N91-20817

NEUROCONTROL AND FUZZY LOGIC: CONNECTIONS AND DESIGNS

by Paul J. Werbos
Room 1151, National Science Foundation
Washington, D.C. 20550
(202)-357-9618

ABSTRACT

Artificial neural networks (ANNs) and fuzzy logic are complementary technologies. ANNs extract information from systems to be learned or controlled, while fuzzy techniques mainly use verbal information from experts. Ideally, both 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 or actuators, etc. -- uses five generalized designs, related to control theory, which 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; maximize performance or minimize cost over time, even in noisy environments. Design tradeoffs and future directions are discussed throughout.

It will begin by discussing the simpler, more common application of ANNs -- to learning a mapping from a vector $\underline{X}$ to a vector $\underline{Y}$. Then it will discuss neurocontrol, and the central importance of neurocontrol to understanding intelligence.

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

ANNs and Fuzzy Logic in General

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

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 backpropagation[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 $\underline{X}$ and $\underline{Y}$ 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 $\underline{X}$ is a set of data characterizing the state of a factory, and $\underline{Y}$ 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 $\underline{X}$.

Which method is better to use, when?

The simplest answer to this question is as follows: since ANNs extract knowledge from ~~training~~ databases, and fuzzy logic extracts rules from human experts, we should simply decide ~~who~~ 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 while expert judgment is only a secondary source; however, when the empirical data is 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 part and fuzzy logic for another part. For example, there may be an intermediate proposition R which has an important influence on Y; we may build a neural net to map from X to R, and a fuzzy logic system to map X and R into 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[3].

In the current literature, many people are using fuzzy logic as a kind of organizing framework, to help them subdivide a mapping from X to 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 in [4]. 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 X to 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[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 the exact same mathematical recipe for how to adapt a particular network, and use different names for the same thing. Personally, 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.

There are some problems which cannot be easily subdivided into expert-based parts and learning-based parts. For example, there are theories of international conflict which 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, one 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:

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

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 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.

So far as I know, the idea of applying backpropagation to a fuzzy logic network was first published in 1988[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 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; (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, which go beyond the capability of what ANN people call "feedforward" networks. The inference structure may be a "simultaneously recurrent" network. Nevertheless, backpropagation can be used on such inference structures (using the memory-saving methods in [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 which learn, through experience, how to perform more and more effective inference (in a nonBoolean environment). Inference may be more difficult than other forms of control problem; however, there are parallels between neurocontrol systems and existing inference engines which suggest some real possibilities here.

Stinchcombe and White have proven (IJCNN 1989) that conventional ANNs can represent essentially any well-behaved nonlinear mapping. However, in applications of ANNs, many researchers have begun to encounter the limitations of any static mapping. In recognizing dynamic patterns[1], like speech or moving targets, or in real-world diagnostics[9], 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[10], in part by adapting intermediate features and developing representations which an expert might not

have thought of.

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[11], 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 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.

****

INSERT FIGURE 1 (VENN DIAGRAM)

****

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 which 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 designs 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 will present numerous boxes labelled 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 often 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 we be interested in functional forms close to the conventional form used in ANNs[1]:

$$x_i \;=\; s\Big(\sum W_{ij} x_j\Big)\,, \qquad (2)$$

where:

$$s(z) \;=\; \frac{1}{1+e^{-z}} \quad ? \qquad (3)$$

(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 White and Gallant 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 or neural nets, and where both approaches involve a similar degree of computational complexity. (For example, 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 -- under encouragement from the U.S. Navy at China Lake (with some NSF support acknowledged in the documentation). 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 to haul along a Cray. Other companies -- such as Syntonics in the U.S. 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 recent 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 Research Laboratories 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[12] -- 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 thousand-fold or million-fold 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 which led to the neuroengineering program at NSF. 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 were 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 which comes from giving the user no choice of functional form) -- the method became much better known[13]. 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 national priority, because of the changing composition of the young adult population in the United States.

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

****

INSERT FIGURE 2 (BRAIN)

****

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 [10].) Because the mathematics involved are general mathematics, they 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 which 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 of engineering importance.

****

INSERT TABLE 1 (Matrix of capabilities versus methods)

****

Table 1 was developed two years ago[11], but it still applies to all the recent research which I am aware of (except that a very 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 which imitates a pre-existing control system; this is like expert systems, except that we copy what a person says instead of what he does, and can operate at higher speed. Direct inverse control builds neural nets which 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.

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.

Areas of Application

Four major areas have been discussed at length [10,11] for possible applications of neurocontrol:

- o Vehicles and structures
- o Robots and manufacturing (especially of chemicals)
- o Teleoperation and aid to the disabled
- o 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, back-up approach to solving very difficult problems which 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 space a realistic possibility. In October of 1990, NSF and McDonnell-Douglas are planning to jointly sponsor 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.

The chemical industry has also been quite active. 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 is also planning a workshop in October, 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 which mainstream academics are barely beginning to address.

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 -- which consists of sensor inputs ($\underline{X}$) and desired actions ($\underline{u}$). Once this training set is available, there are many neural network designs and learning rules (like basic backpropagation) which can learn the mapping from $\underline{X}$ to $\underline{u}$.

Usually, the training set is built up by asking a human expert to perform the desired task, and recording what the human sees ($\underline{X}$) and what the human does ($\underline{u}$). There are many variations of this, of course, depending on the task to be performed. (Sometimes the input to the human, $\underline{X}$, 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 which 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, which allows it to operate at a higher speed -- higher than what 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, which was too expensive or too slow to use in real-time, on-board 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 up 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 which 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 which 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[14] have shown 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.

Advanced practitioners of supervised control no longer think of it as a simple matter of mapping $\underline{X}(t)$, at time t, onto $\underline{u}(t)$. Instead, they use past information as well to predict $\underline{u}(t)$. 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[10].

Supervised control with an ANN was first performed by Widrow[15]. Kawato, in conversation, has stated that Fuji has widely demonstrated working robots based on supervised control. Many other applications have been published.

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.

****

INSERT FIGURE 3 (Direct Inverse Control)

****

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 which move the hand to that point. If the original mapping from $\underline{\theta}$ to $\underline{X}$ 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 awhile, 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[11] has used direct inverse control to achieve great accuracy (error less than 0.1%) in controlling an actual, physical Puma robot. Morita[14] has used direct inverse control with a fuzzy network, but with an ANN learning rule, and claims that this is better than supervised control for the same problem.

In direct inverse control, as in supervised control, it works better to think of the mapping problem in a dynamic context[10], 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.)

Direct inverse control does not work when the original map from $\underline{\theta}$ to $\underline{X}$ is not invertible. For example, if the degrees of freedom of the control variables (like $\underline{T}$) are more or less than the degrees of freedom of the observable (like $\underline{X}$), there is a problem. Eckmiller[16] has 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 [10]) and Jordan [17] 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.

Neural Adaptive Control

Neural adaptive control tries 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 [10,18-20].

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[10].

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 which 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[10]. 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[10] both in control and in system identification; the tradeoffs 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 Liapunov function [18]. In nonlinear adaptive control, it is far more difficult[20]. In actuality, however, the "Critic" networks to be discussed below function very much like Liapunov functions (especially in the BAC design). For many complex, nonlinear problems, it may be necessary to adapt a Liapunov function after the fact, and verify its properties after the fact, rather than specify it in advance.

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, $\underline{u}(t)$. 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[10,11].) In most cases, there will also be a Model network, which inputs a current description of reality, $\underline{R}(t)$, and the action vector $\underline{u}(t)$; it outputs a forecast of $\underline{R}(t+1)$ and of $\underline{X}(t+1)$, 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 $\underline{R}(t)$ and possibly $\underline{u}(t)$, 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 which predicts the optimal future payoff (over all future times) starting from a given value for $\underline{R}(t+1)$, and to use that network as the basis for choosing $\underline{u}(t)$. 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.

****

INSERT FIGURE 4 (Backpropagating Utility)

****

In the backpropagation of utility, we must start with a Model network which has already been adapted, and a Utility network which 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, $\underline{X}(0)$, and use the initial weights in the Action network to predict $\underline{X}(t)$ 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[21]. 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 [10]. There was Jordan's robot arm controller[17], and my own official DOE model of the natural gas industry[22]. Recently, Narendra and Hwang have reported success with this method.

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 [22] 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 which inputs $\underline{X}(t)$ and $\underline{u}(t)$, which uses $\underline{X}(t+1)$ 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 [10].

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 forecasted 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 which can be used to represent noise, even when backpropagating utility, but they are somewhat ad hoc and inefficient[10].

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.23], but those techniques are either very approximate or do not scale well with large problems or both; in any case, Narendra[19] has questioned the stability of such methods. Nevertheless, even if we backpropagate utility in an offline mode, we can still develop a network which adapts in real-time to changes in slowly-varying parameters; we can "learn offline to be adaptive online." [10]. 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 2-Net structures, which work well on small problems, through to complex hybrids, which hopefully encompass what

goes on in the human brain[10,11].

Adaptive critic methods may be defined, in broad terms, as methods which attempt to approximate dynamic programming as first described in [24]. 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.

****

Figure 5 (inputs and outputs of dynamic programming)

****

Dynamic programming requires as its input a utility function U and a model of the external environment, 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. After you know the function J and the model F, it is then a simple problem in function maximization to pick the actions which maximize J. The notation here is taken from Raiffa[25], whose books on decision analysis may 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 [26] and [27].) Adaptive critic methods may be defined more precisely as methods which 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 [28]):

****

Table 2 (Examples of J and U)

****

Please 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 which 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[29] came from Freud's theory that emotional charge is passed backwards from object to object, with a strength proportionate to the usual forwards association between the two objects[30]. 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 2-network arrangement of Barto, Sutton and Anderson[31], 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

****

Figure 6. The 2-Net Design of Barto, Sutton and Anderson

****

punishment signal. This design has worked well on a wide variety of real-world problems, including robotics[32], autonomous vehicles and fuzzy logic systems. Williams, in [20], has 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 which 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 which can help, discussed by myself, by Barto, and by Sutton [10,11,20].

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;
(2) the Action-Dependent Adaptive Critic (ADAC), shown in Figure 8.

****

Insert Figures 7 and 8: BAC and ADAC (as adapting Action net)

****

The BAC design is closer to dynamic programming than is the 2-net design, because there is a more explicit attempt to pick $\underline{u}(t)$ 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 $\underline{u}(t)$. 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 $\underline{u}(t)$ 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[10].

ADAC [26,27] 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 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 [20]), but found the resulting Critic network to be rather complex. In an ideal world, one would want to combine both approaches, so as to combine the modularity and cleanliness of BAC with the model-independent robustness of ADAC; however, BAC may be good enough by itself in many applications. Jameson has reported some preliminary results with BAC[33], and other aerospace-oriented researchers may have dealt with larger applications; however, more work is needed. Whatever the details, the adaptation of 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 which are more or less equivalent[34] and which have severe scaling problems. There are two alternatives which should scale much better: (1) Dual Heuristic Programming (DHP), which outputs estimates of the derivatives of J; (2) Globalized DHP (GDHP), which outputs an estimate of J (or its components), but which 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 [23,24], but are described in more modern language in [10] and [11]. 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.

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 recommended 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; (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 together 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 one 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 -- which affected the original forecast, in one quick sweep through the process. It would report back the ten or the hundred 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[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 two 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 [8]) to identify the path of connections which 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 three would be an extended version of tools one or two. Instead of first derivatives, it would provide information based on low-cost second derivatives (as described in [23], based on calculations like those in [5,11]). 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 four 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 five would be a model calibration tool, based on the backpropagation of error, and robust estimation concepts like those of [10]. 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" which usually goes into most complex models in the real world for calibration purposes. Tool number six would go back and identify how the resulting parameter estimates or rules were influenced by different cases in the input dataset; this would provide an integrated, nonlinear version of the highly respected linear diagnostic tools developed by Belsley,

Kuh and Welsch[35].

These six tools are the most obvious needed tools, exploiting backpropagation, but a host of other tools are possible involving estimation diagnostics, decision diagnostics and convergence tools. 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 described in [28].

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 man-machine communication, so as to exploit the capabilities of both forms of intelligence.

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 tradeoff between "general syntheses" -- which combine everything but require the expense of implementing everything -- and direct, simple designs tuned to particular concrete problems. Given the natural human tendency towards 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 is a wide variety of designs available, ranging from simple off-the-shelf technologies (easily applied to fuzzy logic networks) through to areas where fundamental research is still needed and vital to our understanding of real intelligence.

REFERENCES

1. P.Werbos, Backpropagation through time: What it does and how to do it, Proceedings of the IEEE, October 1990.

2. A.Amano et al, On the use of neural networks and fuzzy logic in speech recognition. In Proceedings of the International Joint Conference On Neural Networks (IJCNN). New York: IEEE, June 1989.

3. J. Schreinemakers and D. Touretzky, Interfacing a neural network with a rule-based reasoner for diagnosing mastitis. IJCNN Proceedings. Hillsdale, NJ: Erlbaum, January 1990.

4. H. Takagi, Fusion technology of fuzzy theory and neural networks. In Proceedings of Fuzzy Logic and Neural Networks, Izzuka, Japan, 1990.

5. P.Werbos, Backpropagation: past and future. In Proceedings of the Second International Conference on Neural Networks. New York: IEEE, 1988. A transcript of the actual talk, with slides, is available from the author, and covers some additional topics.

6. L.Fu, Backpropagation in neural networks with fuzzy conjunction units. IJCNN Proceedings. New York: IEEE, June 1990.

7. L. Hsu et al, Fuzzy logic in connectionist expert systems. IJCNN Proceedings. Hillsdale, NJ: Erlbaum, January 1990.

8. P.Werbos, Generalization of Backpropagation, Neural Networks, October 1988. When simultaneous recurrence is not present, the calculations are much simpler, as in [22].

9. P. Werbos, Making diagnostics work in the real world: a few tricks. In A. Maren (ed.), Handbook of Neural Comp. Appl. Academic Press, 1990.

10. P.Werbos. Neurocontrol and related techniques. In A. Maren (ed.), Handbook of Neural Comp. Appl.. Academic Press, 1990.

11. W.T.Miller, Sutton and Werbos (eds), Neural Networks for Control. Cambridge, Mass.: MIT

Press, 1990.

12. C. Mead, Analog VLSI and Neural Systems. Reading, Mass.: Addison-Wesley, 1989.

13. P.Werbos, Links between artificial neural networks (ANN) and statistical pattern recognition. In I.Sethi and Jain (eds), Artificial Neural Networks and Statistical Pattern Recognition: Old and New Connections. Elsevier, forthcoming.

14. Morita et al, Fuzzy knowledge model of neural network type. IJCNN Proceedings. Hillsdale, NJ: Erlbaum, 1990.

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

16. R. Eckmiller et al, Neural kinematics net for a redundant robot arm. In IJCNN Proceedings. New York: IEEE, June 1989.

17. M. Jordan, Generic constraints on underspecified target trajectories. In IJCNN Proceedings. New York: IEEE, June 1989.

18. K. Narendra and Annaswamy, Stable Adaptive Systems. Englewood, NJ: Prentice-Hall, 1989.

19. K. Narendra and Parthasarathy, Identification and control of dynamical systems using neural networks,IEEE Trans. Neural Networks, Vol. 1, No. 1 (March 1990).

20. K. Narendra (ed), Proceedings of the Sixth Yale Workshop on Adaptive and Learning Control. New Haven, Conn.: K. Narendra, Yale, 1990.

21. P. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. thesis to Harvard U. Committee on Applied Mathematics, November 1974.

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

23. P. Werbos, Applications of advances in nonlinear sensitivity analysis. In R. Drenick ans Kozin (eds), Systems Modeling and Optimization: Proceedings of the International Federation for

Information Processing. New York: Springer-Verlag, 1982.

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

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

26. P.Werbos,Neural networks for control and system identification. In IEEE CDC Proceedings. New York: IEEE, 1989.

27. G.Lukes, B. Thompson and P.Werbos, Expectation driven learning with an associative memory. In IJCNN Proceedings (Washington). Hillsdale, NJ: Erlbaum, 1990.

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

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

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

31. A.Barto, Sutton and Anderson, Neuron-like adaptive elements that can solve difficult learning control problems, IEEE Trans. SMC. SMC-13, p. 834-846.

32. J. Franklin, Reinforcement of robot motor skills through reinforcement learning. In IEEE/CDC Proceedings. New York: IEEE, 1988.

33. J.Jameson, A neurocontroller based on model feedback and the Adaptive Heuristic Critic. In IJCNN Proceedings (San Diego). New York: IEEE, June 1990.

34. P. Werbos, Consistency of HDP applied to a simple reinforcement learning problem, Neural Networks, Vol. 3, p.179-189, March 1990.

35. D.Belsley, Kuh and Welsch, Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. Wiley, 1980.

LIST OF FIGURES, TABLES AND CAPTIONS

Figures

Tables

FIGURE 1

WHAT IS NEUROCONTROL?

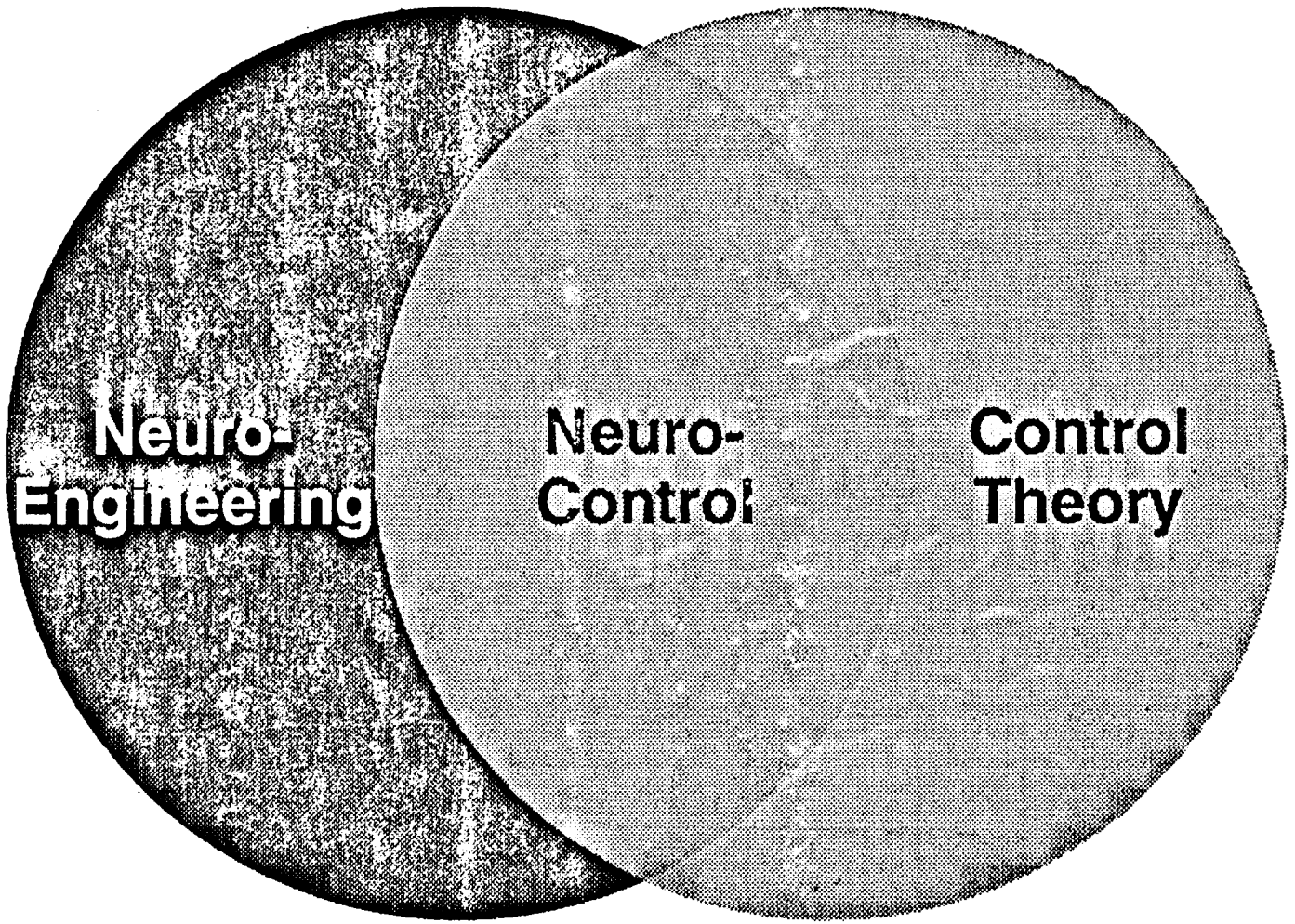


Figure 2

CAN WE DESIGN AND UNDERSTAND INTELLIGENCE?

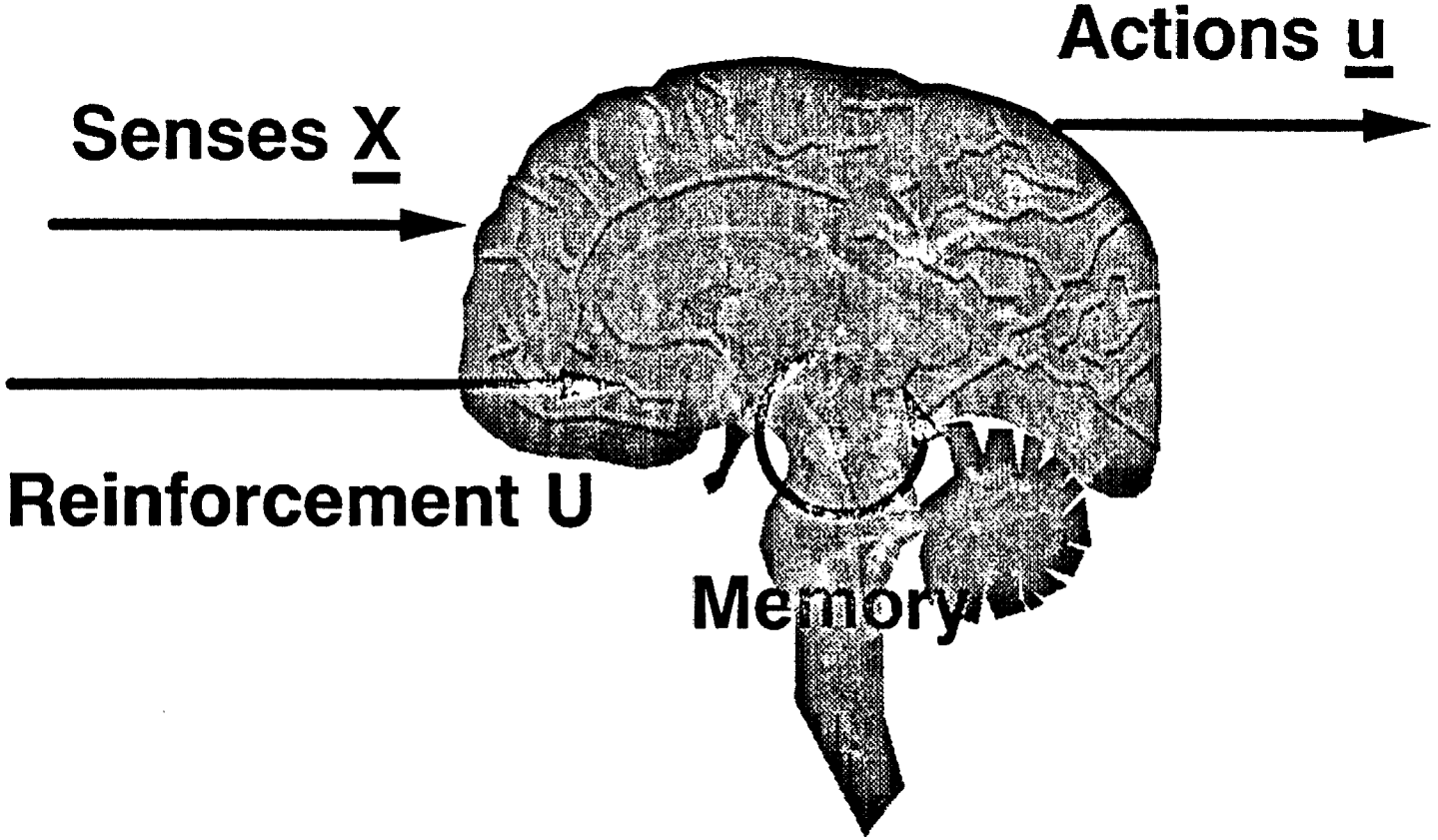
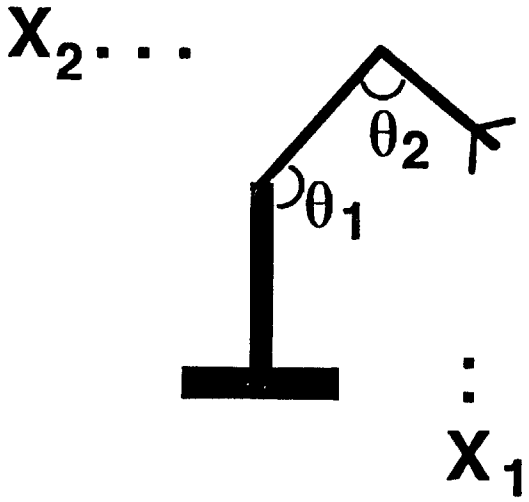


Figure 3

~~BASIC INVERSE CONTROL~~



$$X_1, X_2 = f(\theta_1, \theta_2)$$

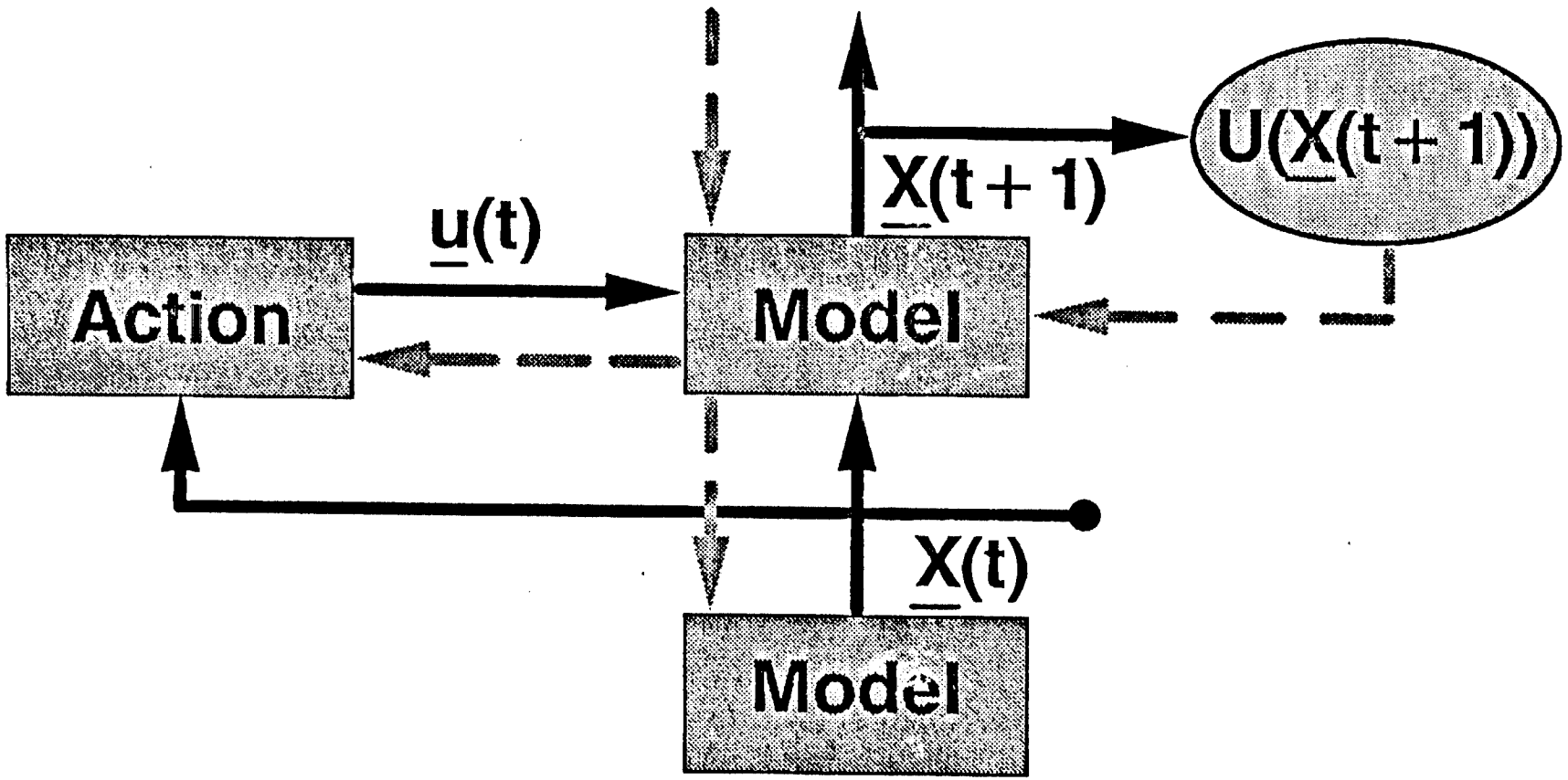
$$\theta_1, \theta_2 = f^{-1}(X_1, X_2)$$

Input X_1, X_2; Target θ_1, θ_2

~~Miller: X (t-1), Low Error~~

~~Jordan, Kawato Not Basic~~

Figure 4



BTT: Werbos, Jordan, Widrow, Kawato

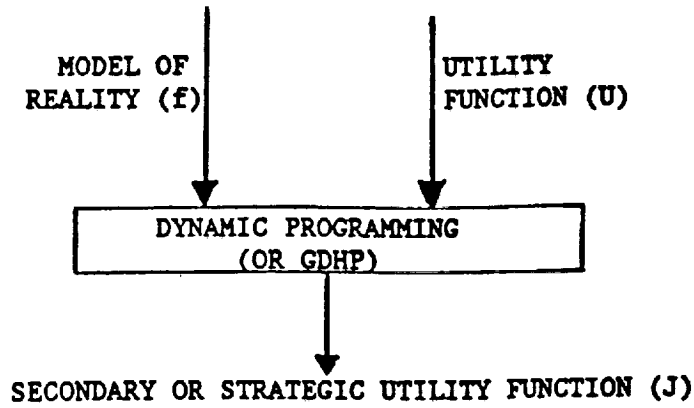
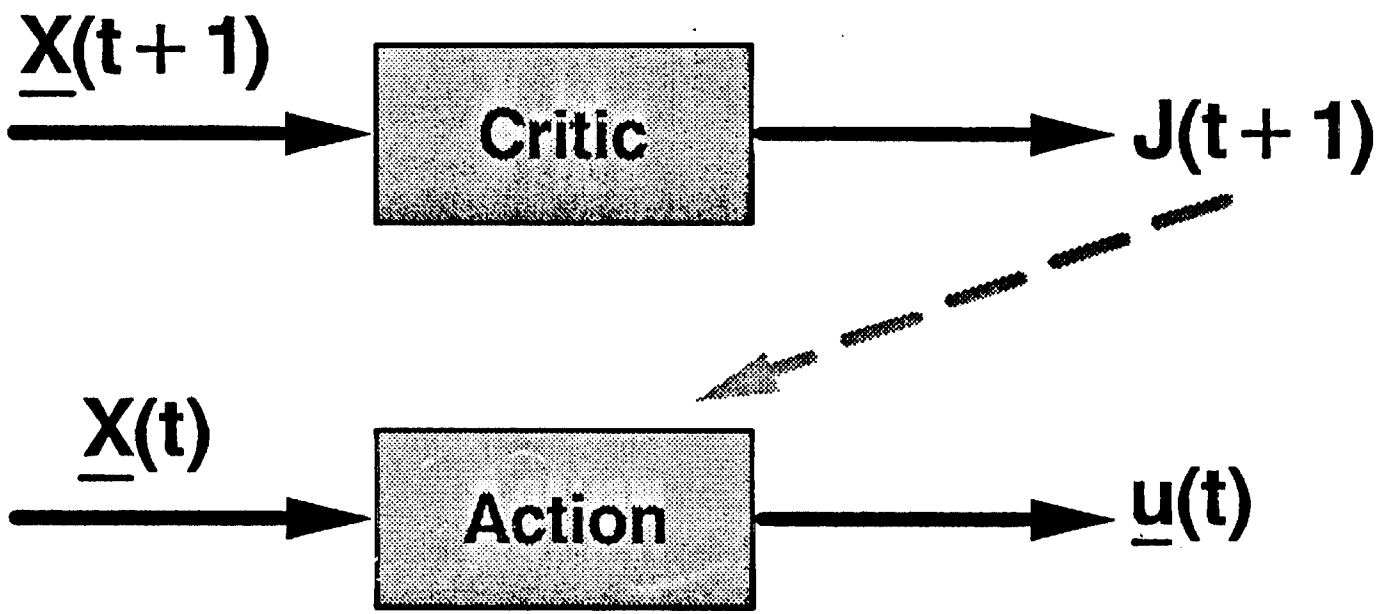
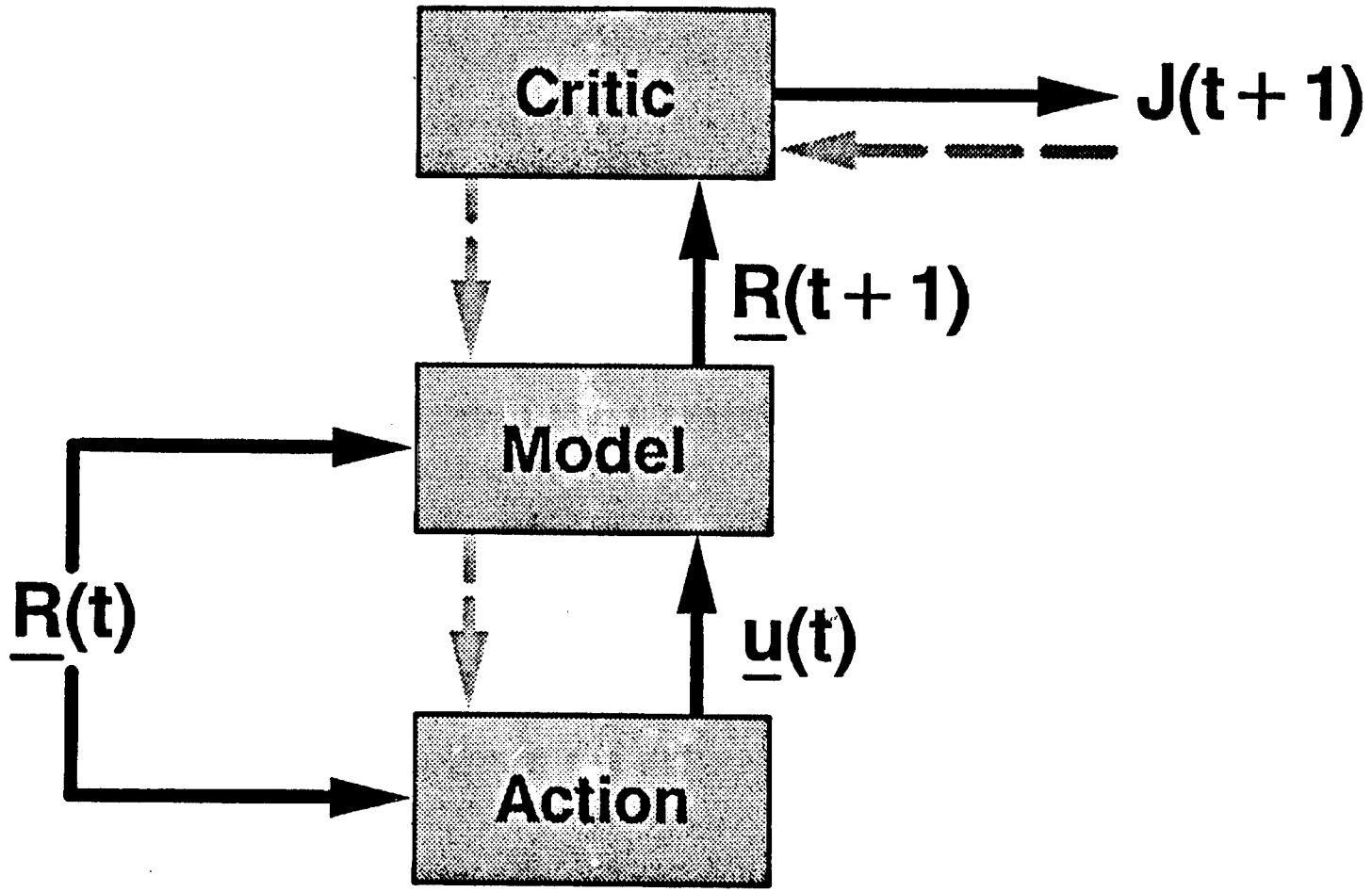


Figure 5

Figure 6



2-Net Adaptive Critic



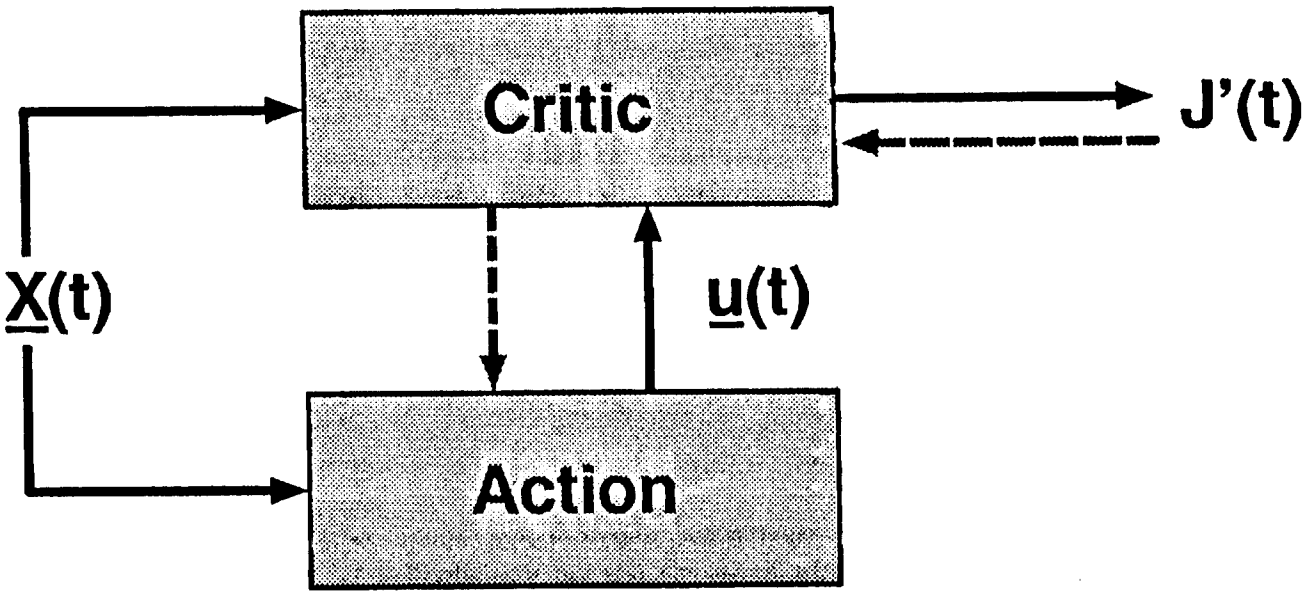
Backpropagated Adaptive Critic

TABLE 1

	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				
2-Net		X	X	X
BAC+DHP,etc.	X	X	X	X

Figure 8

ADAC



Action-Dependent Adaptive Critic [2,6]

TABLE 2

Examples of Strategic Utility (J)

Domain	Basic Utility (U)	Strategic Utility (J)
Chess	Win/Lose	Queen = 9 points, etc.
Business Theory	Current Profit Cash Flow	Present Value of Strategic Assets (Performance Measures)
Human Thought	Pleasure/Pain Hunger	Hope/Fear Reaction to Job Loss
Behavioral Psychology	Primary Reinforcement	Secondary Reinforcement
Artificial Intelligence	Utility Function	Static 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")

Actually, the items in the last row of this table refer to the derivatives of the two kinds of utility. They refer to the change in overall utility which would result from a small change in your level of consumption of a particular good. Likewise, the benefit of increasing a particular measure of environmental quality is really just the derivative of J with respect to that measure. In Freudian psychology, the emotional charge associated with a particular object is really just the derivative of J with respect to the corresponding variable. (All of these examples could be discussed at greater length; for example, I would contend that human "pleasure" includes parental pleasure in experiencing happy children, but this issue goes well beyond the theory of intelligence as such.)

Derivatives are vital to the operation of any intelligent system. For example, when SAS estimates the parameters of a nonlinear model, it starts from the initial guesses which you provide for these parameters. It adjusts these guesses upwards or downwards, so as to minimize forecasting error over historical data. In order to do this efficiently, it has to know whether a small positive adjustment of any parameter would increase error, decrease error, or leave the error unchanged; in other words it has to know the derivative of error with respect to every parameter.