1 ADP: Goals, Opportunities and Principles

PAUL WERBOS
National Science Foundation

1.1 GOALS OF THIS BOOK

Is it possible to build a general-purpose learning machine, which can learn to maximize *whatever* the user wants it to maximize, over time, in a strategic way, even when it starts out from zero knowledge of the external world? Can we develop general-purpose software or hardware to "solve" the Hamilton-Jacobi-Bellman equation for truly large-scale systems? Is it possible to build such a machine which works effectively in practice, even when the number of inputs and outputs is as large as what the mammal brain learns to cope with? Is it possible that the human brain itself is such a machine, in part? More precisely, could it be that the same mathematical design principles which we use to build rational learning-based decision-making machines are also the central organizing principles of the human mind itself? Is it possible to convert all the great rhetoric about "complex adaptive systems" into something that actually works, in maximizing performance?

Back in 1970, few informed scientists could imagine that the answers to these questions might ultimately turn out to be yes, if the questions are formulated carefully. But as of today, there is far more basis for optimism than there was back in 1970. Many different researchers, mostly unknown to each other, working in different fields, have independently found promising methods and strategies for overcoming obstacles that once seemed overwhelming.

This section will summarize the overall situation and the goals of this book briefly but precisely, in words. Section 1.2 will discuss the needs and opportunities for ADP systems across some important application areas. Section 1.3 will discuss the core mathematical principles which make all of this real.

Goal-oriented people may prefer to read Section 1.2 first, but bottom-up researchers may prefer to jump immediately to Section 1.3. The discussion of goals here and in Section 1.2 is really just a brief summary of some very complex lessons learned over time, which merit much more detailed discussion in another context.

This book will focus on the first three questions above. It will try to bring together the best of what is known across many disciplines relevant to the question: how can we develop *better* general-purpose tools for doing optimization over time, by using learning and approximation to allow us to handle *larger-scale, more difficult* problems? Many people have also studied the elusive subtleties of the follow-on questions about the human brain and the human mind [1]; however, we will never be able to construct a good straw-man model of how the mammalian brain achieves these kinds of capabilities, until we understand what these capabilities require of *any* information processing system on *any* hardware platform, wet, dry, hard or ethereal. The first three questions are a difficult enough challenge by themselves.

As we try to answer these three questions, we need to constantly re-examine what they actually mean. What are we trying to accomplish here? What is a "general-purpose" machine? Certainly no one will ever be able to build a system which is guaranteed to survive when it is given just a microsecond to adapt to a totally crazy and unprecedented kind of lethal shock from out of the blue. Mammal brains work amazingly well, but even they get eaten up sometimes in nature. But certainly, we need to look for some ability to learn or converge at reasonable speed (as fast as possible!) across a wide variety of complex tasks or environments. We need to focus on the long-term goal of building systems for the general case of nonlinear environments, subject to random disturbances, in which our intelligent system gets to observe only a narrow, partial window into the larger reality in which it is immersed.

In practical engineering, we, like nature, will usually not want to start our systems out in a state of zero knowledge. But even so, we can benefit from using learning systems powerful enough that they *could* converge to an optimum, even when starting from zero. Many of us consider it a gross abuse of the English language when people use the word "intelligent" to describe high-performance systems which are unable to learn anything fundamentally new, on-line or off-line.

In many practical applications today, we can use ADP as a kind of offline numerical method, which tries to "learn" or converge to an optimal adaptive control policy for a particular type of plant (like a car engine). But even in offline learning, convergence speed is a major issue, and similar mathematical challenges arise.

Is it *possible* to achieve the kind of general-purpose optimization capability we are focusing on here, as the long-term objective? Even today, there is only one exact method for solving problems of optimization over time, in the general case of nonlinearity with random disturbance: dynamic programming (DP). But exact dynamic programming is used only in niche applications today, because the "curse of dimensionality" limits the size of problems which can be handled. Thus in many engineering applications, basic stability is now assured via conservative design, but overall performance is far from optimal. (It is common to optimize or tweak a control parameter here and there, but that is not at all the same as solving the

dynamic optimization problem.) Classical, deterministic design approaches cannot even address questions like: How can we minimize the *probability* of disaster, in cases where we cannot totally guarantee that a disaster is impossible?

There has been enormous progress over the past ten years in increasing the scale of what we can handle with *approximate* DP and learning; thus the goals of this book are two-fold: (1) to begin to unify and consolidate these recent gains, scattered across disciplines; and (2) to point the way to how we can bridge the gap between where we are now and *the next major watershed* in the basic science — the ability to handle tasks as large as what the mammal brain can handle, and even to connect to what we see in the brain.

The term "ADP" can be interpreted either as "Adaptive Dynamic Programming" (with apologies to Warren Powell) or as "Approximate Dynamic Programming" (as in much of my own earlier work). The long-term goal is to build systems which include *both* capabilities; therefore, I will simply use the acronym "ADP" itself. Various strands of the field have sometimes been called "reinforcement learning" or "adaptive critics" or "neurodynamic programming," but the term "reinforcement learning" has had many different meanings to many different people.

In order to reach the next major watershed here, we will need to pay more attention to two major areas: (1) advancing ADP as such, the main theme of this book; (2) advancing the critical subsystems we will need as components of ADP systems, such as systems that learn better and faster how to make predictions in complex environments.

1.2 FUNDING ISSUES, OPPORTUNITIES AND THE LARGER CONTEXT

This book is the primary product of an NSF-sponsored workshop held in Mexico in April 2002, jointly chaired by Jennie Si and Andrew Barto. The goals of the workshop were essentially the same as the goals of this book. The workshop covered some very important further information, beyond the scope of the book itself, posted at http://www.eas.asu.edu/~nsfadp

The potential of this newly emerging area looks far greater if we can *combine* what has been achieved across all the relevant disciplines, and put the pieces all together. There are substantial unmet opportunities here, including some "low lying fruit." One goal of this book is to help us see what these opportunities would look like, if one were to bring these strands together in a larger, more integrated effort. But as a practical matter, large funding in this area would require three things: (1) more proposal pressure in the area; (2) more unified and effective communication of the larger vision to the government and to the rest of the community; (3) more follow-through on the vision *within* the ADP community, including more cross-disciplinary research and more unified education.

Greater cross-disciplinary cooperation is needed for intellectual reasons, as you will see from this book. It is also needed for practical reasons. The engineering and

operations research communities have done a good job on the applications side, on the whole, while the computer science communities have done a good job on educational outreach and infrastructure. We will need to combine both of these together, more effectively, to achieve the full potential in either area.

At a lower level, the Control, Networks and Computational Intelligence (CNCI) program in the Electrical and Communication Systems (ECS) Division of NSF has funded extensive work related to ADP, including earlier workshops which led to books of seminal importance [2, 3]. The center of gravity of CNCI has been the IEEE communities interested in these issues (especially in neural networks and in control theory), but ECS also provided some of the early critical funding for Barto, Bertsekas and other major players in this field. ADP remains a top priority in CNCI funding. It will benefit the entire community if more proposals are received in these areas. The Knowledge and Cognitive Systems program in Intelligent Information Systems (IIS) Division of NSF has also funded a great deal of work in related areas, and has joined with ECS in joint-funding many projects.

In the panels that I lead for CNCI, I ask them to think of the funding decision itself as a kind of long-term optimization problem under uncertainty. The utility function to be maximized is a kind of 50-50 sum of two major terms — the potential benefit to basic scientific understanding, and the potential benefit to humanity in general. These are also the main considerations to keep in mind when considering possible funding initiatives.

Occasionally, some researchers feel that these grand-sounding goals are too large and too fuzzy. But consider this analogy. When one tries to sell a research project to private industry, one *must* address the ultimate bottom line. Proposals which are not well-thought-out and specifically tuned to maximize the bottom line usually do not get funded by industry. The bottom line is different at NSF, but the same principle applies. Strategic thinking is essential in all of these sectors. All of us need to reassess our work, strategically, on a regular basis, to try to maximize our own impact on the larger picture.

The biggest single benefit of ADP research, in my personal view, is based on the hope that answers to the first three questions in Section 1.1 will be relevant to the further questions, involving the brain and the mind. This hope is a matter for debate in the larger world, where it contradicts many strands of inherited conventional wisdom going back for centuries. This book cannot do justice to those complex and serious debates, but the connection between those debates and what we are learning from ADP has been summarized at length elsewhere (e.g. [9, 52], with reference to further discussions).

1.2.1 Benefits to Fundamental Scientific Understanding

In practice, the fundamental scientific benefit of most of the proposals which I see comes down to the long-term watershed discussed above. There are few questions as fundamental in science as "What is Mind or Intelligence?" Thus I generally ask panelists to evaluate what the impact might be of a particular project in allowing us

to get to the watershed earlier than we would without the project. Since we will clearly need some kind of ADP to get there, ADP becomes the top scientific priority. But, as this book will bring out, powerful ADP systems will also require powerful *components* or *subsystems*. This book will touch on some of the critical needs and opportunities for subsystems, such as subsystems which learn to predict or model the external world, subsystems for memory, subsystems for stochastic search and subsystems for more powerful function approximation. This book may be uniquely important in identifying what we need to get from such subsystems, but there are other books which discuss those areas in more detail. CNCI funds many projects in those areas from people who may not know what ADP is, but are providing capabilities important to the long-term goals. Likewise, there are many projects in control system design which can feed into the long-term goal here, with or without an explicit connection to ADP. In every one of these areas, greater unification of knowledge is needed, for the sake of deeper understanding, more effective education, and reducing the common tendencies toward reinventing or spinning wheels.

1.2.2 Broader Benefits to Humanity

Optimal performance turns out to be critical to a wide variety of engineering and management tasks important to the future of humanity — tasks which provide excellent test-beds or drivers for new intelligent designs. The ECS Division has discussed the practical needs of many major technology areas, directly relevant to the goals of achieving sustainable growth here on earth, of cost-effective settlement of space, and of fostering human potential in the broadest sense. ADP has important potential applications in many areas, such as manufacturing [3], communications, aerospace, Internet software and defense, all of interest to NSF; however, test-beds related to energy and the environment have the closest fit to current ECS activities.

This past year, ECS has played a major role in three cross-agency funding activities involving energy and the environment, where ADP could play a critical role in enabling things which could not be done without it. It looks to me as if all three will recur or grow, and all three are very open to funding partnerships between ADP researchers and domain experts. The three activities focus on: (1) new crossdisciplinary partnerships addressing electric power networks (EPNES); (2) space solar power (JIETSSP); and (3) sustainable technology (TSE). To learn about these activities in detail, search on these acronyms at http://www.nsf.gov. Here I will talk about the potential role of ADP in these areas.

1.2.2.1 Potential Benefits In Electric Power Grids Computational intelligence is only one of the core areas in the CNCI program. CNCI is also the main funding program in the US government for support of electric utility grid research. CNCI has long-standing ties with the IEEE Power Engineering Society (PES) and the Electric Power Research Institute (EPRI). Transitions and flexibility in the electric power sector will be crucial to hopes of achieving a sustainable global energy system. Many

people describe the electric power system of the Eastern United States as "the largest, most complicated single machine ever built by man."

The workshop led by Si and Barto was actually just one of *two* coordinated workshops held back-to-back in the same hotel in Mexico. In effect, the first workshop asked: "How can we develop the algorithms needed in order to better approximate optimal control over time of extremely large, noisy, nonlinear systems?" The second workshop, chaired by Ron Harley of the University of Natal (South Africa) and Georgia Tech, asked: "How can we develop the necessary tools in order to do integrated global optimization over time of the electric power grid as one single system?" It is somewhat frightening that some engineers could not see how there could be any connection between these two questions, when we were in the planning stages.

The Harley workshop — sponsored by James Momoh of NSF and Howard University, and by Massoud Amin of EPRI and the University of Minnesota — was actually conceived by Karl Stahlkopf, when he was a Vice-President of EPRI. Ironically, Stahlkopf urged NSF to co-sponsor a workshop on that topic as a kind of act of reciprocity for EPRI co-sponsoring a workshop I had proposed (when I was temporarily running the electric power area) on hard-core transmission technology and urgent problems in California. Stahlkopf was very concerned by the growing difficulty of getting better whole-system performance and flexibility in electric power grids, at a time when nonlinear systems-level challenges are becoming more difficult but control systems and designs are still mainly based on piecemeal, local, linear or static analysis. "Is it possible," he stressed, "to optimize the *whole* thing in an integrated way, as a single dynamical system?"

It will require an entire new book to cover all the many technologies and needs for optimization discussed at the second workshop. But there are five points of particular relevance here.

First, James Momoh showed how his new Optimal Power Flow (OPF) system — being widely distributed by EPRI — is much closer to Stahlkopf's grand vision than anything else now in existence.

Real-world suppliers to the electric utility industry have told me that OPF is far more important to their clients than all the other new algorithms put together. OPF does provide a way of coordinating and integrating many different decisions across many points of the larger electric power system. Momoh's version of OPF — using a combination of nonlinear interior point optimization methods and genetic algorithms — has become powerful enough to cope with a growing variety of variables, all across the system. But even so, the system is "static" (optimizes at a given time slice) and deterministic. Stahlkopf's vision could be interpreted *either* as: (1) extending OPF to the dynamic stochastic case — thus creating dynamic stochastic OPF (DSOPF); (2) applying ADP to the electric power grid. The chances of success here may be best if *both* interpretations are pursued together in a group which understands how they are equivalent, both in theory and in mathematical details. Howard University may be particularly well equipped to move toward DSOPF in this top-down, mathematically-based approach.

Some researchers in electric power are now working to extend OPF by adding the "S" but not the "D". They hope to learn from the emerging area of stochastic programming in operations research. This can improve the treatment of certain aspects of power systems, but it does provide a capability for anticipatory optimization. It does not provide a mechanism for accounting for the impact of present decisions on future situations. For example, it would not give credit to actions that a utility system could take at present in order to prevent a possible "traffic jam" or breakdown in the near future. Simplifications and other concepts from stochastic programming could be very useful within the context of a larger, unifying ADP system, but only ADP is general enough to allow a unified formulation of the entire decision problem.

Second, Venayagamoorthy of Missouri-Rolla presented important results which form a chapter of this book. He showed how DHP — a particular form of ADP design — results in an integrated control design for a system of turbogenerators which is able to keep the generators up, even in the face of disturbances three times as large as what can be handled by the best existing alternative methods. This was shown on an actual experimental electric power grid in South Africa. The neural network models of generator dynamics were trained in real time as part of this design. The ability to keep generators going is very useful in a world where containing and preventing system-wide blackouts is a major concern. This work opens the door to two possible extensions: (1) deployment on actual commercially running systems, in areas where blackouts are a concern or where people would like to run generators at a higher load (i.e. with lower stability margins); (2) research into building up to larger and larger systems, perhaps by incorporating power-switching components into the experimental networks and models. This may offer a "bottom-up" pathway toward DSOPF. This work was an outcome of a CNCI grant to fund a partnership between Ron Harley (a leading power engineer) and Don Wunsch (a leading researcher in ADP from the neural network community).

Third, the DHP system used in this example outputs value signals which may be considered as "price signals" or "shadow prices." They represent the *gradient* of the more conventional scalar value function used in the Bellman equation or in simpler ADP designs. They are generalizations of the Lagrange Multipliers used in Momoh's OPF system. They may also offer new opportunities for a better interface between the computer-based grid control systems and the money-based economic control systems.

Fourth, there is every reason to worry that the technology used by Venayagamoorthy — which works fine for a dozen to a few dozen continuous state variables — will start to break down as we try to scale up to thousands of variables, *unless* the components of the design are upgraded, as I will discuss later in this chapter. The effort to scale up is, once again, the core research challenge ahead of us here. One warning: some writers claim to handle a dozen state variables when they actually mean that they handle a dozen possible states in a finite-state Markhov chain system; however, in this case, I really am referring to a state space which is R12, a twelve-dimensional space defined by continuous variables, which is highly nonlinear, and not reducible to clusters around a few points in that space.

Finally, ADP researchers who are interested in exploring this area are strongly urged to look up "EPNES" at http://www.nsf.gov, and look in particular at the benchmark problems discussed there. The benchmark problems there, developed in collaboration between the two sponsors (NSF and the Navy), were designed to make it as easy as possible for non-power-engineers to try out new control approaches. As a practical matter, however, EPNES is more likely to fund partnerships which do include a collaborator from hard-core power engineering.

1.2.2.2 Potential Benefits in Space Solar Power (SSP) The Millennium Project of the United Nations University (http://millennium-project.org) recently asked decision makers and science policy experts all over the world: "What challenges can science pursue whose resolution would significantly improve the human condition?" The leading response was: "Commercial availability of a cheap, efficient, environmentally benign non-nuclear fission and non-fossil fuel means of generating base-load electricity, competitive in price with today's fossil fuels." Space solar power is a high risk vision for future technology, but many serious experts believe that it is the most promising of the very few options for meeting this larger need. (Earth-based solar is another important option.) Thus in March 2002, NASA, NSF and EPRI issued a solicitation for proposals called the "Joint Investigation of Enabling Technologies for Space Solar Power (JIETSSP)." John Mankins of NASA and I served as co-chairs of the Working Group which managed this activity.

Reducing cost was the number one goal of this effort. For each proposal, we asked the reviewers to consider: What is the potential impact of funding this work on reducing the time we have to wait, before we know enough to build SSP systems which can safely beat the cost of nuclear power in developing regions like the Middle East? We also asked the usual questions NSF asks about the potential benefits to basic science and other impacts.

Earlier proposals for SSP, dating back to the 1960's and 1970's, could not meet this kind of cost target, for several reasons. One of the key problems was the enormous cost of sending up humans to do all of the assembly work in space. Consider the total costs of sending up six people to live in a space station, multiply by a thousand, and you begin to see how important it is to reduce this component of cost as much as possible.

JIETSSP was partly the outcome of a workshop organized by George Bekey of USC in April 2000, jointly sponsored by NSF and NASA. The original goal was to explore how radically new approaches to robotics, such as robots building robots or robots with real intelligence, might be used to reduce costs either in space solar power or in building earth-based solar power systems.

The most practical concept to emerge from that workshop was the concept of "teleautonomy" as described by Rhett Whittaker of Carnegie-Mellon. Truly autonomous, self-replicating robots may indeed become possible, using ADP and other new technologies, but we do not really need them for SSP, and we need to make SSP affordable as soon as possible. But by the same token, we cannot afford to wait until

domain specialists in robotics develop complete three-dimensional physics models of every possible task and every possible scenario that might occur in space.

Whittaker proposed that we plan for a system in which hundreds of human operators (mostly on earth) control hundreds or thousands of robots in space, to perform assembly and maintenance. He proposed that we build up to this capability in an incremental way, by first demonstrating that we can handle the full range of required tasks in a more traditional telerobotics mode, and then focus on coordination and cost reduction. Ivory tower robotics research can easily become a gigantic, unfocused, nonproductive swamp; Whittaker's kind of strategy may be crucial to avoiding such pitfalls. Greg Baiden, of Penguin ASI in Canada and Laurentian University, has implemented real-world teleautonomy systems used in the mining industry which can provide an excellent starting point. Lessons from assembling the International Space Station need to be accounted for as well.

In the short term, the most promising role for ADP in SSP robotics may be mid-level work, analogous to the work of Venayagamoorthy in electric power. Venayagamoorthy has developed better sensor-based control policies, flexible enough to operate over a broader range of conditions, for a *component* of the larger power system. In the same way, ADP could be used to improve performance or robustness or autonomy for *components* of a larger teleautonomy system. Half of the intellectual challenge here is to scope out the testbed challenges which are really important within realistic, larger systems designs in the spirit of Whittaker or Baiden.

What kinds of robotics tasks could ADP really help with, either in space or on earth?

Practical robotics in the United States often takes a strictly domain-specific approach, in the spirit of expert systems. But the robotics field worldwide respects the achievements of Hirzinger (in Germany) and Fukuda (in Japan), who have carefully developed ways to use computational intelligence and learning to achieve better results in a wide variety of practical tasks. Fukuda's work on free-swinging, ballistic kinds of robots (motivated by the needs of the construction industry) mirrors the needs of space robotics far better than the domain-specific Japanese walking robots that have become so famous. Hirzinger has found ways to map major tasks in robotics into tasks for general-purpose learning systems [4], where ADP (and related advanced designs) may be used to achieve capabilities even greater than what Hirzinger himself has achieved so far.

It is also possible that new research, in the spirit of Warren Powell's chapter 10 in this book, could be critical to the *larger* organizational problems in moving thousands of robots and materials around a construction site in space. Powell's methods for approximating a value function have been crucial to his success so far in handling dynamic optimization problems with thousands of variables — but even more powerful function approximators might turn out to be important to SSP, if coupling effects and nonlinearities should make the SSP planning problems more difficult than the usual logistics problems on earth. We do not yet know.

In 2002, JIETSSP funded twelve or thirteen research projects (depending on how one counts). Five of these were on space robotics. These included an award to Singh,

Whittaker and others at Carnegie-Mellon, and other projects of a more immediate nature. They also included two more forwards-looking projects — one led by Shen at USC and one by Jennie Si — aiming at longer-term, more fundamental improvements in what can be done with robots. The Si project emphasizes ADP and what we can learn from biology and animal behavior. The Shen project emphasizes robots which can adapt their *physical form*, in the spirit of Transformers, in order to handle a wide range of tasks.

A deep challenge to NASA and NSF here is the need to adapt to changes in *high-level* design concepts for SSP. "What are we trying to build?" ask the roboticists. But we don't yet know. Will it be something like the Sun Tower design which Mankins' groups developed a few years ago? Or will the core element be a gigantic light-to-light laser, made up of pieces "floating" in a vacuum, held together by some kind of tensegrity principle with a bit of active feedback control? Could there even be some kind of safe nuclear component, such as a laser fusion system in space, using the new kinds of fuel pellets designed by Perkins of Lawrence Livermore which allow light-weight MHD energy extraction from the protons emerging from D-D fusion? One should not even underestimate ideas like the Ignatiev/Criswell schemes for using robotic systems to exploit materials on the moon. Given this huge spectrum of possibilities, we clearly need approaches to robotic assembly which are as flexible and adaptive as possible.

*1.2.2.3 **Benefits to Technology for a Sustainable Environment (TSE)*** Complicated as they are, the electric power grid and the options for SSP are only two examples taken from a much larger set of energy/environment technologies of interest to the Engineering Directorate of NSF and to its partners, (e.g., see [5]).

Many of the other key opportunities for ADP could now fit into the recently enlarged scope of Technology for a Sustainable Environment (TSE), a joint initiative of EPA and NSF. Recent progress has been especially encouraging for the use of intelligent control in cars, buildings and boilers.

Building energy use is described at length in the chapter by Charles Anderson et al., based on a grant funded jointly by CNCI and by the CMS Division of NSF. Expert reviewers from that industry have verified that this new technology really does provide a unique opportunity to reduce energy use in buildings by a significant factor. Furthermore, use of ADP should make it possible to learn a *price-responsive* control policy, which would be very useful to the grid as a whole (and to saving money for the customer). There are important possibilities for improved performance through further research — but the biggest challenge for now is to transfer the success already achieved in the laboratory to the global consumer market. There are no insuperable barriers here, but it always takes a lot of time and energy to coordinate the many aspects of this kind of fundamental transition in technology.

ADP for cars has larger potential, but the issues with cars are more complicated. There are urgent needs for reduced air pollution and improved fuel flexibility and efficiency in conventional cars and trucks. Many experts believe that the emission of NOx compounds by cars and trucks is the main accessible cause of damage to

human health and to nature today, from Los Angeles to the Black Forrest of Germany. Improved fuel flexibility could be critical to the ability of advanced economies to withstand sharp reductions in oil supply from the Middle East ten years in the future — an issue of growing concern. In the longer term, cost and efficiency concerns will be critical to the rate at which new kinds of cars — hybrid-electric, pure electric or fuel-cell cars — can actually penetrate the mass market.

Up to now, Feldkamp's group at Ford Research has demonstrated the biggest proven opportunity for learning systems to yield important benefits here. By 1998, they had demonstrated that a neural network learning system could meet three new stringent requirements of the latest Clean Air Act in an affordable way, far better than any other approach ever proven out: (1) on-board diagnostics for misfires, using time-lagged recurrent networks (TLRN); (2) idle speed control; (3) control of fuel/air ratios. In September of 1998, the President of Ford committed himself (in a major interview in *Business Week*) to deploying this system in every Ford car made in the world by 2001. An important element of this plan was a new neural network chip design from Mosaix LLC of California (led by Raoul Tawel of the Jet Propulsion Laboratory), funded by an SBIR grant from NSF, which would cost on the order of $1–$10 per chip. Ken Marko — who until recently ran a Division at Ford Research which collaborated closely with Feldkamp's Division — was asked to organize a major conference on clean air technology for the entire industry, based in part on this success. But there are many caveats here.

First, there have been changes in the Clean Air Act in the past few years, and major changes in Ford management.

Second, management commitment by itself is not sufficient to change the basic technology across a large world-wide enterprise, in the throes of other difficult changes. The deployment of the lead element of the clean air system — the misfire detector — has moved ahead very quickly, in actuality. Danil Prokhorov — who has an important chapter in this book — has played an important role in that activity, under Lee Feldkamp.

Third, the neural network control system was not based on ADP, and was not trained to reduce NOx as such. Statistics on NOx reduction were not available, at last check. The control system and the misfire detectors were *both* based on TLRNs trained by backpropagation through time (BPTT [7]). In effect, they used the kind of control design discussed in Section 2.10 of my 1974 Ph.D. thesis on backpropagation [6], which was later implemented by four different groups by 1988 [2] including Widrow's famous truck-backer-upper, and was later re-interpreted as "neural model predictive control" ([3, ch. 13], [8]). Some re-inventions of the method have described it as a kind of "direct policy reinforcement learning," but it is not a form of ADP. It does not use the time-forwards kind of learning system required of a plausible model of brain-like intelligence. Some control engineers would call it a kind of receding horizon method. Feldkamp's group has published numerous papers on the key ideas which made their success possible, summarized in part in [4].

In his chapter 15, Prokhorov raises a number of questions about ADP versus BPTT which are very important in practical engineering. Properly implemented (as at Ford),

BPTT tends to be far superior in practice to other methods commonly used. Stability is guaranteed under conditions much broader than what is required for stability with ordinary adaptive control [8, 9]. Whenever BPTT and ADP can both be applied to a task in control or optimization, a good evaluation study should use both on a regular basis, to provide a kind of cross-check. Powell's chapter gives one good example of a situation where ADP works far better than a deterministic optimization. (Unless one combines BPTT with special tricks developed by Jacobsen and Mayne [10], BPTT is a deterministic optimization method.) There are many connections between BPTT and ADP which turn out to be important in advanced research, as Prokhorov and I have discussed. Many of the techniques used by Ford with BPTT are essential to larger-scale applications of ADP as well.

More recently, Jagannathan Sarangapani of the University of Missouri-Rolla has demonstrated a 50 percent reduction in NOx (with a clear possibility for getting it lower) in tests of a simple neural network controller on a Ricardo Hydra research engine with cylinder geometry identical to that of the Ford Zetec engine [11], under a grant from CNCI. Sarangapani has drawn heavily on his domain knowledge from years of working at Caterpillar, before coming to the university. While BPTT and ADP both provide a way to maximize utility or minimize error *over the entire future*, Sarangapani has used a simpler design which tries to minimize error *one time step into the future*, using only one active "control knob" (fuel intake). But by using a good measure of error, and using *real-time* learning to update his neural networks, he was still able to achieve impressive initial results.

Saragapani explains the basic idea as follows. People have known for many years that better fuel efficiency and lower NOx could be achieved, if engines could be run in a "lean regime" (fuel air ratios about 0.7) and if exhaust gasses could be recycled into the engine at a higher level. But in the past, when people did this, it resulted in a problem called "cyclic dispersion" — a kind of instability leading to misfires and other problems. Physics-based approaches to modeling this problem and preventing the instabilities have not been successful, because of real-world variations and fluctuations in engine behavior. By using a simple neural network controller, inspired by ideas from his Ph.D. thesis adviser Frank Lewis [12], he was able to overcome this problem. This approach may lead to direct improvements in air quality from large vehicles which do not use catalytic converters (and are not likely to soon, for many reasons).

As with the Anderson work, it will be important to take full advantage of the initial breakthrough in performance here. But one can do better. One may expect better performance and extensions to the case of cars by incorporating greater foresight — by minimizing the same error or cost measure Sarangapani is already using, but minimizing it over the long-term, and multiplying it by a term to reflect the impact of fuel-oxygen ratios on the performance of the catalytic converter (or using a more direct measure or predictor of emissions); ADP provides methods necessary to that extension. ADP should also make it possible to account for additional "control knobs" in the system, such as spark plug advance and such, which will be very important to wringing out optimal performance from this system. It may also be

important to use the approaches Ford has used to train TLRNs to *predict* the system, in order to obtain additional *inputs* to the control systems, in order to really maximize performance and adaptability of the system. (See 1.3.2.2.) It may even be useful to use something like the Ford system to *predict* the likelihood of misfire, to construct an error measure still better than what Saragapani is now using. In the end, theory tells us to expect improved performance if we take such measures, but we will not know how large such improvements can actually be until we try our best to achieve them.

ADP may actually be more important to *longer-term* needs in the world of cars and trucks.

For example, if mundane issues of mileage and pollution are enough to justify putting learning chips into all the cars and trucks in the world, then we can use those chips for additional purposes, without adding too much to the cost of a car. Years ago, the first Bush Administration made serious efforts to try to encourage *dual-fired* cars, able to flip between gasoline and methanol automatically. Roberta Nichols (now with University of California Riverside) once represented Ford in championing this idea at a national level. But at that time, it would have cost about $300 per car to add this flexibility, and oil prices seemed increasingly stable. In 2003, it seems increasingly clear that more "fuel insurance" would be a very good thing to have, and *also* an easier thing to accomplish. Materials are available for multi-fuel gas tanks. It does not have to be just gasoline *or* methanol; a wide spectrum can be accommodated. But more adaptive, efficient engine control becomes crucial. Learning-based control may provide a way to make that affordable. Fuel flexibility may also be important in solving the "chicken and egg" problems on the way to a more sustainable mix of energy sources.

Cost and efficiency of control are central issues as well in hybrid, electric and fuel cell vehicles. As an example, chemical engineers involved in fuel cells have often argued that today's fuel cell systems are far less efficient than they could be, if only they took better advantage of synergies like heat being generated in one place and needed in another, with the right timing. Optimization *across time* should be able to capture those kinds of synergies.

1.2.2.4 Benefits in Other Application Domains For reasons of space, I cannot do real justice to the many other important applications of ADP in other areas. I will try to say a little, but apologize to those who have done important work I will not get to.

Certainly the broader funding for ADP, beyond ECS, would benefit from greater use of ADP in other well-funded areas such as counterterrorism, infrastructure protection above and beyond electric power, health care, transportation, other branches of engineering and software in general. Sometimes strategic partnerships with domain experts and a focus on benchmark challenges in those other sectors are the best way to get started. When ECS funding appears critical to opening the door to large funding from these other sources, many of us would bend over backwards to try to help, within the constraints of the funding process. In some areas — like

homeland security and healthcare, in particular — the challenges to humanity are so large and complex that the funding opportunities available today may not always be a good predictor of what may become available tomorrow, as new strategies evolve to address the underlying problems more effectively.

Ferrari and Stengel in this book provide solid evidence for an astounding conclusion — namely, that significant improvements in fuel efficiency and maneuverability are possible even for conventional aircraft and conventional maneuvers, using ADP. Stengel's long track record in aerospace control and optimal control provides additional evidence that we should take these results very seriously.

ADP clearly has large potential in "reconfigurable flight control (RFC)," the effort to control aircraft so as to minimize the probability of losing the aircraft after damage so severe that an absolute guarantee of survival is not possible. The large current efforts in RFC across many agencies (most notably NASA Ames) can be traced back to successful simulation studies using ADP at McDonnell-Douglas, using McDonnell's in-house model of its F-15 [3]. Mark Motter of NASA organized a special session at the American Control Conference (ACC01 Proceedings) giving current status and possibilities. Lendaris' work in this book was partly funded by NSF, and partly funded under Motter's effort, in association with Accurate Automation Corporation of Tennessee. Jim Neidhoefer and Krishnakumar developed important ADP applications in the past, and have joined the NASA Ames effort, where Charles Jorgensen and Arthur Soloway made important earlier contributions, particularly in developing verification and validation procedures.

It is straightforward to extend ADP for use in strategic games. Some of the hybrid control designs described by Shastry and others are general-purpose ADP designs, with clear applicability to strategic games, autonomous vehicles and the like. Balakrishnan has demonstrated success in hit-to-kill simulations (on benchmark challenges provided by the Ballistic Missile Defense Organization) far beyond that of other competitors. As with electric power, however, upgraded *components* may be critical in making the transition from controlling one piece of the system, to optimal management of the larger theater.

There is also a strong parallel between the goal of "value-based management" of communication networks and the goal of DSOPF discussed in Section 1.2.2.1. In particular, preventing "traffic jams" is essentially a dynamic problem, which static optimization and static pricing schemes cannot address as well.

Helicopter control and semiconductor manufacturing with ADP have been addressed by Jennie Si in this book, and by David White in [3] and in other places. Unmet opportunities probably exist to substantially upgrade the manufacturing of high-quality carbon-carbon composite parts, based on the methods proven out by White and Sofge when they were at McDonnell-Douglas [3].

As this book was going to press, Nilesh Kulkarni of Princeton and Minh Phan of Dartmouth reported new simulations of the use of ADHDP in design of plasma hypersonic vehicles, a radically new approach to lower-cost reusable space transportation, where it is crucial to have a more adaptive nonlinear sort of control scheme like ADP. (This was jointly funded by CNCI and by the Air Force.) During the meeting at

the Wright-Patterson Air Force Base, the key aerospace designers and funders were especially enthusiastic about the possible application of ADP to "Design for Optimal Dynamic Performance," which we had asked Princeton to investigate. In DODP, the simulator would contain truly vicious noise (reflecting parameter uncertainties), and ADP would be used to tune both the weights *and* the physical vehicle design parameters so as to optimize performance over time. Researchers such as Frank Lewis and Balakrishnan have demonstrated promising systems and opportunities with MicroElectroMechanical Systems (MEMS), a key stream of nanotechnology. Many forward-looking funding discussions stress how we are moving toward a new kind of global cyberinfrastructure, connecting in a more adaptive way from vast networks of sensors to vast networks of action and decision, with more and more fear that we cannot rely on Moore's Law alone to keep improving computing throughput per dollar beyond 2015. ADP may provide a kind of unifying framework for upgrading this new cyberinfrastructure, especially if we develop ADP algorithms suitable for implementation on parallel distributed hardware (such as cellular neural networks)with analog aspects (as in the work of Chris Diorio, funded by ECS).

Many researchers such as John Moody have begun to apply ADP or related methods in financial decision-making. Older uses of neural networks in finance have taken a "behaviorist" or "technical" approach, in which the financial markets are predicted as if they were stochastic weather systems without any kind of internal intelligence. George Soros, among others, has argued very persuasively and very concretely how limited and dangerous that approach can become (even though many players are said to be making a lot of money in proprietary systems). The financial system itself may be analyzed as a kind of value-calculation system or critic network. More concretely, the systems giving supply and demand (at a given price) may be compared to the Action networks or policy systems in an ADP design, and systems which adapt prices to reflect supply and demand and other variables may be compared to certain types of critic systems (mainly DHP). The usual pricing systems assumed in microeconomic optimality theorems tend to require perfect foresight, but ADP critics remain valid in the stochastic case; thus they might have some value in areas like auction system development, stability analysis, or other areas of economic analysis.

Again, of course, the work by Warren Powell on large-scale logistics problems is a major watershed in showing how ADP can already outperform other methods, and is doing so today on large important real-world management problems.

1.3 UNIFYING MATHEMATICAL PRINCIPLES AND ROADMAP OF THE FIELD

1.3.1 Definition of ADP, Notation and Schools of Thought

The key premise of this book is that many different schools of thought, in different disciplines, using different notation have made great progress in addressing the same underlying mathematical design challenge. Each discipline tends to be fiercely

attached to its own notation, terminology and personalities, but here I will have to choose a notation "in the middle" in order to discuss how the various strands of research fit each other. I will try to give a relatively comprehensive review of the main ideas, but I hope that the important work which I do not know about will be discussed in the more specialized reviews in other chapters of this book.

Before I define ADP, I will first define a focused concept of reinforcement learning, as a starting point. The founders of artificial intelligence (AI)[13] once proposed that we build artificial brains based on *reinforcement learning* as defined by Figure 1.1:

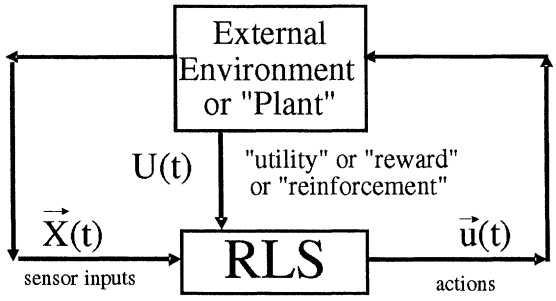

Fig. 1.1 Reinforcement Learning in AI. Reinforcement Learning Systems (RLS) choose actions u so as to maximize U, informed by raw sensor data X.

(See Figure 3.1 of [2] for a more intuitive cartoon version.). At each time t, a Reinforcement Learning System (RLS) outputs a list of numbers $u_1(1), u_2(2) \ldots u_n(t)$, which form a vector $\vec{u}(t)$. These numbers form the "action vector" or "decisions" or "controls" which the RLS uses in order to influence its environment. The RLS is supposed to operate in a digital kind of fashion, from time tick to time tick; thus the time t is treated as an integer. I would define an RLS as a certain type of system which is designed to learn how to output actions, $\vec{u}(t)$, so as to maximize the expected value of the sum of future utility over all future time periods:

$$\text{MAXIMIZE } \left\langle \sum_{k=0}^{\infty} \left(\frac{1}{1+r}\right)^k U(t+k) \right\rangle. \tag{1.1}$$

Already some questions of notation arise here.

Here I am proposing that we should use the letter "U" for utility, because the goal of these systems is to maximize the expectation value of "cardinal utility," a concept rigorously developed by Von Neumann and Morgenstern [14] long before the less rigorous popular versions appeared elsewhere. The concept of cardinal utility is central to the field of decision analysis [15] (or "risk assessment") which still has a great deal to contribute to this area. Other authors have often used the letter "r" for "reinforcement" taken from animal learning theory. Animal learning theory also has a great deal to contribute here, but the concepts of "reinforcement

learning" used in animal learning theory have been far broader and fuzzier than the specific definition here. The goal here is to focus on a mathematical problem, and mathematical specificity is crucial to that focus.

Likewise, I am using the letter "r" for the usual discount rate or interest rate as defined by economists for the past many centuries. It is often convenient, in pure mathematical work, to define a discount *factor* γ by:

$$\gamma = \frac{1}{1+r}. \tag{1.2}$$

However, the literature of economics and operations research has a lot to say about the meaning and use of r which is directly relevant to the use of ADP in engineering as well [16]. Some computer science papers discuss the choice of γ as if it were purely a matter of computational convenience — even in cases where the choice of γ makes a huge difference to deciding what problem is actually being solved. Values of γ much less than one can be a disaster for systems intended to survive for more than two or three time ticks.

From the viewpoint of economics, the control or optimization problems which we try to solve in engineering are usually just subproblems of a larger optimization problem — the problem of maximizing profits or maximizing value-added for the company which pays for the work. To do justice to the customer, one should use an interest rate r which is only a few percent *per year*. True, ethical foresight demands that we try to set $r = 0$, in some sense, for the utility function U which we use at the highest level of decision-making [16]. It is often more difficult to solve an optimization problem for the case where $r \to 0$; however, our ability to handle that case is just as important as our ability to scale up to larger plants and larger environments.

Finally, I am using angle brackets, taken from physics, to denote the expectation value.

There are many other notations for expectation value used in different disciplines; my choice here is just a matter of convenience and esthetics.

Over the years, the concept of an RLS in AI has become somewhat broader, fuzzier and not really agreed upon. I will define a more focused concept of RLS here, just for convenience.

Much of the research on RLS in AI considers the case where the time horizon is not *infinite* as in Eq. (1.1). That really is part of RLS research, and part of ADP as well, so long as people are using the kinds of designs which can address the case of an infinite time horizon. But when people address the classic problem of stochastic programming (i.e., when their future time horizon is just one period ahead), that is not ADP. There are important connections between stochastic programming and ADP, just as there are connections between nonlinear programming and ADP, but they are different mathematical tasks.

Likewise, in Figure 1.1, we are looking for an RLS design which marches forwards in time, and can be scaled up "linearly" as the complexity of the task grows. It is

tricky to define what we mean by "linear scaling" in a precise way. (See [9] for a precise statement for the particular case of Multiple-Input Multiple-Output (MIMO) linear control.) But we certainly are not looking for designs whose foresight is totally based on an explicit prediction of every time period τ between the present t and some future ultimate horizon $t+H$. Designs of that sort are basically a competitor to ADP, as discussed in Section 1.2.2.3. We are not looking for designs which include explicit solution of the N^2-by-N^2 supraoperator [17] equations which allow explicit solution of matrix Riccati equations. Nevertheless, ADP research does include research which uses less scalable components, if it uses them *within* scalable designs and if it teaches us something important to our long-term goals here (e.g., [18, 19]).

ADP is almost the same as this narrow version of RLS, *except that* we assume that utility is specified as a function $U(\vec{\mathbf{X}}(t))$ instead of just a "reinforcement signal" $U(t)$. ADP is more general, because we could always append a "reinforcement signal" to the vector $\vec{\mathbf{X}}(t)$ of current inputs, and define $U(\vec{\mathbf{X}})$ as the function which picks out that component. ADP also includes the case where time t may be continuous.

In ADP, as in conventional control theory, we are interested in the general case of *partially observed* plants or environments. In my notation, I would say that the "environment" box in Figure 1.1 contains a "state vector," $\vec{\mathbf{R}}(t)$, which is not the same as $\vec{\mathbf{X}}(t)$. (In control theory [12, 20, 21], the state vector is usually denoted as $\vec{\mathbf{x}}$, while the vector of observed or sensed data is called $\vec{\mathbf{y}}$.) "R" stands for "reality." Within an ADP system, we do not know the true value of $\vec{\mathbf{R}}(t)$, the true state of objective reality; however we may use Recurrent systems to Reconstruct or update a Representation of Reality. (See Section 1.3.2.2.)

True ADP designs all include a component (or components) which estimates "value." There are fundamental mathematical reasons why it is impossible to achieve the goals described here, without using "value functions" somehow. Some work in ADP, in areas like hybrid control theory, does not use the phrase "value function," but these kinds of functions are present under different names. Value functions are related to utility functions U, but are not exactly the same thing. In order to explain why value functions are so important, we must move on to discuss the underlying mathematics.

1.3.2 The Bellman Equation, Dynamic Programming and Control Theory

Traditionally, there is only one exact and efficient way to solve problems in optimization over time, in the general case where noise and nonlinearity are present: dynamic programming. Dynamic programming was developed in the field of operations research, by Richard Bellman, who had close ties with John Von Neumann.

The basic idea of dynamic programming is illustrated in Figure 1.2. In dynamic programming, the user supplies both a utility function, the function to be maximized, and a stochastic model of the external plant or environment. One then *solves for* the unknown function $J(\vec{\mathbf{x}}(t))$ which appears in the equation in Figure 1.2. This function may be viewed as a kind of secondary or strategic utility function. The key theorem in dynamic programming is roughly as follows: the strategy of action

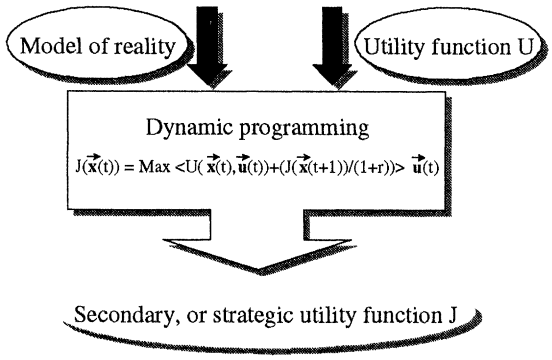

Fig. 1.2 The basic idea of dynamic programming: given U, solve the Bellman equation to get J; use J to calculate optimal actions.

$\vec{u}$ which maximizes J in the immediate future (time $t+1$) *is* the strategy of action which maximizes the sum of U over the long-term future (as in Eq. (1.2)). Dynamic programming converts a problem in optimization over time into a "simple" problem in maximizing J just one step ahead in time.

1.3.2.1 Explanation of Basic Elements in Figure 1.2 Before going further, we need to explain Figure 1.2 very carefully. There are some hidden issues here which are important even to advanced research in ADP.

The equation in Figure 1.2 is a variant of the original equation developed by Bellman. The task of optimization in the deterministic case, where there is no noise, was studied centuries before Bellman. Bellman's equation was re-interpreted, years ago, as the stochastic generalization of the Hamilton-Jacobi equation of physics. Thus many control theorists now refer to it as the "Hamilton-Jacobi-Bellman" (HJB) Equation.

Many people have found that DP or ADP can be useful even in the deterministic case. The case of zero noise is just a special case of the general method. Many books start out by explaining DP in the case of zero noise, and then discussing the general case, which they call "stochastic dynamic programming." But for researchers in the field, "stochastic dynamic programming" is a redundant expression, like "mental telepathy."

In Figure 1.2, J is the value function. Many researchers use "J" [15, 25], but many others — especially in AI — use "V," in part because of the seminal paper by Barto, Sutton and Anderson [26]. The idea is that "V" stands for "value." But there are other types of value function which occur in ADP research.

The equation in Figure 1.2 deviates a little from the original formulation of Von Neumann, insofar as the utility function U is now shown as a function of the state $\vec{x}(t)$ *and of* the actions $\vec{u}(t)$. I did not include that extension, back in the

first crude paper which proposed that we develop reinforcement learning systems by approximating Howard's version of dynamic programming [27, 22]. In that paper, I stressed the difference between primitive, unintelligent organisms — whose behavior is governed by in-born automatic stimulus-response action responses — versus intelligent organisms, which learn to choose actions based on the *results* which the actions lead to. There is no real need for the extension to $U(\vec{\mathbf{x}}, \vec{\mathbf{u}})$, since we can always account for variables like costs in the state vector. Nevertheless, the extension turns out to be convenient for everyday operations research. When we make this extension, the optimal $\vec{\mathbf{u}}(t)$ is no longer the $\vec{\mathbf{u}}(t)$ which maximizes $<J(\vec{\mathbf{x}}(t+1))>$; rather, it is the $\vec{\mathbf{u}}$ which maximizes $<U(t) + J(t+1)>$, as shown in Figure 1.2.

The reader who is new to this area may ask where the stochastic model fits into this equation. The stochastic model of reality is necessary in order to compute the expectation values in the equation.

1.3.2.2 Partial Observability and Robust Control: Why "$\vec{\mathbf{x}}$" in Figure 1.2 Is a Central Problem Note that I used "$\vec{\mathbf{x}}$" here to describe the state of the plant or environment. I deliberately did not write "$\vec{\mathbf{X}}$" or "$\vec{\mathbf{R}}$," in Figure 1.2, because there is a very serious problem here, of central importance both in theory and in practice, in DP and in ADP. The problem is that classical DP assumes a *fully observed system*. In other words, it assumes that the system which chooses $\vec{\mathbf{u}}(t)$ is able to make its decision based on *full knowledge* of $\vec{\mathbf{x}}(t)$. More precisely, it assumes that we are trying to solve for an optimal strategy or policy of action, which may be written as a function $\vec{\mathbf{u}}(\vec{\mathbf{x}})$, and it assumes that the observed variables obey a Markhov process; in other words, $\vec{\mathbf{x}}(t+1)$ is governed by a probability distribution which depends on $\vec{\mathbf{x}}(t)$ and $\vec{\mathbf{u}}(t)$, and is otherwise independent of anything which happened before time t. Failure to fully appreciate this problem and its implications has been a major obstacle to progress in some parts of ADP research.

In control theory and in operations research, there has long been active research into Partially Observed Markhov Decision Problems (POMDP). ADP, as defined in this book, is really an extension of the general case, POMDP, and not of DP in the narrowest sense.

Control theory has developed many insights into partial observability. I cannot summarize all of them here, but I can summarize a few highlights.

There are three very large branches of modern control theory: (1) robust control, (2) adaptive control and (3) optimal control. There are also some important emerging branches, like discrete event dynamical systems (DEDS) and hybrid control, also represented in this book. Even within control theory, there are big communication gaps between different schools of thought and different streams of practical applications.

Optimal control over time in the *linear case*, for a known plant subject to Gaussian noise, is a solved problem. There are several standard textbooks [20, 28]. For this kind of plant, an optimal controller can be designed by hooking together two components: (1) a Kalman filter, which estimates the state vector $\vec{\mathbf{R}}(t)$, using a kind of recurrent update rule mainly based on the past $\vec{\mathbf{R}}(t-1)$, the current observations $\vec{\mathbf{X}}(t)$ and the

past actions $\vec{u}(t-1)$; (2) a "certainty equivalence" controller, which is essentially the same as what DP would ask for, *assuming that* the estimated state vector is the true state vector. This kind of "certainty equivalence" approach is not exactly correct in the nonlinear case, but it can be a useful approximation, far better than assuming that the state vector is just the same as the vector $\vec{X}$!

Robust and adaptive control are two different ways of responding to the question: "What can we do when the dynamics of the plant themselves are unknown?" Robust control and adaptive control were both developed around the goal of *stability*, first, rather than performance. In both cases, you usually assume that you have a mathematical model of the plant to be controlled, but there are parameters in that model whose values are unknown. In robust control, you try to find a control law or controller design $\vec{u}(\vec{X})$ which guarantees that the plant will not blow up, no matter what the unknown parameters are, over a very wide space of possibilities. There are computational tools now widely available for "mu synthesis" and such, which allow you to maximize the margin of safety (the acceptable range of parameters) for the usual linear case. In adaptive control, you try to *adapt* your control rule in real time, based on real-time observations of how the plant actually behaves. Note that adaptive dynamic programming (ADP) is not a special case of adaptive control in the usual sense!

Several advanced researchers in robust control, such as Athans and Barras, have addressed the task of robust control *in the general nonlinear case*. It turns out that the solution to that task reduces to the task of trying to solve the HJB equation! Thus the most accurate possible approximation to true nonlinear robust control, in the general case, can only be found by developing software which approximates the solution to the HJB equation. From the viewpoint of nonlinear robust control, this entire book can be seen as a book about the computational issues in implementing nonlinear robust control. But in actuality, when we use an HJB equation to derive a robust controller, we can add terms in the utility function to represent concepts like energy use and cost, to represent more accurately what we really want.

Barras [29] has done important fundamental work on partial observability in this case. Crudely, his work appears to say that we can solve for the true "optimal" nonlinear robust controller, if we take the certainty equivalence approach, but we use a different kind of filter instead of the Kalman filter. We must use a nonlinear function approximator, and we must tune or train it so as to minimize a special loss function which properly penalizes different kinds of prediction errors. The reader is urged to study the original paper for a more precise statement of the findings. (See [9, 20] and the chapter by Anderson et al. in this book for some other connections between robust control, optimal control, adaptive control and ADP.)

Both in nonlinear robust control *and* in other ADP applications, there will always be a certain amount of approximation error in solving the HJB equations, no matter how hard we work to minimize that error. This does have some implications for stability analysis, for real-world nonlinear robust control just as much as for other forms of ADP. As Balakrishnan has frequently argued, we know that the controller we get *after the fact* is stable, if we have found a way to converge to a reasonably

accurate solution of the HJB equation in off-line learning, even if we had to try out different sets of initial weights to get there, and exploit tricks like "shaping" ([3], Foreword).

The problem of partial observability is crucial to practical applications, in many ways. For example, Section 1.2.2.3 mentions the work by Ford Research in developing a clean car chip. Both Ford and Sarangapani have stressed how variations between one car engine and another explain why classical types of control design (implemented at great detail and at great cost) failed to achieve the kind of success they were able to achieve. *No* fixed feedforward controller, conventional or neural, can be expected to perform well across such a wide range of plants. However, there are two approaches which have worked here: (1) as in Sarangapani, the approach of learning or adapting in real time to the individual plant; (2) as at Ford, the use of a kind of certainty equivalence approach.

In the simplest neural certainty equivalence approach (or "neural observer" approach, as described by Frank Lewis), we train a Time-Lagged Recurrent Network [3, 6] to predict the observed variables $\vec{\mathbf{X}}(t)$ as a function of $\vec{\mathbf{X}}(t-1)$, $\vec{\mathbf{R}}(t-1)$ and $\vec{\mathbf{u}}(t-1)$, where R now represents a set of recurrent neurons. In fact, the combination of $\vec{\mathbf{X}}(t)$ and of this $\vec{\mathbf{R}}(t)$ is basically just an estimated state vector! When we make this expanded vector of inputs available to our policy, $\vec{\mathbf{u}}(\vec{\mathbf{X}}, \vec{\mathbf{R}})$, we arrive at a reasonable approximation to the exact results of Stengel, Barras and others. The recurrent tunable prediction structure does not have to be a neural network, but for the general nonlinear case, it should be some kind of universal nonlinear approximator.

Eventually, brain-like capability will require a *combination* of real-time learning (to learn the laws of nature, in effect) and time-lagged recurrence (to allow fast adaptation to changes in familiar but unobserved parameters like friction and how slippery a road is [4]). It will also require moving ahead from deterministic forecasting components, to more general stochastic components, unifying designs such as adaptive hidden Markhov models [30], Stochastic Encoder-Decoder-Predictors ([3, ch. 13]), and probability distribution estimators like Kohonen's self-organizing maps [31], etc. The development of better stochastic adaptive system identifiers is a major area for research in and of itself.

Finally, there may exist other ways to cope with partial observability which do not require use of a neural observer. Some kind of internal time-lagged recurrence or "memory" is necessary, because without memory our system cannot infer anything about the dynamics of the plant which it is trying to control. But the recurrence could be elsewhere. The recurrence could even be *within* the control rule or policy itself. In ([3, ch. 13]), I showed how the concept of "Error Critic" could be used to construct a model of the olive-cerebellum system in the brain, in which the "policy" is a time-lagged recurrent neural network. This approach results in a system which does not learn as fast, over time, as the neural observer approach, but it allows the resulting controller to function at a much higher sampling rate. One can easily imagine a kind of "master-slave" design in which the "master" ADP system learns quickly but has a lower sampling rate (say, 4 hertz or 10 hertz), while the "slave" system runs at a higher sampling rate (say, 200 hertz) and tries to maximize a utility function defined

as something like $J^*(t+1) - J^*(t)$, where J^* is the value function computed by the "master" system.

Partial observability presents even larger challenges, in the longer term. In AI, there is a classic paper by Albus [32] which proposes a design for intelligent systems, in which there is *both* an estimated state vector *and* a "World Model" (a regularly updated map of the current state of the entire world of the organism). At first, this seems hard to understand, from an engineering point of view; after all, the estimated state vector is supposed to be a complete image of reality all by itself. But as the environment becomes more and more complex, something like this two-level system may turn out to be necessary after all. The Albus design may only be a kind of cartoon image of a truly optimizing learning system, but it may offer some real clues to the kinds of issues and patterns that we will need to address as we scale up to the level of complexity that the mammal brain can handle. Lokendra Shastri of Berkeley has addressed related issues as well, using ADP methods as part of a reasoning system.

In many engineering applications today, the easiest way to overcome the hurdles involving stability proof and verification is to combine off-line ADP learning with training sets based on very nasty "meta" simulation models like those used in the Ford work. It is essential that the ADP components contain the recurrence necessary to allow good performance for a fixed feedback controller over such nasty training sets. This kind of approach can be honestly described as an application of nonlinear robust control. It is necessary to achieve and verify convergence after the fact, but it is not necessary to provide an absolute guarantee that the offline learning would always converge without human assistance on all possible plants! In the case of true real-time adaptation, the adaptive control community has shown [21] that stability can be guaranteed only under very stringent conditions, even for multi-input multi-output (MIMO) linear systems. The usual methods for training critic or value-approximation networks do not possess guaranteed robust stability even in the linear case [9, sec. 7]. One may question whether even the brains of humans are universally stable under all possible streams of input experience. Nevertheless, I have proposed a suite of new ways to train value functions which do overcome these problems in the linear stochastic case [[9], section 9]. I conjecture that these methods provide the missing part of the long-sought design for true distributed MIMO adaptive control possessing total system stability for all controllable linear systems [9]. Proving such a theorem is one of the many important opportunities now at hand in this area.

1.3.3 ADP: Methods and Opportunities to Beat the Curse of Dimensionality

Exact dynamic programming, as illustrated in Figure 1.2, cannot be used in the general case. There is no general method to calculate an exact closed form solution to any arbitrary nonlinear differential equation! In fact, there is no such method even for simple algebraic equations, when they get beyond fourth-order or fifth-order polynomial equations. For the general case, *numerical methods* or *approximations* are the best that can be done. Even the human brain itself cannot exactly "solve"

NP-hard problems like playing a perfect game of chess or of Go; in reality, the goal is to do as well as possible, not more.

Exact solutions of the Bellman equation are possible in two important special cases: (1) when the dynamics of the environment are purely linear, with Gaussian noise and a quadratic utility function, the optimal control policy is well known [20, 28]; (2) when the plant or environment is a *finite-state* system, the mathematics become much easier.

A finite-state system is *not* a system which has a finite number of state variables; rather, it is a system which can only exist in a finite number of possible states. If a plant can only exist in N possible states, $i = 1$ to N, we only need to solve for N numbers $J(i)$. *After* we choose a specific policy or strategy of action, $\vec{\mathbf{u}}(i)$, the stochastic model of the plant reduces to a simple Markhov chain, M_{ij}, a matrix containing the probabilities of transition from any state j at time t to state i at the next time tick. We can represent the state of such a system at any time as a simple integer i, but it turns out to be more convenient to use a more complicated representation, representing the state as a vector $\vec{\mathbf{x}}$ in $\Re^N$. If the system is in state number i, we set $x_i = 1$ and $x_j = 0$ for all the other states j. In this representation, the function $J(\vec{\mathbf{x}})$ is represented by a simple vector $\vec{\mathbf{J}}$; in other words:

$$J(\vec{x}) = \vec{J}^T \vec{x}, \tag{1.3}$$

and likewise for $U(\vec{\mathbf{x}})$, if we delete the unnecessary dependence on $\vec{\mathbf{u}}$. The Bellman equation then reduces to:

$$\vec{J}^T \vec{x}(t) = (\vec{J}^T M^* + \vec{U}^T)\vec{x}(t) \text{ for all possible } \vec{\mathbf{x}}(t), \tag{1.4}$$

where M^* is the transition matrix for the *optimal* policy. This is easily solved algebraically:

$$\vec{J} = (I - M^{*T})^{-1}\vec{U}. \tag{1.5}$$

Equation (1.5) is not a complete analytic solution of the Bellman equation in this case, because we still need to *find* the optimal $\vec{\mathbf{u}}(i)$. Sometimes this can be done analytically. But often we can use the iterative dynamic programming methods of Howard [22], which could be viewed as the first ADP designs. Howard also developed methods to deal with "crossroads" problems which can occur sometimes, when the interest rate r is zero and the time horizon is infinite. (Crossroads as in "the crossroads of history.") These crossroads problems are a real substantive issue at times, and not just a mathematical anomaly; thus I deeply regret that I did not use the language of social science correctly in describing such problems, in the first published paper [33] which defined several of the basic ADP methods in relatively complete, mathematical fashion (stressing the use of backpropagation as a tool to enable scaling up to larger problems).

Very few decision or control problems in the real world fit exactly into one of these two special cases. Even for problems of inventory management and control, where

the number of possible states N is often finite *in principle*, the number of possible states in the real world is often much too large to let us use Eq. (1.5) exactly.

Early attempts to approximate dynamic programming were mostly based on a very simple approximation to the Bellman equation. The true environment, however complex, was approximated as a finite state system. For example, if the state vector $\vec{x}$ is a bounded vector in $\Re^N$, we can represent $J(\vec{x})$ by using a kind of N-dimensional lookup table containing all possible values of $\vec{x}$. But the size of such a lookup table grows exponentially with N! This is the core of the "curse of dimensionality." Warren Powell, in his chapter, discusses the curse of dimensionality in more detail.

Strictly speaking, ADP does not throw out that old approach to approximation! It treats it as a special case of one of the **three more general principles which are the core of ADP**:

(1) **Value approximation:** Instead of solving for $J(\vec{x})$ *exactly*, we can use a universal approximation function $J(\vec{x}, W)$, containing a set of parameters W, and try to estimate parameters W which make $J(\vec{x}, W)$ a good approximation to the true function J;

(2) **Alternate starting points:** Instead of always starting from the Bellman equation directly, we can start from *related* recurrence equations (Section 1.3.3.2) which sometimes improve the approximation;

(3) **Hybrid design:** We can combine multiple ADP systems and other systems in more complex hybrid designs, without losing generality, in order to scale up to larger real-world problems, such as problems which involve multiple time scales and a mixture of continuous and discrete variables at multiple scales of space.

Dozens of different ADP designs or algorithms have been developed and used in different disciplines, based on one or more of these three general approaches. In order to build a *unified* ADP design, which can learn to handle the broadest possible variety of complex tasks (as well as the mammalian brain does), we will need to exploit all three approaches to the utmost, and learn how to combine them together more effectively.

The remainder of this section will discuss each of these three approaches, in turn. It will proceed systematically from simpler ADP designs — easier to implement but less able to cope with difficult tasks — all the way up to the larger research challenges critical to the long-term strategic goals discussed above. In so doing, it will try to provide a kind of global roadmap of this very complicated field.

1.3.3.1 Value Approximation and the Adaptive Critic This subsection will mainly discuss how we choose and tune an approximate value function $J(\vec{x}, W)$ based on the Bellman equation proper. But the reader should be warned that the simplest reinforcement learning packages based on this approach often converge slowly when they are used on engineering tasks with a moderate number of continuous

variables in them. Many critics of reinforcement learning have claimed that reinforcement learning could not be useful on large-scale problems — mainly because their students, using the easiest off-the-shelf tools in MatLab, were unable to tackle a large problem at first pass. The simplest designs are an important place to start, but one must build up beyond them (as in Sections 1.3.2.2, 1.3.3.2 and even 1.3.3.3) in order to handle larger problems. And of course we need further research and better software in order to capture the full power of the approach.

Even if we limit ourselves to ADP methods which tune $J(\vec{\mathbf{x}}, W)$ based on the Bellman equation proper, there is still a huge variety of more-or-less general designs. There are four decisions you must make when you put together such a system:

(1) What function $J(\vec{\mathbf{x}}, W)$ do you choose to approximate $J(\vec{\mathbf{x}})$?

(2) **Value updates:** For any *given* strategy or policy or controller $\vec{\mathbf{u}}(\vec{\mathbf{x}})$, how do you adapt the weights W to "best fit the Bellman equation"?

(3) **Policy updates:** How do you represent and adapt $\vec{\mathbf{u}}(\vec{\mathbf{x}})$ itself as part of this process?

(4) How do you coordinate, manage and time the value updates and policy updates?

With regard to question 1, many researchers still use lookup tables or linear basis function approximators:

$$\hat{J}(\vec{x}, W) = \sum_{\alpha} W_{\alpha} f_{\alpha}(\vec{x}), \tag{1.6}$$

where the functions f_{α} are preprogrammed basis functions or features chosen by the user. A truly general-purpose computer program to calculate value updates and policy updates should be able to accommodate these particular choices as a special case. These special cases can also yield mathematical insights [25] which might be useful as a stepping-stone to where we want to go. But our goal here is to get to the *general* case, where we need to be able to use a universal approximator able to approximate "any" nonlinear function. Powell, in particular, has stressed that his success in handling large problems is based on going beyond what he could get with linear basis functions as in [25].

But which universal approximator $J(\vec{\mathbf{x}}, W)$ do we use? The choice of function approximator has been one of the key issues explaining why some researchers have done much better than others in real-world applications of ADP. We will also need to use approximators much more powerful than anyone has used in the past, in order to scale up to more brain-like performance.

Universal approximation theorems have been proven for many, many systems such as Taylor series, fuzzy logic systems, wavelet systems, simple "global" neural networks (like the Multilayer Perceptron, MLP [7, 34]), and "local" neural networks like the Radial Basis Function (RBF), the CMAC, the SOM, the ART, etc. These all have advantages in various application domains. Some people from statistics argue

that there can never be any real science here, because "there is no free lunch." They argue that some approximators will be better in some domains, and others will be better in others, and there never can be a truly general-purpose system.

Certainly it makes sense for software designers to give their users a wide choice of options, but the "no free lunch" theory misses a few critical points. It misses a very large historical literature on the foundations of induction and learning, which is relevant here but is too complicated to discuss in detail (e.g., see ([4, ch. 10]) and sources cited in [34].) It misses the fact that approximators in family A can be dense in family B but not vice-versa. More concretely, it does not fully account for the important, practical theorems proven by researchers like Barron [35] and Sontag on function approximation.

Taylor series, lookup tables, gain schedulers, CMAC and RBF are all examples of the broad class of linear basis function approximators. Barron [35] proved that linear basis function systems and MLPs are both universal approximators, *but with different accuracy in approximation.* When we try to approximate smooth functions, and maintain a certain level of accuracy with more and more inputs (m), the *number of parameters* or nodes needed to maintain that accuracy rises at an exponential rate with m, for linear basis function approximators, while it only rises as a gentle polynomial function of m for MLPs. This makes intuitive sense, because we know that the required size of a lookup table grows exponentially with m, and we know that other linear basis function approximators are based on the same underlying principle.

Barron's result is only the tip of a very large iceberg, but it already conveys an important lesson. In practical work in the neural network field, it is well known that MLPs allow greater accuracy in approximation for *moderately large* m than the local approximators. (There are some exceptions, when the data is all clustered around a few points in state space, but that is not the general case.) A majority of the practical successes discussed in Section 1.2 were based on using MLPs as the value approximator(s). On the other hand, it usually takes more time and skill to train MLPs than to train linear basis function approximators. Thus one has to work harder if one wants to get better results; this is a recurrent theme across all parts of the ADP field. There is also still a need for more research on ways to blend global networks like MLPs and local networks together, to combine the advantages of both, *particularly* when the speed of learning is an urgent issue; this is one of the subsystem-level issues I referred to in Section 1.2.1.

I have speculated [36] that Elastic Fuzzy Logic systems (ELF) might be as powerful as MLPs in function approximation, but for now this is just a speculation, so far as I know.

MLPs can do well in approximating smooth functions with m on the order of $50 - 100$. That is a huge step up from the old lookup-table dynamic programming. *But it is still a far cry from a brain-like scale of performance.* It is good enough when the job is to control a single car engine, or airplane, or turbogenerator or router. It is *not good enough* to meet the needs of large-scale network control as discussed in Section 1.2.

Many people from AI and statistics have argued that we will never be able to scale up to brain-like complexity, without exploiting a whole lot of prior information. This is true, in principle. But researchers going back to the philosopher Emmanuel Kant and the statistician Brad Efron have stressed that we can use a particular *type* of prior information which does not block our ability to learn new things in an open-minded way. From a practical viewpoint, we can exploit tacit prior assumptions about symmetry in order to design whole new classes of neural network designs, which are capable of addressing large-scale network management, image processing and segmentation and the like. There is a ladder of designs which now exists here, rising up from the generalized MLP [7, 34], to the Simultaneous Recurrent Network (SRN), to the Cellular SRN [37, 38], to the ObjectNet [39]. These should not be confused with the simple recurrent networks which are widely discussed in psychology but are not so useful here.

A few quick comments about strategic games may be of interest to some readers. Fogel [18] has argued that the past success of reinforcement learning in playing games has been oversold. Reinforcement learning has generated human-class performance in backgammon and checkers, *but only* when the programmers spent enormous effort in crafting the features they use in a linear basis function value function. There was never any true learning of the game from scratch. With tic-tac-toe, he was able to train a perfect game player simply by using genetic algorithms to evolve an optimal game *player* ($\vec{u}(\vec{x})$). But this did not really work for checkers, as he had hoped. He *was* able to achieve human-level performance for the first time in checkers without cheating, by evolving a value-approximating MLP network $J(\vec{x}, W)$, and using that network as a basis for choosing moves. This was an historic achievement. But no one has ever achieved real human-level performance in the game of Go, with or without learning; based on what we learned from earlier work on lattices similar to Go boards [37], I would conjecture that: (1) Fogel's approach might work with Go, if he replaced the MLP with a cellular SRN or ObjectNet; (2) even his earlier attempt to evolve a controller directly for checkers might have succeeded if he had used an SRN; (3) success in theater-like strategy games would also require Object Nets, but research on Go would be a good starting point to get there. I do not claim that this simple approach would do full justice to the game of Go. Rather, it offers a tangible hope to show that learning systems might outperform existing Go players. One could do better, to start, by using more advanced evolutionary computing methods, such as the use of particle swarm methods to adapt the continuous valued weights in the value-approximating network. In parallel, we need to develop better gradient-exploiting methods to train recurrent recurrent systems like ObjectNets, in order to train networks too large for evolutionary computing to handle, and in order to get closer to brain-like approaches. Go can provide a testbed for even more powerful general concepts, but it would be useful to get this stream of research extended and disseminated much further immediately, based on the best we can do today.

Again, this is all the tip of an iceberg, but the other three design questions above also merit serious discussion here.

First, the issue of how to do value updates. Beginners to ADP often suggest that we should try to train the weights or parameters W by minimizing the following measure of error as a function of W:

$$E_G = \sum_t (J(\vec{x}(t), W) - (U(\vec{x}(t), \vec{u}(t)) + \gamma J(\vec{x}(t+1), W))^2, \quad (1.7)$$

where the sum is taken over real or simulated pairs of data $(\vec{\mathbf{x}}(t), \vec{\mathbf{x}}(t+1))$. (The subscript "G" refers to Galerkin.) Years ago, I evaluated the performance of this value-updating method, for the case of linear multiple-input multiple-output (MIMO) plants with nonzero noise. I proved that it converges to the wrong estimate of W almost always [9, 40].

The connection to Galerkin is discussed at length in [9]. The procedure of minimizing E_G is essentially a special case of one of the more general methods proposed by Galerkin decades ago to approximate the solution to partial differential equations (PDE). But the Bellman equation is not strictly a PDE, because of its stochastic aspect. The stochastic aspect turns out to be the source of the problem in trying to apply Galerkin's approach here [9]. More recently, Balakrishnan has adapted a different method of Galerkin (specialized to time-forwards systems rather than PDE in general) to the challenge of optimal, ADP control of systems like fluid flows governed by PDE.

In 1977 [41], I proposed instead that we approximate Howard's iterative procedure directly, by *training* the network $J(\vec{\mathbf{x}}, W)$ to try to match or predict the targets:

$$J^*(t) = U(t) + J(t+1)/(1+r). \quad (1.8)$$

This training could be done by minimizing least square error and backpropagation, or we could use *any other* method used to train any kind of function approximator to match examples of input vectors and desired output vectors. I called this method "Heuristic Dynamic Programming" (HDP) [41]. This statement of the method is quite brief, but it is complete. The method really is quite simple, in the general case. (See [3] for elaborate discussions of how to embed this idea in simple, general form within a larger software system for ADP.)

Earlier, in 1973 [42], Widrow implemented an alternative training method for a neural network which he called a "critic", for use in playing blackjack. In effect, he trained $J(\vec{\mathbf{x}}(t), W)$ directly to predict:

$$J^*(t) = \sum_{\tau=t}^{T-1} U(\tau) + U^*(T), \quad (1.9)$$

where U^* is a kind of terminal payoff at a terminal time T. There was no connection made to utility functions or dynamic programming. In 1983, Barto, Sutton and Anderson [26] generalized Widrow's method to define the class of methods $TD(\lambda)$, such that $\lambda = 0$ and $\lambda = 1$ would yield a choice between Eq. (1.8) and Eq. (1.9). There was no indication of a connection to [41] or to any form of dynamic

programming at that time; that connection was made later, in 1987, when Sutton read [43], and set up the discussions at GTE which later led to the 1988 NSF workshop reported in [2]. Since then, we have used the term "critic" or "critic network" to refer to any parameterized network used to approximate $J(\vec{\mathbf{x}})$ *or to approximate* any of the other various value functions which emerge from relatives of the Bellman equation (Section 1.3.3.2). An "adaptive critic" system is any system which contains such a critic network, plus some mechanism to adapt or tune the critic based on off-line learning or on-line learning,

Many researchers such as Williams have been able to show how pathological training sets $\{\vec{\mathbf{x}}\}$ can cause HDP itself to converge to the wrong answer or even to diverge. Van Roy [44] has proven that it will always converge to the right value, under reasonable conditions, *if* the training set is chosen from the normal stream of experience one would expect to see while controlling and observing the actual plant. There are several more complex variants of HDP, for which I have proven that the method will always converge to the right value function for linear MIMO plants (including stochastic versions), even when the training sets are chosen in a pathological manner [9]. There are also a wide variety of other methods available, some more truly learning-based and some based on linear programming, as described by Van Roy's chapter 10 in this book. It would be interesting to see whether Van Roy's approach could be made more powerful, by exploiting the nonlinear programming methods described in Momoh's chapter 22. It would also be interesting to see whether such methods could be restructured or adapted into true, scalable learning-based methods for adapting $J(\vec{\mathbf{x}}, W)$, similar in spirit to some of the work by Trafalis in training simpler types of neural networks based on sophisticated ideas from operations research [45].

Policy updates are much more straightforward than value updates, in theory. Many researchers in AI address tasks where there is only a finite set of possible actions to take at any time t. In designing a system to play Go, for example, one might search over every possible move, and do the maximization on the fly by brute force. (There are many other similar choices familiar to AI researchers.) In engineering applications today, it is reasonable to train a controller $\vec{\mathbf{u}}(\vec{\mathbf{x}}, W_u)$, which may also be a universal approximator. For example, we may train the weights $W_{u\alpha}$ so as to maximize $<U(t) + \gamma J(t+1)>$, by adapting them in response to a stochastic gradient estimate:

$$\begin{aligned} &\frac{\partial}{\partial W_{u\alpha}} \left(U(t) + \frac{J(t+1)}{1+r} \right) \\ &\approx \sum_i \left(\left(\frac{\partial U}{\partial u_i(t)} + \gamma \sum_j \frac{\partial J(t+1)}{\partial x_j(t+1)} \cdot \frac{\partial x_j(t+1)}{\partial u_i(t)} \right) \cdot \frac{\partial u_i(t)}{\partial W_{u\alpha}} \right). \end{aligned} \tag{1.10}$$

The control theorist should immediately see that this is *exactly* the same as the procedure used to update or train a controller in Indirect Adaptive Control (IAC) [46], except that IAC eliminates "U" and it replaces $J(t+1)$ with a *fixed* measure of tracking error which is basically arbitrary. Both here and in IAC, we must train or obtain a model of the plant in order to estimate the required partial derivatives. In both

cases, the derivative calculations and summation can be done more quickly and more efficiently in real time by using an algorithm which I call "backpropagation through a model." [3, 7]. This algorithm calculates the required derivatives efficiently through *any* differentiable model, neural, fuzzy, classical or other. In 1988 [2], I coined the term "Backpropagated Adaptive Critic" (BAC) [2] to name this way of training a controller or Action Net $u(\vec{x}, W_u)$. The stability guarantees for IAC depend on *highly* restrictive assumptions, even in the linear case [21]; however, there is good reason to believe that the *extension* of IAC provided by the use of new HDP variants can yield universal total system stability guarantees for *all* controllable linear plants [9].

More recently, some researchers have used the term "Policy Gradient Control Synthesis" (PGCS) for this approach to updating or training an Action Net. In certain designs, this may require the efficient real-time calculation of selected second derivatives. The form of second-order backpropagation required for such applications was first discussed in 1979 [33], but explained in more detail in [3].

High-level decision problems sometimes require more than just brute-force maximization or an Action network of this type. As we scale up to brain-like capabilities, we will need to design and train something like Option Networks, which generate reasonable *choices* for $\vec{u}(t)$ decisions, along with search and selection systems to go with them [47]. This involves some kind of stochastic search capability, related to evolutionary computing (EC), but not at all the same. In conventional EC, the user supplies a function $U(\vec{u})$, and a fixed algorithm searches through possible values of $\vec{u}$. But in brain-like stochastic search (BLiSS?), we try to develop learning systems which learn to do better and better stochastic search for *families* of optimization problems $U(\vec{u}, \vec{x})$. For example, one such function $U(\vec{u}, \vec{x})$ would be the *family* of all travelling salesman problems for 1,000 cities, where x is the vector of coordinates of those cities. There has been some preliminary research related to BLiSS by Wunsch and by Serpen, but much more will be needed.

Finally, the issue of how to manage, time and control value updates and policy updates is an extremely complex issue, very dependent on the specific choices one makes. Many chapters in this book will discuss various aspects of these issues. As a general rule, we do know that mammal brains somehow do concurrent real-time updates of *all* their components. But they also need periods of offline-learning and consolidation, in deep sleep and dreams. There are learning tasks in ADP which clearly relate to these biological functions [1, 2, 43], but more research will be needed to pin them down more completely.

1.3.3.2 Alternative Starting Points: Beyond the Bellman Equation Proper The previous section discussed value approximation for ADP designs based *directly* on the Bellman equation — HDP, TD and variations of them. The Bellman equation itself may be viewed as a kind of recursion equation for the function $J(\vec{x})$.

There are two alternative starting points which are important to applications of ADP today. One of them is the recursion equation which underlies methods called Action-Dependent HDP (ADHDP), Q-learning, and related methods. The other is

the recursion relation underlying Dual Heuristic Programming (DHP); that equation is essentially a stochastic generalization of the Pontryagin equation, which is as important and well known as the Hamilton-Jacobi equation in some fields of science.

Crudely speaking, the ADHDP/Q-recurrence leads to ADP designs which do not directly require a model of the plant or environment. (However, they do not escape the issues discussed in Section 1.3.3.2.) The DHP recurrence equation was developed in order to overcome scaling problems with all methods discussed so far in this section, for tasks involving continuous variables.

Q-learning and ADHDP are different methods, dating back to two independent sources in 1989 — the classic Ph.D. thesis of Watkins [23] and a far briefer paper given at the IEEE Conference on Decision and Control [24], respectively. But both can be derived from the same recurrence equation. Here I will derive that equation as in [24], but will now label the new value function as "Q" instead of "J."

If we write the Bellman equation as:

$$J(\vec{\mathbf{x}}(t)) = \max_{\vec{\mathbf{u}}(t)} < U(\vec{\mathbf{x}}(t), \vec{\mathbf{u}}(t)) + \gamma J(\vec{\mathbf{x}}(t+1)) >, \tag{1.11}$$

we may simply *define*:

$$Q(\vec{\mathbf{x}}(t), \vec{\mathbf{u}}(t)) = \ < U(\vec{\mathbf{x}}(t), \vec{\mathbf{u}}(t)) + \gamma J(\vec{\mathbf{x}}(t+1)) > . \tag{1.12}$$

Using this definition, Eq. (1.11) may be expressed equivalently as:

$$J(\vec{\mathbf{x}}(t)) = \max_{\vec{\mathbf{u}}(t)} Q(\vec{\mathbf{x}}(t), \vec{\mathbf{u}}(t)), \tag{1.13}$$

substituting Eq. (1.13) into Eq. (1.12), we deduce:

$$Q(\vec{\mathbf{x}}(t), \vec{\mathbf{u}}(t)) = \ < U(\vec{\mathbf{x}}(t), \vec{\mathbf{u}}(t)) + \gamma \max_{\vec{\mathbf{u}}(t+1)} Q(\vec{\mathbf{x}}(t+1), \vec{\mathbf{u}}(t+1)) > . \tag{1.14}$$

Equation (1.14) is the recurrence equation underlying both ADHDP and Q-learning.

Q-learning generally uses lookup tables to represent or approximate the function Q. In the 1990 workshop which led to [3], Watkins did report an effort to approximate Q using a standard CMAC neural network, for a broom-balancing task; however, he reported that that variation did not work.

With Q-learning, as in TD learning, one can derive the policy $\vec{\mathbf{u}}(\vec{\mathbf{x}})$ simply by considering every possible choice of actions, in tasks where there is only a manageable, finite list of choices. However, in the TD case, dynamic programming clearly calls on us to try to *predict* $< J(t+1) >$ for each choice, which then requires some kind of model of the plant or environment. With Q-learning, you only need to know the Q function. You escape the need for a model. On the other hand, if $\vec{\mathbf{x}}$ and $\vec{\mathbf{u}}$ are actually continuous variables, and if you are using a lookup table approximation to Q, then the curse of dimensionality here is even worse than with conventional DP.

For Action-Dependent HDP (ADHDP) [3, 24], I proposed that we approximate $Q(\vec{\mathbf{x}}(t), \vec{\mathbf{u}}(t))$ and $\vec{\mathbf{u}}(\vec{\mathbf{x}})$ by using universal approximators, like neural networks. We

can train a network $Q(\vec{x}, \vec{u}, W)$ to try to match targets Q^* based on the recurrence Eq. (1.14), using *exactly* the same procedure as in HDP. (See [3] for pseudo-code, flowcharts, examples and analysis.) We can train $\vec{u}(\vec{x}, W_u)$ to maximize Q, by tuning the weights of W_u in response to the derivatives of Q, backpropagated from Q to $\vec{u}$ to the weights.

Some neural network researchers would immediately ask: "Doesn't the use of backpropagation lead to slow learning here?" Not necessarily. When a traditional MLP is trained by backpropagation, the learning is indeed slower than what we see with radial basis functions (RBF), for example. But the problem lies with the MLP structure, more than with backpropagation. (It is also important to know modern techniques for adjusting learning rates and such.) More powerful function approximators (like humans!) are even trickier to train, because they are searching a space of possibilities which is even larger. But when backpropagation is used to train simpler, local networks, it can be reasonably fast.

As an example, David White — co-sponsor of the 1990 joint NSF/McDonnell-Douglas workshop which led to [3] — applied ADHDP in 1990 to control the McDonnell model of the F-15 aircraft in simulation. He and Urnes did a study in which the simulated aircraft was badly damaged in random ways, and the ADHDP system tried to relearn the control fast enough (2 seconds) to keep the craft from crashing. It succeeded about half the time, which was far better than conventional methods (2 percent survival or so). The neural networks used to approximate Q and $\vec{u}$ were a new, differentiable variant of the CMAC design. (See [3] for details.) White and Sofge also used ADHDP to produce high-quality carbon-carbon parts using a cost-effective process which had defied earlier, expensive approaches based on more traditional methods. (See [3] and Section 1.2 of this chapter.)

ADHDP in some form has been independently rediscovered by a number of researchers, who have reported good results. For example, Ford Research [48] reported a quick and easy solution of the bioreactor control challenge problem in [3], which, according to Ungar, defied the very best and most modern adaptive control methods. Many of Si's results reported in this book use a design in this category. Shibata's "fuzzy critic" for robot control [49], developed under Fukuda, also appears to be in this category. It is good news that we are beginning to integrate and consolidate more of this work.

ADHDP can handle larger-scale engineering problems than lookup-table Q or TD approaches. (From an engineering viewpoint, I have described the lookup-table and "associative trace" versions of Q and TD as "level 1," versus ADHDP as "level 2.") But it still encounters problems as one tries to scale up to even larger problems, and it also has problems related to "persistence of excitation" ([3, ch. 13]). Because of these issues, and because of the issues of partial observability, I have argued that the combination of HDP and BAC discussed in the previous section tends to be more powerful — "level 3" — on engineering problems which involve continuous variables. In order to scale up still further — "level 4" — many of us have gone on to use a different recurrence equation altogether, the DHP recurrence equation.

The correct recurrence equation for DHP may be derived in a straightforward way, by first defining:

$$\vec{\lambda}(t) = \nabla_{\vec{x}} J(\vec{x}(t)), \tag{1.15}$$

which is another way of writing:

$$\lambda_i(t) = \frac{\partial J(\vec{x}(t))}{\partial x_i(t)} \qquad (\forall i), \tag{1.16}$$

by performing the calculation for a particular policy $\vec{u}(\vec{x})$, and by differentiating Eq. (1.11) with respect to $x_i(t)$ to get:

$$\begin{aligned}\lambda_i(t) = \Bigg\langle & \frac{\partial U(\vec{x}(t),\vec{u}(t))}{\partial x_i} + \gamma \sum_j \frac{\partial J(\vec{x}(t+1))}{\partial x_j(t+1)} \cdot \frac{\partial x_j(t+1)}{\partial x_i(t)} \\ & + \sum_k \left(\frac{\partial U(\vec{x}(t),\vec{u}(t))}{\partial u_k(t)} + \gamma \sum_j \frac{\partial J(\vec{x}(t+1)}{\partial x_j(t+1)} \cdot \frac{\partial x_j(t+1)}{\partial u_k(t)} \right) \cdot \frac{\partial u_k(\vec{x}(t))}{\partial x_i(t)} \Bigg\rangle .\end{aligned} \tag{1.17}$$

This general type of complex derivative calculation relies heavily on mathematical foundations discussed further in [6].

Some control theorists have asked at first: "Can you do this? Just differentiate the Bellman equation?" In fact, we *know* that Eq. (1.11) must hold for the optimal policy. So long as J and u and the plant are differentiable (within the Kolmogorov probability formalism [10]), we know that Eq. (1.17) *must hold* as well, for an optimal policy. It is a necessary condition for an optimum. Likewise, the policy update equivalent to Eq. (1.10) is trivial to deduce. (See [3].) But Eq. (1.10) is not a *sufficient* condition for a globally optimal policy; that is why I discussed stochastic search issues after the discussion of Eq. (1.10), and similar issues apply here. For complex nonlinear problems in general it is often possible to guarantee local optima, superiority to linear methods, and global optima in various convex cases in a rigorous way, but intelligent systems design can never guarantee "perfect" creativity in finding global optima. Stronger guarantees exist for GDHP (to be discussed below) than for DHP.

The recurrence equation for $\vec{\lambda}$ follows by using the definition of $\vec{\lambda}$ to simplify Eq. (1.17):

$$\begin{aligned}\lambda_i(t) = \Bigg\langle & \frac{\partial U}{\partial x_i} + \gamma \sum_j \lambda_j(t+1) \cdot \frac{\partial x_j(t+1)}{\partial x_i} \\ & + \sum_k \left(\frac{\partial U}{\partial u_k} + \gamma \sum_j \lambda_j(t+1) \cdot \frac{\partial x_j(t+1)}{\partial u_k} \right) \cdot \frac{\partial u_k}{\partial x_i} \Bigg\rangle ,\end{aligned} \tag{1.18}$$

where variables without time arguments refer to values at time t.

The obvious way to approximate this recurrence relationship is to train an approximator, $\vec{\lambda}(\vec{x}, W)$, to meet the targets $\vec{\lambda}^*$ defined by the right-hand side of Eq. (1.18); more precisely, starting from any estimate of W, one can update the estimate of W to make $\vec{\lambda}(\vec{x}, W)$ better match $\vec{\lambda}^*(\vec{x}, W)$, where $\vec{\lambda}^*(\vec{x}, W)$ is defined by the right-hand side of Eq. (1.18), using the *current* estimates of W to estimate the values of $\vec{\lambda}(t+1)$ by $\vec{\lambda}(\vec{x}(t+1), W)$ as required to calculate to right-hand side of Eq. (1.18). Details, flow-charts, pseudo-code and some theoretical analysis (along with a couple of unfortunate typos) can be found in [3, ch. 13]. Using modular code design, and a basic understanding of backpropagation, it is not necessary to be an expert in control theory or on these equations in order to implement DHP.

Certain warnings are needed here. Although I discussed the general idea of DHP back in 1977 [41], I did not really specify the method then (unlike the case for HDP). In 1981 [50]. I discussed a way to implement the idea, using backpropagation and an error function similar to Eq. (1.7); however, that kind of variation (which I now call DHPG) converges to the wrong answer almost always for linear dynamical systems with noise [9]. In order to converge to the correct answer, it is necessary in principle to include the $(\partial u_k / \partial x_i)$ terms, which were explained for the first time in [3]. (See [9] for extensions important to robust and adaptive control.) All of the actual implementations and applications of DHP occurred after the publication of [3]. Many of the most important ones appear in this book.

The vector $\vec{\lambda}$ of DHP has many connections to concepts in other fields of science and engineering. For example, to the economist, $\lambda_i(t)$ represents a kind of "marginal utility" or "shadow price" — the long-term market value of the commodity whose quantity or level is given by the variable x_i. DHP critics could actually be used as pricing systems. In the Pontryagin equation, λ_i is called a "costate variable" in modern optimal control [20, 28], the letter "λ" is still used for costate variables. The vector $\vec{\lambda}$ is also closely related to the gradients calculated in backpropagation through time, which creates many opportunities for seamless, integrated hybrid designs, discussed in part in the chapter 15 by Prokhorov.

DHP could have some interesting applications in network control, as discussed in Section 1.2.2. For a large network system, we would need something like an ObjectNet (Section 1.3.3.1) in order to input observations across an *entire electric power grid*, and output a vector of shadow prices or values for all of those observed values. *Physically*, an ObjectNet may be thought of as a specific kind of highly coordinated assembly of *component* networks, each of which refers to a particular object in the grid; thus we could choose to implement such a value-approximation network as a distributed system, with chips located near each object to implement the component network for that object. Such a system would output the estimate of values which apply to each object, from the node which is actually located at that object. At each object, we could perform a kind of local optimization over time, using a lower-level ADP system which responds to these global values, using a master-slave kind of arrangement similar to the one discussed in Section 1.3.2.2. Communication constraints could simply be represented as part of the topology specification of the ObjectNet to be trained.

DHP is not the only method in its class. There have also been a few simulations of Action-Dependent DHP (ADDHP), Globalized DHP (GDHP), ADGDHP and related Error Critic designs. (See [3, 9] and some of the work of Wunsch and Prokhorov.) In essence, GDHP is a hybrid of HDP and DHP. It trains a scalar critic which estimates the J function, but uses second derivatives in order to achieve the same effect as DHP. In effect, GDHP provides a way of performing DHP (as described above) while also guaranteeing strict adherence to the requirement that the vectors $\vec{\lambda}$ are the gradient of a function J. More generally [43], GDHP allows one to train the critic to minimize a weighted sum of the DHP second-order error measure and the usual HDP/TD first-order error measure; if *some* state variables are continuous while others are discrete, one can use GDHP on the entire J function by simply by not using the (undefined) second-order terms for the discrete state variables. (The discussion of [43] needs to be updated, to reflect the convergence results in [3] and [9].) Benchmark studies by Wunsch and Prokhorov show little difference in performance between the easier DHP method and GDHP in difficult engineering problems involving continuous variables. Thus it may be premature to say more about GDHP at the present time.

1.3.3.3 Hybrid and Large-Scale Designs: Closing the Gap to the Mammal Brain Level This chapter started out by posing a question: when and how can we handle dynamic optimization problems as large and as complex as what the smallest mammal brain can learn to handle? (Of course, such systems could do far better than mice or humans on *some problems*, as a byproduct of this effort.)

Years ago, I believed that the kinds of designs discussed in Sections 1.3.3.1 and 1.3.3.2 might be enough to achieve that goal by themselves. After all, the famous neuropsychologist Hebb argued decades ago that all the complexities of higher-order intelligence might result as an *emergent property* of a much simpler kind of learning system [51]. Why could they not emerge from these kinds of more sophisticated and powerful learning design? The parallels to what was known about the brain in 1980 were also quite strong [43].

Recent research both in neuroscience and in technology has made it ever more clear that our earlier beliefs were mistaken.

For example, neuroscience has learned a great deal about the *basal ganglia* [52, 53]. Decision-making in the brain does not follow the highly rigid, pre-programmed kinds of hierarchies that were used in classical AI and in the old Red Army — but it clearly does have a mechanism for exploiting *multiple time scales*. Furthermore, it now seems more and more clear that critic methods can *converge faster* — by orders of magnitude at times — when they include a way to exploit such structure. The challenge lies in how to build systems which exploit such properties as effectively as possible, *without* using ad hoc patches and hierarchies that interfere with the flexibility of learning and the ability to converge in the end to a true multi-level optimum.

Three strands of research have demonstrated promising ideas to help us handle this issue. The field of *hybrid control*, represented here by Shankar Shastry, has studied decoupled methods for the HJB equation, particularly for problems involving

a mix of continuous an discrete variables. AI researchers in reinforcement learning have also explored a number of ideas; the chapter 2 by Dietterich gives an excellent overview, and the chapter 14 by Barto describes further recent work. I myself have also developed some modified multilevel Bellman equations [47, 54] based on the concept of fuzzy or crisp partitions of the state space, which may have some role to play in this strand of research. (These were inspired in part by some earlier work by Sutton [55], and by some work on matrix decomposition theory which I did in 1978, but go substantially beyond the initial inspirations.)

Spatial structure is also very critical, as was already discussed in Section 1.3.3.1. Large-scale network control will be an important testbed in learning how to handle spatial structure, but concepts like the Object Net still leave open some key questions. For example, how do brains *learn* object types? How can systems like the brain implement such structures, using some kind of multiplexing or thalamic gating [56]? How does spatial complexity interface with temporal complexity?

As we move up to systems which truly make high-level decisions, in order to manage multiple levels of time, issues related to stochastic search and stochastic system identification become ever more important. This chapter has already discussed these issues, but we need to remember that they will become more important in the future.

None of this work would allow us to build an artificial human mind. The *human mind* involves a whole new set of issues [1] far beyond the scope of this chapter. But 99 percent of the human brain is more or less equivalent to structures which exist in the smallest mouse. A deeper, more functional understanding of the latter should be a big step forward, in *allowing* us someday to understand the former more deeply as well.

Bibliography

1. P. Werbos, What do neural nets and quantum theory tell us about mind and reality, in K. Yasue, M. Jibu and T Della Senta (eds.), *No Matter, Never Mind : Proc. of Toward a Science of Consciousness : Fundamental Approaches (Tokyo '99)*, John Benjamins Pub Co, 2002. See also P.Werbos, "Optimization: A Foundation for understanding consciousness," in D. Levine and W. Elsberry (eds.), *Optimality in Biological and Artificial Networks,* Erlbaum, 1997.

2. W. T. Miller, R. Sutton and P. Werbos (eds.), *Neural Networks for Control*, MIT Press, Cambridge, MA, 1990, now in paper.

3. D. White and D. Sofge (eds.), *Handbook of Intelligent Control*, Van Nostrand, New York, 1992.

4. P. Werbos, Neurocontrollers, in J. Webster (eds.), *Encyclopedia of Electrical and Electronics Engineering*, Wiley, New York, 1999.

5. M. Hoffert et al, Advanced Technology Paths to Global Climate Stability: Energy For a Greenhouse Planet, *Science*, 2002.

6. P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. Thesis, Committee on Applied Mathematics, Harvard U., 1974. Reprinted in its entirety in P. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*, Wiley, New York, 1994.

7. P. Werbos, Backpropagation through time: what it does and how to do it, *Proc. IEEE*, vol. 78, no. 10, 1990. Updated version reprinted as chapter 8 of [6].

8. J. A. Suykens, B. DeMoor and J. Vandewalle, Nlq theory: a neural control framework with global asymptotic stability criteria, *Neural Networks*, vol. 10, no. 4, pp. 615–637, 1997.

9. P. Werbos, *Stable Adaptive Control Using New Critic Designs*. ArXiv.org: adap-org/9810001 (1998)

10. D. Jacobson and D. Mayne, *Differential Dynamic Programming*, American Elsevier, 1970.

11. P. He and J. Sarangapani, Neuro Emission Controller for Minimizing Cyclic Dispersion in Spark Ignition Engines, TNN submitted, 2003. Condensed versions are in press in IJCNN 2003 Proc. (IEEE) and CCA 2003 Proc. (IEEE).

12. F. Lewis, J. Campos and R. Selmic, *Neuro-Fuzzy Control of Industrial Systems with Actuator Nonlinearities*, SIAM, Philadelphia, 2002.

13. E. A. Feigenbaum and J. Feldman, *Computers and Thought*, McGraw-Hill, 1963.

14. J. Von Neumann and O. Morgenstern, *The Theory of Games and Economic Behavior*, Princeton University Press, Princeton, NJ, 1953.

15. H. Raiffa, *Decision Analysis*, Addison-Wesley, Reading, MA 1968.

16. P. Werbos, Rational approaches to identifying policy objectives, *Energy: The International Journal*, vol. 15, no. 3/4, pp. 171–185, 1990.

17. D. F. Walls and G. F. Milburn, *Quantum Optics*, Springer, New York, 1994.

18. D. B. Fogel, *Blondie24: Playing at the Edge of AI*, Morgan-Kauffman, San Francisco, 2001.

19. T. Landelius, *Reinforcement Learning and Distributed Local Model Synthesis*, Ph.D. thesis and Report No. 469, Department of Electrical Engineering, Linkoping U., 58183, Linkoping, Sweden.

20. R. F. Stengel, *Optimal Control and Estimation*, Dover edition, 1994.

21. K. Narendra and A. Annaswamy, *Stable Adaptive Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1989; Hemisphere, Washington, DC, 1982.

22. R. Howard, *Dynamic Programming and Markhov Processes*, MIT Press, Cambridge, MA 1960.

23. C. J. C. H. Watkins, *Learning From Delayed Rewards*, Ph.D. thesis, University of Cambridge, England, 1989. See also Watkins and Dayan, Technical note: Q-learning, *Machine Learning*, vol. 8, no. 3/4, pp. 279–292, 1992.

24. P. Werbos, Neural networks for control and system identification, *IEEE Proc. CDC89*, IEEE, 1989.

25. D. P. Bertsekas and J. N. Tsisiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.

26. A. Barto, R. Sutton and C. Anderson, Neuronlike adaptive elements that can solve difficult learning control problems, *IEEE Trans. SMC*, vol. 13, no. 5, pp. 834–846, 1983.

27. P. Werbos, The elements of intelligence, *Cybernetica (Namur)*, no. 3, 1968.

28. A. Bryson and Y. C. Ho, *Applied Optimal Control*, Ginn, 1969.

29. J. S. Baras and N. S. Patel, Information state for robust control of set-valued discrete time systems, *Proc. 34th Conf. Decision and Control (CDC)*, IEEE, pp. 2302, 1995.

30. S. Mukhopadhyay and B. Jain, Multi-agent Markhov decision processes with limited agent communication, *Proc. of the Int'l Joint Conf. on Control Applications and Int'l Symposium on Intelligent Control (IEEE CCA/ISIC01)*, IEEE: 2001.

31. T. Kohonen, *Self-Organizing Maps*, New York: Spinger, 1997, Second Edition. Also see H. Ritter, T. Martinetz, and K. Schulten, *Neural Computation and Self-Organizing Maps*, Addison-Wesley, Reading, MA, 1992.

32. J. Albus, Outline of Intelligence, *IEEE Trans. Systems, Man and Cybernetics*, vol. 21, no. 2, 1991.

33. P. Werbos, Changes in global policy analysis procedures suggested by new methods of optimization, *Policy Analysis and Information Systems*, vol. 3, no. 1, 1979.

34. P. Werbos, Backpropagation: General Principles and Issues for Biology, in D. Fogel and C. Robinson (eds.), *Computational Intelligence: The Experts Speak*, IEEE, 2003.

35. A. R. Barron, Universal approximation bounds for superpositions of a sigmoidal function, *IEEE Trans. Info. Theory*, vol. 39, no. 3, pp. 930–945, 1993.

36. P. Werbos, Elastic fuzzy logic: a better fit to neurocontrol and true intelligence, *J. Intelligent & Fuzzy Systems*, vol. 1, no. 4, 1993.

37. X. Z. Pang and P. Werbos, Neural network design for J function approximation in dynamic programming, *Math. Modelling and Scientific Computing*, vol. 5, no. 2/3, 1996 (physically 1998). Available also as adap-org/ 9806001 at arXiv.org. See also P. Werbos and X. Z. Pang, "Generalized maze navigation: SRN critics solve what feedforward or Hebbian nets cannot," *Proc. Conf. Systems, Man and Cybernetics (SMC)*, Beijing, IEEE, 1996.

38. T. Yang and L. O. Chua, Implementing Back-Propagation-Through-Time Learning Algorithm Using Cellular Neural Networks, *Int'l J. Bifurcation and Chaos*, vol. 9, no. 9, pp. 1041–1074, 1999.

39. See P. Werbos posted at http://www.iamcm.org, and [47].

40. P. Werbos, Consistency of HDP applied to a simple reinforcement learning problem, *Neural Networks*, 1990.

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

42. B. Widrow, N. Gupta and S. Maitra, Punish/reward: learning with a Critic in adaptive threshold systems, *IEEE Trans. SMC*, vol. 5, pp. 455–465, 1973.

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

44. J. Tsitsiklis and B. Van Roy, An analysis of temporal-difference learning with function approximation, *IEEE Trans. Auto. Control*, vol. 42, no. 5, 1997.

45. T. B. Trafalis and S. Kasap, Artificial neural networks in optimization and applications, *Handbook of Applied Optimization*, in P. M. Pardalos and M. G. C. Resende (eds.), Cambridge University Press, 2000.

46. K. Narendra and S. Mukhopadhyay, Intelligent control using neural networks, in M. Gupta and N. Sinha (eds.), *Intelligent Control Systems*, IEEE Press, 1996.

47. P. Werbos, A Brain-Like Design To Learn Optimal Decision Strategies in Complex Environments, in M. Karny, K. Warwick and V. Kurkova (eds.), *Dealing with Complexity: A Neural Networks Approach*, Springer, London, 1998. Also in S. Amari and N. Kasabov, *Brain-Like Computing and Intelligent Information Systems*, Springer, 1998.

48. F. Yuan, L. Feldkamp, G. Puskorius and L. Davis, A simple solution to the bioreactor benchmark problem by application of *Q*-learning, *Proc. World Congress on Neural Networks*, Erlbaum, New York, 1995.

49. T. Shibata, *Hierarchical Intelligent Control of Robotic Motion*, Master's Thesis, chapter 5, Dept. of Electronic Mechanical Engineering, Nagoya University, Japan, 1992.

50. P. Werbos, Applications of advances in nonlinear sensitivity analysis, in R. Drenick and F. Kozin (eds.), *System Modeling and Optimization: Proc. IFIP Conf. (1981)*, Springer 1982; reprinted as chapter 7 in [6].

51. D. O. Hebb, *The Organization of Behavior*, Wiley, New York, 1949.

52. J. C. Houk, J. L. Davis and D. G. Beiser (eds.), *Models of Information Processing in the Basal Ganglia*, MIT Press, Cambridge, MA, 1995.

53. K. H. Pribram, (ed.), *Brain and Values,* Erlbaum: Hillsdale, NJ, 1998. (See also earlier books edited by Pribram in the same series from Erlbaum.)

54. P. Werbos, Multiple Models for Approximate Dynamic Programming and True Intelligent Control: Why and How, in K. Narendra (ed.), *Proc. 10th Yale Conf. on Learning and Adaptive Systems*, New Haven: K. Narendra, EE Dept., Yale University, 1998.

55. R. Sutton, TD Models: Modeling the World at a Mixture of Time Scales, *CMP-SCI Technical Report*, pp. 95–114, University of Massachussetts at Amherst,

December 1995, later published in *Proc. 12th Int. Conf. Macjine Learning*, pp. 531–539, Morgan Kaufmann, 1995.

56. C. H. Anderson, B. Olshausen and D. Van Essen, Routing networks in visual cortex, in M. Arbib (ed.), *The Handbook of Brain Theory and Neural Networks*, First Edition, pp. 823–826, MIT Press, Cambridge, MA, 1995.