into most complex models in the real world for calibration purposes.

Tool number 6 would go back and identify how the resulting parameter estimates or rules were influenced by different cases in the input data set; this would provide an integrated, nonlinear version of the highly respected linear diagnostic tools developed by Belsley et al. [41].

These six tools are the ones most obviously needed, exploiting backpropagation, but a host of other tools are possible involving estimation diagnostics, decision diagnostics, and convergence. Also, there is no need to develop the six tools in the order of my discussion.

In principle, one can even build a strategic assessment or stochastic planning tool, based on adaptive critic methods but permitting user-specified assessment models, as I described in [33].

To bring all these tools together in a general-purpose modeling package capable of running on desktop workstations would not be a trivial task. However, there are important applications, and some work has begun in this direction. All of these tools aim at effective *two-way* human–machine communication, so as to exploit the capabilities of both forms of intelligence.

10. REASONING, PLANNING, AND CHUNKING: THOUGHT FOR FUTURE RESEARCH

Experts in traditional artificial intelligence (AI) often ask two questions about neurocontrol or about neural networks in general:

1. How could such low-level architectures be extended to large-scale planning systems capable of a long-term planning horizon and capable of structuring very complex decision processes?
2. Where does symbolic reasoning fit into this picture?

This section will mainly focus on the first of these questions. There are other mammals besides humans that are clearly capable of complex forms of problem solving without using symbolic reasoning as such. At the neural level, it seems very clear that the brains of other mammals are really quite similar to those of humans. Early speculations (and hopes) that language is based on fundamentally unique kinds of neural structure have not held up well in recent research (Belsley et al. [42]). Formal symbolic reasoning as we practice it today is relatively recent, even within the history of the human species, and—at its best—it uses the entire structure of strategic planning of the brain to *learn* complex rules for manipulating those actions we call speech or writing. One should not expect to reduce the actual content of these rules and strategies to a simple, modular structure transmittable by the genes. In other words, the rules of symbolic reasoning as such may not be hardwired in the brain, but learned. To learn such complex rules, however, the brain must have a great inborn capacity for what is called "planning" in AI.

The problem of hierarchical or multilayer planning occurs not only in neural networks, but in AI and control theory as well. For example, automatic control of the main arm of the space shuttle presents a severe challenge to classical control theory. Seraji developed a hierarchical control scheme that worked on a Puma robot arm to some degree but was computationally intensive and never deployed. This past year (Parten et al. [43]), the joint controllers used by Seraji were replaced by neurocontrollers (using direct inverse control and computationally affordable), with a substantial improvement in performance, at least in simulations. Tests on the real arm on the ground have just started up.

At any one level of abstraction or aggregation, the higher order adaptive critic architectures described in Section 8 are very similar in spirit to the planners used in AI, as Table 2 would suggest, except for their learning ability. The emphasis is on refining the evaluation function rather than performing better tree searches; however, studies of human abilities in games like chess and GO suggest that a high-quality evaluation function is the real root of those abilities. One would therefore expect neurocontrol designs to be reasonable for planning problems at this level.

The greater difficulty lies in how to handle *multiple* levels of abstraction or aggregation—a problem that is often called "chunking" in the literature of AI. As in AI, some neural network researchers have developed elaborate hierarchies to break down complex decision problems, both to allow chunking and to prevent any one neurocontroller in the system from being overloaded with too many inputs. With adaptive critic controllers, for example, one can use $J(t + 1) - J(t) - U(t)$ as calculated by a higher level controller as the intrinsic utility function $U(t)$ to be input to a subordinate controller (Werbos [11]).

Unfortunately, these kinds of designs tend to require prior knowledge about how to structure the hierarchy. True hierarchies are not consistent with the more modular or "heterarchical" structure of the brain, and they limit the possibility of rich feedback between layers. Fortunately, a hierarchical array of neural networks performing similar functions can usually be represented as a *single* neural network with restrictions on which neuron inputs from which. Thus one can build a *single* network that initially reflects one's prior knowledge (if that prior knowledge points toward a hierarchy) and then allow the network to make and break new connections in all directions, based on adaptive rules for making and breaking connections. This allows us to recover modularity and flexibility without losing the crucial benefits of a sparsely connected network.

This sparsity approach can work very well *if* the various networks in the original hierarchy all work on a common cycle time. For example, in the human limbic system [which appears to function like a Critic network (Werbos [44])], there is a standard cycle time of about a quarter of a second, which corresponds to the classical theta rhyme of brainwaves. This example leads to a fundamental question that has yet to be answered: Can an adaptive critic system, based on a uniform cycle time shorter than a second, *effectively* plan over very long intervals of time? If we adapt such networks by use of new learning rules based on the best that numerical analysis has to offer (as proposed by Jameson in [38]), will the approximation to dynamic programming be fast enough to compete with brute force methods like AI planning? The brute force approach tends to fit poorly with the neural network approach, because it does not have a neural way of accommodating complex, noisy, nonlinear environments in the general case. However, the use of adaptive critics is still relatively new, and the possibilities for faster convergence on larger problems with sophisticated learning rules have hardly been tested at all.

One way to extend the foresight capability of adaptive critics—and thus help explain the foresight of biological brains—would be to use complex types of networks that somehow lend themselves better to foresight. However, simple feedforward networks can learn to represent almost any function (Sontag [9]). *All* of the known methods for adapting Critic networks are based on supervised learning (Werbos [11, 12, 40], Miller et al. [12]) with properly calculated inputs and targets. How could it help to use a more complex network design when solving a supervised learning problem, if simple feed-forward networks are certain to be good enough anyway? There are many ways that this can happen, especially if the structure of a complex network fits what we know about the problem to be solved; even though a feed-forward network can do the job, eventually, it may be possible to *learn* the job much faster with a more appropriate network design, a design that requires fewer weights or parameters to fit the problem at hand.

In the special case of planning problems, there is an interesting argument to suggest that *recurrent* networks might in fact work better (Werbos [44]).

In conventional planning, one tries to build networks or models that represent well-defined *tasks*. The descriptors of these tasks are like Model networks that input a description of the task initial state A (at time t) and immediately output the state B that would be achieved (at time $t + T$). In a stochastic world, this approach becomes increasingly difficult as the complexity and the noise increase; it becomes every more difficult to predict the final state. In fact, the whole notion of *deterministic* planning becomes less and less valid as the time horizon grows. One way to deal with this problem is to rely on *evaluation* networks rather than *models* to jump over multiple time intervals. Since evaluation functions (J) are based on dynamic programming, the paradigm remains valid even at very high noise levels. The challenge is to build an evaluation or Critic network that can translate a *value* or *goal* B for time $t + T$ into a value or goal A for time t in a single step of calculation. The obvious way to do this is to build a Critic network in which the value weight on variable B is represented by a neuron that is then used as input to the neuron that evaluates A. Once this connection is learned, then *changes* in the value placed on B (which are not really time-indexed) should lead *immediately* into changes in the value placed on A. This kind of approach would work only with the more sophisticated types of critic architecture, and it is critical to *avoid* treating the recurrent links as if they were memories of the external environment at earlier times.

An interesting aspect of this approach is that the Critic network could look more like the traditional Hopfield network, which is known to be highly effective in solving combinatorial optimization problems. Designs of this sort would probably be ideal in applications like SDI, where dynamic control is required, but the critic must somehow accomplish calculations that are qualitatively similar to combinatorial optimization. Recurrent networks can take time to settle down to an optimum or equilibrium output; this, in turn, is consistent with our subjective understanding of how humans sometimes take time to perform evaluations in games like go or Tetris. Feed-forward nets can learn to *emulate* recurrent nets, but only with greater complexity, which requires greater learning time; this is consistent with the way in which humans can learn to do well in games like Tetris at a slow speed, and then only slowly learn to perform the evaluations in a faster, more reflex mode.

In summary, the best solution to the long-term planning problem, in a noisy environment, may well be an adaptive critic structure with a recurrent Critic network. The best short-term motor control may use an adaptive critic design with a feed-forward critic. Nevertheless, one may still argue that the ultimate learning control system would still take advantage of more deterministic task-oriented planning in the intermediate term—at time intervals short enough that defined tasks have a high probability of being accomplished. One can imagine a three-level control systems, with a long-term controller directing an intermediate-term controller and both of them controlling a short-term optimization system. Each of the three would be as complex as the BAC system described in Section 8.

From an engineering point of view, we are a long way from needing that degree of complexity, and we may never really need it. However, these notions do have intriguing biological parallels. There does exist a kind of intermediate-term control structure in the brain [a part of the basal ganglia (Brooks [45], Marsden [46])] that does seem to generate a kind of task representation. The basal ganglia are clearly subordinate to the longer term planning system, made up of the limbic system and the cerebral cortex proper (neocortex), and superior to the short-term motor control system, made up of the brain stem and cerebellum. As evolution has progressed, the basal ganglia appear to have grown less important; however, the cerebral cortex itself has important links, both in anatomy and in evolution, with the basal ganglia.

How could one build such an intermediate-term task-oriented planning system out of analog distributed hardware like neurons? The biological literature may offer some valuable clues, or we may need to work with the biologists to help design new experiments that provide the kind of clues we need for our purposes. One partial possibility among many would be to design an adaptive critic system, in which the output of a mid-level Critic network would consist of two vectors, **R*** (to be interpreted as a short-term goal state) and **w**, with the estimate of J estimated implicitly as

$$J(\mathbf{R}) = \sum_i w_i(R_i^* - R_i)^2$$

Despite its unique structure, this Critic network could still be adapted by HDP or GDHP, or perhaps a learning rule could be found that works better for this specific case. If goal states tend to be attained, under the proper conditions, then the part of the Critic network that generates **R*** as a function of **R** could then be used as a kind of higher level Model network, jumping from one time to a much later time. This provides for only one level of temporal chunking, but the biological literature (Brooks [45], Marsden [46]) suggests that that is all we need. The success of such a scheme would depend critically on the ability of the system to learn new vector components R_i that tend to represent the degree of progress toward accomplishing particular tasks. In this connection, it is interesting that the basal ganglia appear to have less of a distributed, holographic architecture than many other parts of the brain; a single cell may well play a crucial role in triggering an entire task.

Considerable research will be needed to fully exploit and understanding all the higher-order options described in this section. One may hope that neural networks that can learn complex planning tasks will also be able to learn symbolic reasoning, without any hardwiring of the basic concepts, just as humans appear to do. Research on those lines is promising (Shastri [47], Werbos [48]) but has only just begun.

11. CONCLUSIONS

Neurocontrol and fuzzy logic are complementary, rather than competitive, technologies. There are numerous ways of combining the two technologies. Which combination is best depends very heavily on the particular application; there is always a trade-off between "general syntheses"—which combine everything but requires the expense of *implementing* everything—and direct, simple designs tuned to particular concrete problems. Given the natural human tendency toward inertia, it is critical to be aware of a wide variety of options and to ask "Why not?" when considering new approaches. Even within neurocontrol, there are a wide variety of designs available, ranging from simple off-the-shelf technologies (easily applied to fuzzy logic networks) to areas where fundamental research is still needed and vital to our understanding of real intelligence.

References

1. Werbos, P., Backpropagation through time: what it does and how to do it, *Proc. IEEE*, October 1990.
2. Amano, A., et al., On the use of neural networks and fuzzy logic in speech recognition, *Proceedings of the International Joint Conference on Neural Networks* (IJCNN), IEEE, New York, June 1989.
3. Schreinemakers, J., and Touretzky, D., Interfacing a neural network with a rule-based reasoner for diagnosing mastitis, *IJCNN Proceedings*, Erlbaum, Hillsdale, N.J., January 1990.
4. Takagi, H., Fusion technology of fuzzy theory and neural networks, *Proc. Fuzzy Logic and Neural Networks*, Izzuka, Japan, 1990.
5. Werbos, P., Backpropagation: past and future, *Proceedings of the 2nd International Conference on Neural Networks*, IEEE, New York, 1988. A transcript of the actual talk, with slides, is available from the author, and covers some additional topics.
6. Fu, L., Backpropagation in neural networks with fuzzy conjunction units, *IJCNN Proceedings*, IEEE, New York, June 1990.
7. Hsu, L. S., Teh, H. H., Chan, S. C. and Loe, K. F., Fuzzy logic in connectionist expert systems, *IJCNN Proceedings*, Erlbaum, Hillsdale, N.J., January 1990.
8. Werbos, P., Generalization of backpropagation, *Neural Networks*, October 1988.
9. Sontag, E., *Feedback Stabilization Using Two-Hidden-Layer Nets*, SYCON-90-11, Rutgers Univ. Center for Systems and Control, New Brunswick, N.J., October 1990.
10. Werbos, P., Making diagnostics work in the real world: a few tricks, in *Handbook of Neural Computing Applications*, (A. Maren, Ed.), Academic, New York, 1990, p. 337.
11. Werbos, P., Neurocontrol and related techniques, in *Handbook of Neural Computing Applications*, (A. Maren, Ed.), Academic, New York, 1990.
12. Miller, W. T., Sutton, R. S., and Werbos, P., Eds., *Neural Networks for Control*, MIT Press, Cambridge, Mass., 1990.

13. Mead, C., *Analog VLSI and Neural Systems*, Addison-Wesley, Reading, Mass., 1989.

14. Werbos, P., Links between artificial neural networks (ANN) and statistical pattern recognition, in *Artificial Neural Networks and Statistical Pattern Recognition: Old and New Connections*, (I. Sethi and A. Jain, Eds.), North-Holland, New York, 1991.

15. Sugeno, *Industrial Applications of Fuzzy Control*, North-Holland, Amsterdam, 1985.

16. Sofge, D., and White, D., Neural network based process optimization and control, in *Proceedings of the 29th IEEE Conference on Decision and Control*, IEEE, New York, January 1991.

17. Morita et al., Fuzzy knowledge model of neural network type, *IJCNN Proceedings*, Erlbaum, Hillsdale, N.J., 1990.

18. Kosko, B., Comparison of fuzzy and neural truck backer-upper control systems, *IJCNN Proceedings*, IEEE, New York, 1990.

19. Widrow, B., and Smith, Pattern-recognizing control systems, *1963 Computer and Information Sciences (COINS) Symposium Proceedings*, Spartan, Washington, D.C.

20. Narendra, K., Ed., *Proceedings of the Sixth Yale Workshop on Adaptive and Learning Control*, K. Narendra, Yale, New Haven, Conn., 1990.

21. Eckmiller, R., Beckmann, J., Werntges, H., and Lades, M., Neural kinematics net for a redundant robot arm, *IJCNN Proceedings*, IEEE, New York, June 1989.

22. Jordan, M., Generic constraints on underspecified target trajectories, *IJCNN Proceedings*, IEEE, New York, June 1989.

23. Narendra, K., and Annaswamy, A. *Stable Adaptive Systems*, Prentice-Hall, Englewood, N.J., 1989.

24. Narendra, K., and Parthasarathy, K., Identification and control of dynamical systems using neural networks, *IEEE Trans. Neural Networks*, **1**(1), 4-27 (1990).

25. Werbos, P., *Beyond regression: new tools for prediction and analysis in the behavioral sciences*, Ph.D. thesis, Applied Mathematics, Harvard Univ., November 1974.

26. Werbos, P., Maximizing long-term gas industry profits in two minutes in Lotus using neural network methods, *IEEE Trans. SMC*, March/April 1989.

27. Donat, J., Bhat, N., and McAvoy, T., Optimizing neural net based predictive control, *1990 American Control Conference*, IEEE, New York, 1990.

28. Werbos, P., Applications of advances in nonlinear sensitivity analysis, in *Systems Modeling and Optimization: Proceedings of the International Federation for Information Processing*, (R. Drenick and F. Kozin, Eds.), Springer-Verlag, New York, 1982, p. 762-770.

29. Werbos, P., Advanced forecasting methods for global crisis warning and models of intelligence, *General Systems Yearbook*, 1977.

30. Raiffa, H., *Decision Analysis: Introductory Lectures on Making Choices Under Uncertainty*, Addison-Wesley, Reading, Mass., 1968.

31. Werbos, P., Neural networks for control and system identification, *IEEE CDC Proceedings*, IEEE, New York, 1989.

32. Lukes, G., Thompson, B., and Werbos, P., Expectation driven learning with an associative memory, *IJCNN Proceedings* (Washington), Erlbaum, Hillsdale, N.J., 1990.

33. Werbos, P., Generalized information requirements of intelligent decision-making systems, *SUGI-11 Proceedings*, SAS Institute, Cary, N.C. A revised version, available from the author, is somewhat easier to read and elaborates more on connections to humanistic psychology.

34. Werbos, P., Elements of intelligence, *Cybernetica* (Namur), No. 3, 1968.

35. Yankelovitch, D., and Barrett, *Ego and Instinct: The Psychoanalytic View of Human Nature – Revised*, Vintage, New York, 1971. This is an unusually clear presentation, though my own work would suggest it is too pessimistic in its conclusions.

36. Barto, A., Sutton, R. S., and Anderson, C. W., Neuron-like adaptive elements that can solve difficult learning control problems, *IEEE Trans. Syst., Man, Cybern.* **13**, 834-846, 1983.

37. Franklin, J., Reinforcement of robot motor skills through reinforcement learning, *IEEE/CDC Proceedings*, IEEE, New York, 1988.

38. Jameson, J., A neurocontroller based on model feedback and the adaptive heuristic critic, *IJCNN Proceedings* (San Diego), IEEE, New York, June 1990.

39. Werbos, P., Consistency of HDP applied to a simple reinforcement learning problem, *Neural Networks*, **3**, 179-189, 1990.

40. Werbos, P., in D. White and D. Sofge, eds., *Handbook of Intelligent Control*, Van Nostrand, 1992.

41. Belsley, D., Kuh, and Welsch, *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*, Wiley, New York, 1980.

42. Segalowitz, S. J., Ed., *Language Functions and Brain Organization*, Academic, New York, 1983.

43. Parten, C., Pap, R., and Thoma, C., Neurocontrol applied to telerobotics for the space shuttle, *Proceedings of the International Neural Network Conference* (Paris, 1990), Vol. 1, Kluwer, Boston, 1990.

44. Werbos, P., Building and understanding adaptive systems: a statistical/numerical approach to factory automation and brain research, *IEEE Trans. SMC* Jan./Feb. 1987.

45. Brooks, V. B., *The Neural Basis of Motor Control*, Oxford Univ. Press, New York, 1986.

46. Marsden, C. D., The mysterious motor function of the basal ganglia, *Neurology (NY)* **32**, 514–539, 1982.

47. Shastri, L., Ed., *Connectionism Meets AI: Workshop Proceedings*, Univ. Pennsylvania, Philadelphia, Pa., 1990.

48. Werbos, P., The cytoskeleton: Why it may be crucial to human learning and to neurocontrol. *Nanobiology*, Vol. 1, No. 1.