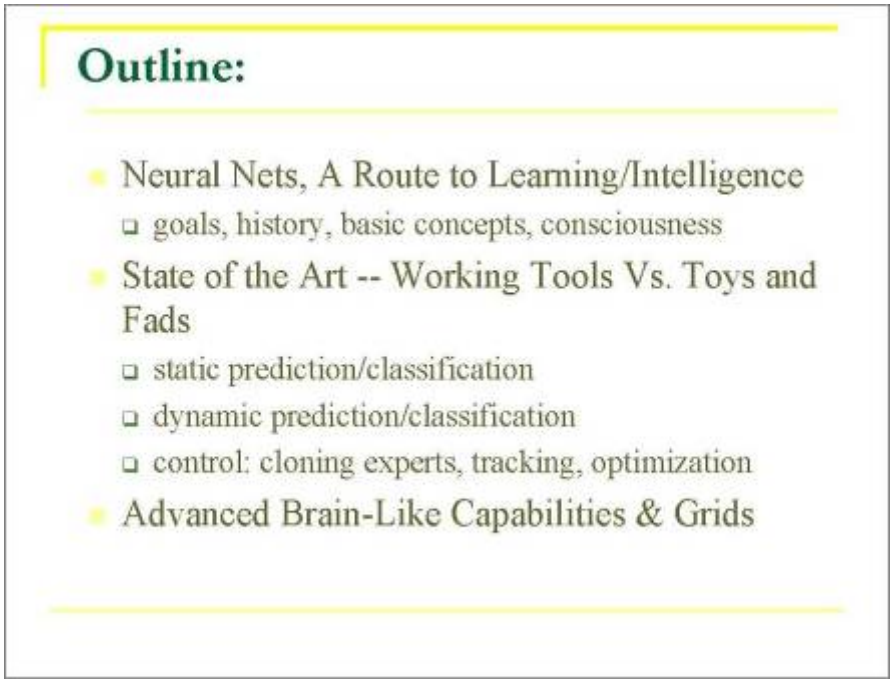
Neural Networks That Actually Work In Diagnostics, Prediction & Control:

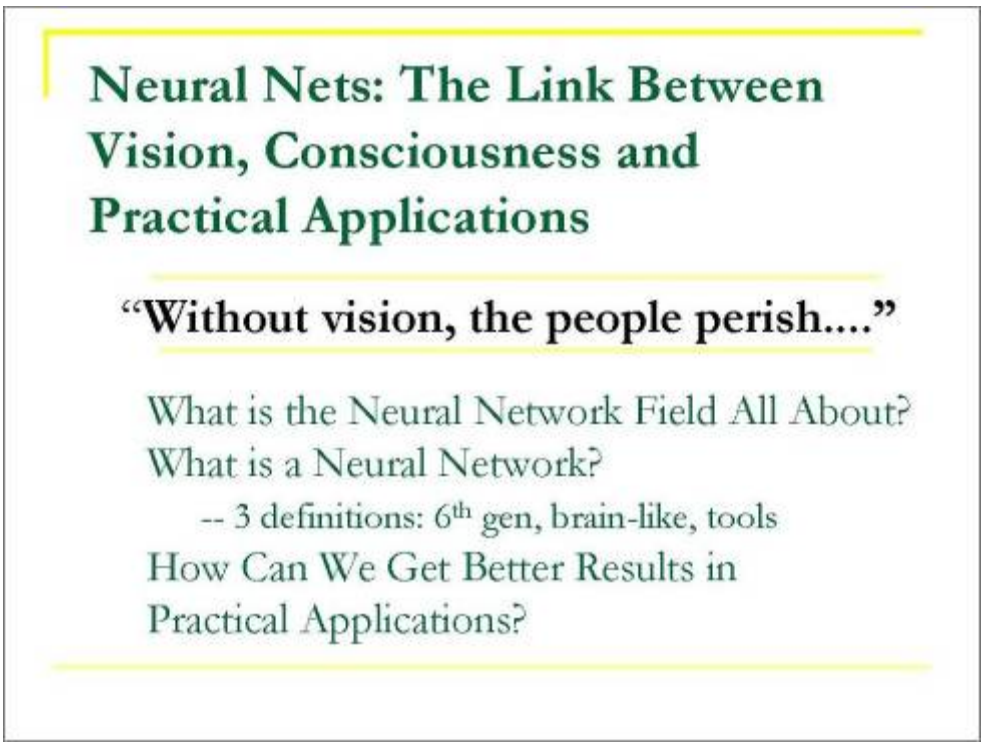
Common Misconceptions Vs. Real-World Success



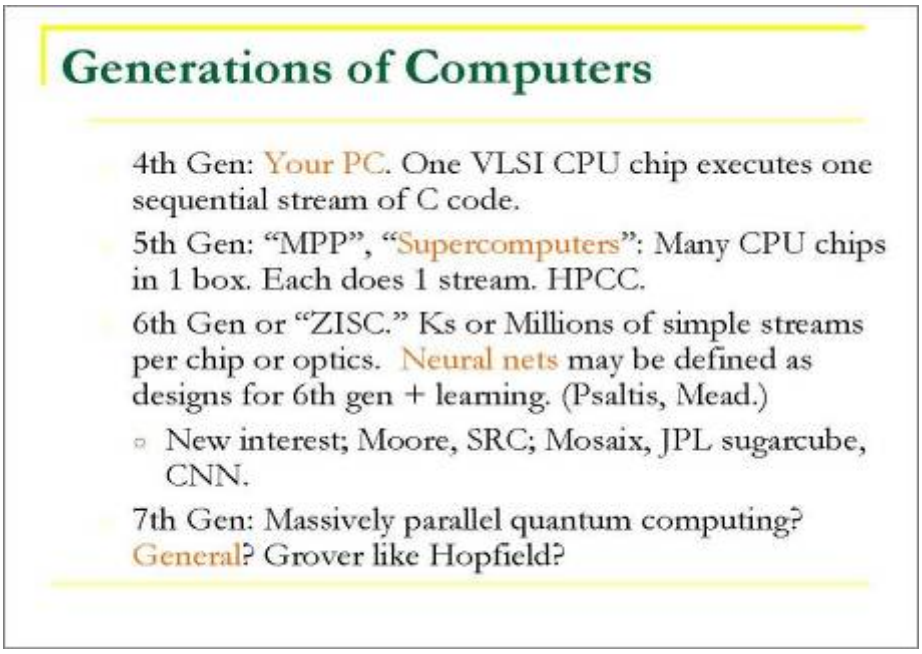
ABSTRACT: Neural nets are used in a growing variety of real-world applications, in a growing number of areas (including the modeling otural intelligence). But this has been a mixed blessing, as various subcultures have not kept up with each other. For example, at http://www.eas.asu.edu/~nsfadp/ applications of adaptive dynamic programming (aka advanced reinforcement learning) are reported whi beat the previous best performance in electric power grid control, aircraft control, in large-scale logistics, etc. --but best performance requires integrating concepts from many subcultures. Yet in some subcultures, people are still using "mappings from sensory to motor coordinates" (direct adaptive control), whose limits were clear years ago. Some are using indirect methods inspired by control theory with well-known stability theorems --theorems whose conditions are rarely satisfied, and methods which are easily improved upon. Similarly, in prediction, some subcultures falsely believe that recurrent networks are hard to train and of marginal benefit, even as people in industry u em routinely, and report performance better than extended Kalman filters and equal (at lower cost) to elaborate particle filter methods. e foundations of learning also bear on the practical choices. This tutorial will try to offer a kind of practical roadmap of these issues, and suggest how it points towards future functionality in tasks as large-scale as what brains can handle.

As neural networks have been used in more and more fields, more and more courses have been taught from the viewpoint of other fields, such as AI, statistics, biology or control theory. This is good in a way, but it often doesn't tell people what the neural network field is really about. This tutorial will begin by discussing the core vision of the neural network field, which is still a revolutionary new paradigm and has yet to achieve full understanding from the mainstream. It will then go into depth into the best state-of-the-art tools for predictions (including diagnostics) and control or decision-making or management of large systems – including a discussion of the pathway for how to get to systems which replicate the kind of higher-level intelligence we see in mammalians brains.

Some related slides (with Notes) are posted at www.eas.asu.edu/~nsfadp and at www.iamcm.org/~publications, and various URLS herein. Please forgive the informality of these notes, which reflect no official views and are tentative, oversimplified, etc.



Today, I will really be talking about three things -the basic vision of the neural network field, -common pitfalls in the field and opportunities to get better results in practical applications -how to scale up to handling LARGER problems, in particular – problems like optimal management of large-scale power grids. Many people in engineering feel uncomfortable talking about broader goals and vision. Sometimes engineers even feel uncomfortable talking to the head of a factory, when the factory director tells them what he really needs. But the fact is that we all need to have a sense of goals and directions, or we can get lost. There has been a lot of confusion lately about neural networks and what they are, as people have sometimes forgotten the fundamental goals and principles of the field. But don't worry, I also have some specific equations to point you towards. Almost never will I show 100 equations in a talk… but you can find a lot more than that if you follow up on some the citations I will give you. When I start talking about "what is a neural network" and "what this field is about," let's face up to the fact that there are really three DIFFERENT definitions of what the neural network field is about. All three are legitimate in their own way, and we are lucky that they are reasonably consistent with each other in practice. Some people define artificial neural networks (ANNs) as a collection of tools available off-the-shelf. That's one definition, and it's an important part of the field. But the neural network field is not really a collection of inherited tools. The field is defined by the QUESTIONS we ask, and the GOALS we are trying to achieve. The next two slides will describe two different definitions of what our goals are. These definitions may sound very different, but they actually turn out to be almost the same in practice.

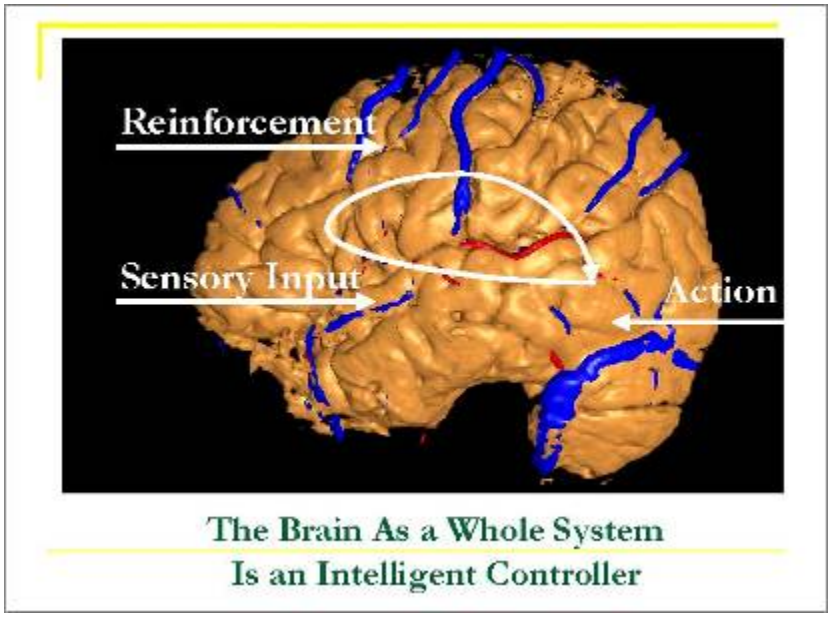


NSF initially funded "neuroengineering" as an offshoot of work in optical computing. Photonics people like Psaltis argued that optical computing could provide a kind of "sixth generation of computing," far beyond the fifth-generation massively parallel computers then popular in funding agencies. People like Carver Mead said they could achieve the same benefits in VLSI – but only if we could devise new architectures or algorithms to allow such chips to be useful in GENERAL-PURPOSE computing. Is it possible to achieve really general-purpose kinds of capabilities within the constraints of enormously massive parallel processing, using very simple processing units?

Mead cited the brain as living proof that it is possible. That then created a challenge to research: how can we make good on this example, by coming up with algorithms or designs for brain-like capabilities that we could implement on such chips?

After initial enthusiasm to build neural chips, to make good on the high potential throughput, there was a long lull in interest. Most people using neural networks as tools deliberately used the simplest designs that they could easily implement on PCs or in MatLab – which reduced the motivation and market for new neural chips. There were a few important achievements nevertheless – a Motorola design that came close to real market; a chip from Mosaix LLC that came extremely close to widespread deployment (and may yet) and missed more for political reasons than technological reasons; the "sugarcube" design giving enormous throughput albeit in a specialized application (image matching for missiles); and Chua's Cellular Neural Networks. But on the whole, it was hard to keep up with Moore's Law.

More recently, as more people have worried about how we can continue Moore's Law, the need to increase functionality per chip has started to attract attention again. Some view quantum computing as the ultimate salvation of Moore's Law… but, like 6th gen computing, it has an "algorithm bottleneck," and may need our kind of massively parallel distributed design.



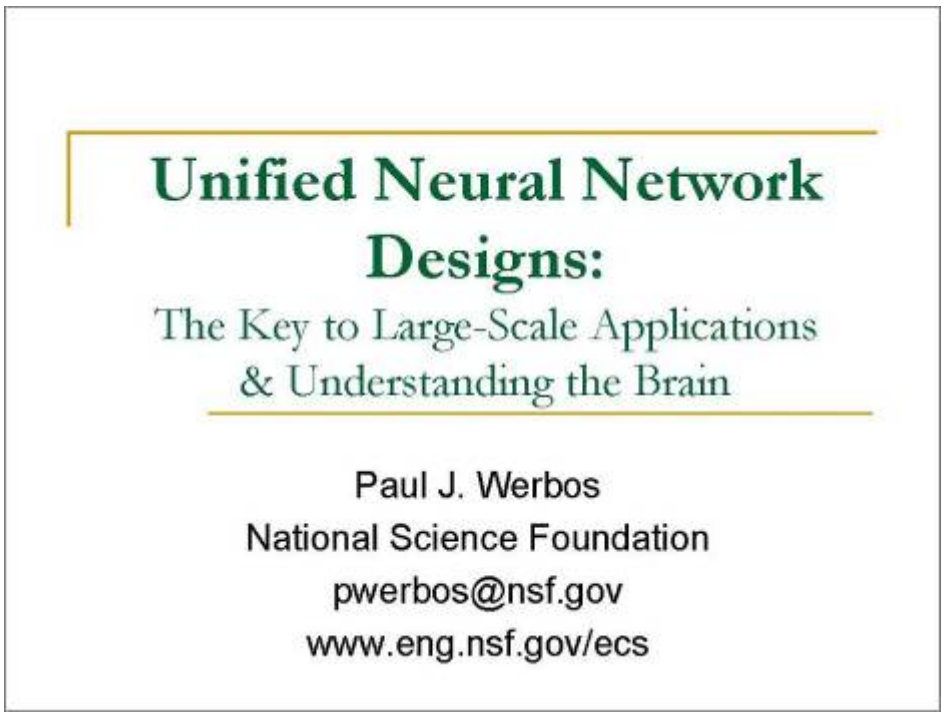
ANNs have also been defined as “Abstract Neural Networks.” They are defined as well-specified mathematical systems designed to capture the highest kind of intelligence that we find in mammalian brains. They are designed to capture the functional capability of the system, not the precise details of what is presently known about the hardware itself.

But how can we understand what the brain as a whole system really does? What kind of mathematic is needed, in order to express the kind of function which this information processing system performs?

In this slide, I remind us that the brain AS A WHOLE SYSTEM is an intelligent controller. It includes pattern recognition and memory and prediction and other key capabilities as subsystems – but you can’t really understand what a subsystem is doing unless you see how it fits in as part of the larger system. Every piece of the brain has evolved so as to contribute to the function of the whole – the function of calculating decision outputs (sometimes called “squeezing and squirting”) which contribute to the long-term goals of the organism.

Thus in order to develop an integrated, functional understanding of how the brain performs this function, we need to understand the mathematics of effective intelligent control, that really works in flexibly learning to handle wide varieties of tough control problems.

Many areas of technology try to teach their students a textbook of a hundred alternate methods to solve a hundred different tasks. And they often try to teach neural networks that way. But that does not do justice either to the brain or to what the neural network field is about. The brain provides a SINGLE flexible system which somehow INTEGRATES the various principles of learning and control, so that ONE system can do it all! There are still variations from brain to brain, but each individual mammal brain has a kind of universal learning ability. Mammal brains do not start out as an “empty slate” – but they are capable of relearning almost all of the specific abilities that they are normally born with. Our primary goal in ANN research is to capture that universal learning ability in designs we can implement and use.

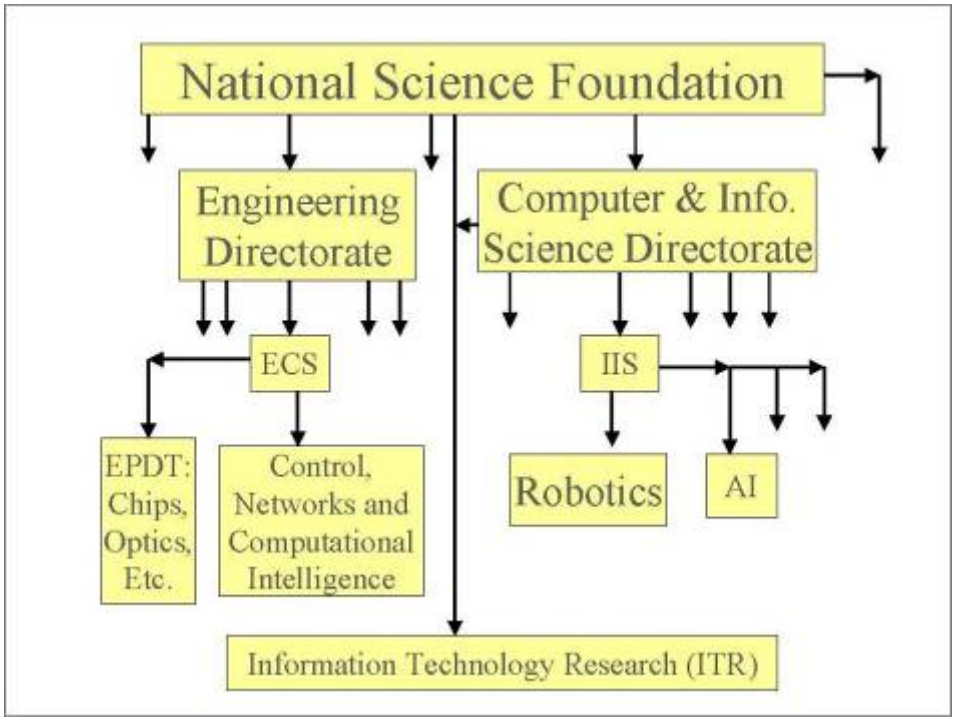


This slide repeats and stresses a key point I just made: in the neural net field, we try to develop UNIVERSAL designs. That's a major part of what defines the field. The brain is living proof that it is POSSIBLE to design one system that "does it all," that incorporates and unifies all the principles needed for intelligence and intelligent action.

As a practical matter, we don't need universal designs for everything. But when we have a universal design, we don't have to make quite so many decisions about the details, and we can move on to new tasks much more easily than we can when we use narrow domain-dependent tools. The initial cost is higher when you develop a more universal design, but after that, each new task becomes much easier.

There are many simple tools in the ANN field which really do have limited, narrow use in practice. But there is a ladder of ever more sophisticated tools, which do have more generality … a ladder which we can see rising up above us, through the clouds, to intelligent systems as powerful and flexible as the mammal brain.

Sometimes people break off and ask the question:"Hey, how good are mammal brains anyway? Should I hire a mouse to do my trades on the stock market? If we want a mammal brain, why don't I just use a human?" In fact, there certainly are times when we simply need to make better use of our own human brains, to solve an engineering problem! But with ANNs, we can make them pay attention. And we can insert them into little boxes in manufacturing plants or cars or airplanes, where they will keep working 24 hours a day, at gigahertz speed, and not cost a lot. If we use ANNs instead of humans to read ZIP codes and sort mail, they will not do a better job (it now seems) – but we can save a lot of money, and free humans to do more interesting things. But these are just a few examples.



The core home for ANN research at NSF is in the CNCI program, for which I am one of the Directors. The magic words “neural network” help route proposals to me, if that is your intent. (The more the better!)

But outside of the core, neural networks – while fundable – are often badly misunderstood.

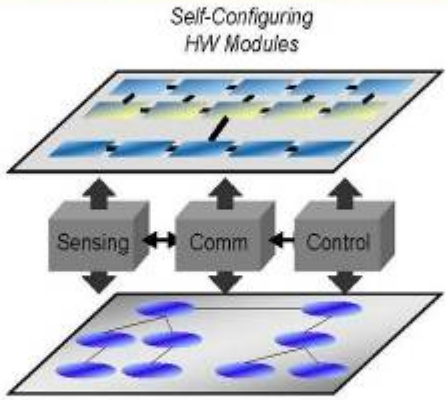
People often think of ANNs as a very narrow class of tools – in part because they are not aware of the large scope of what is being addressed in the field. ANNs are not just the designs NOW in MatLab; they include new designs in the literature, and any future designs which address the larger, more fundamental goals I just discussed.

ANNs are actually the central glue of "embodied cyberinfrastructure," a concept depicted on this slide.

There has been growing interest in studying and enhancing the "embodied cyberinfrastructure" of the US – the vast "nervous system" of network connections from sensors to actuators on a national scale, used to manage and integrate all kind of physical infrastructure networks, like electric power, communications, transportation, finance, water, etc.

And yet – this kind of "artificial nervous system" really is the kind of integrated system we are trying to design CORRECTLY in the ANN field! We have for decades been addressing the exact problem of designing the whole flow of information ("the information layer") from sensors to actuators, in a complex distributed system. ANNs have sometimes been CALLED "parallel distributed processing" (PDP) systems. We have more work to do, of course, but this is what we have been working on, and we have made progress.

Cyberinfrastructure: The Entire Web From Sensors To Decisions/Actions/Control For Max Performance



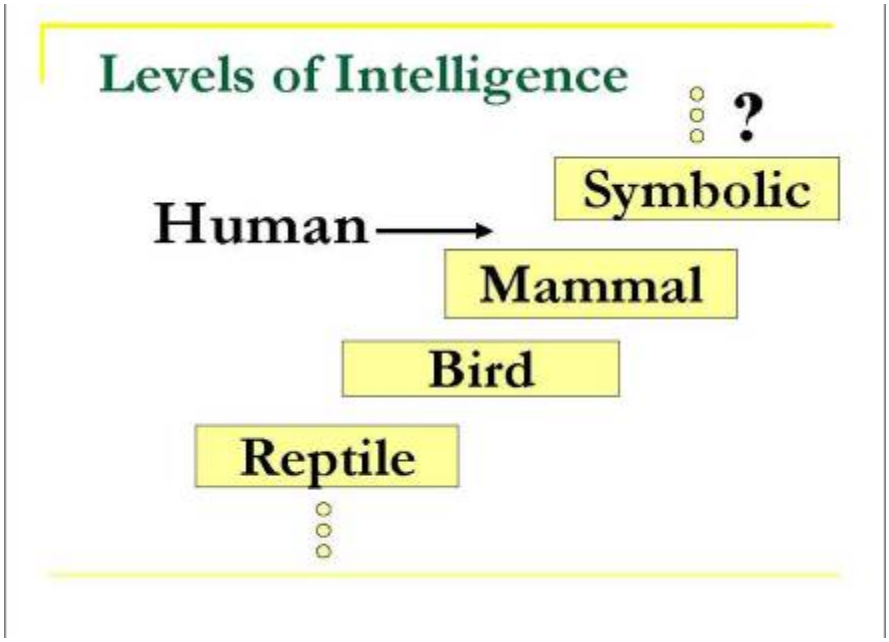
This chart really elaborates on the preceding one.

Notice that I have added two key words: "Max Performance."

It is easy to develop a general-purpose controller for any complex system – so long as one does not care about performance. Likewise, ad hoc domain dependent solutions can survive well in areas where they only compete with each other, and where maximum performance is not pursued in a deep intellectual way. Much of the literature on "agents" is limited in that way.

It is easy even to assure stability, if one does not care about performance at all. One need only drop a bomb on the system. Then it won't move at all.

The real challenge here is to find general methods to maximize performance. The desire to avoid catastrophe can be incorporated into the MEASURE of performance. There is a strong technical basis for this claim: research into nonlinear robust control has shown that general nonlinear robust control problems map rigorously into optimization problems ("the Hamilton Jacobi Bellman equation.").



This slide leads into many important issues. For now, I want to use it to discuss the relation between ANNs and AI.

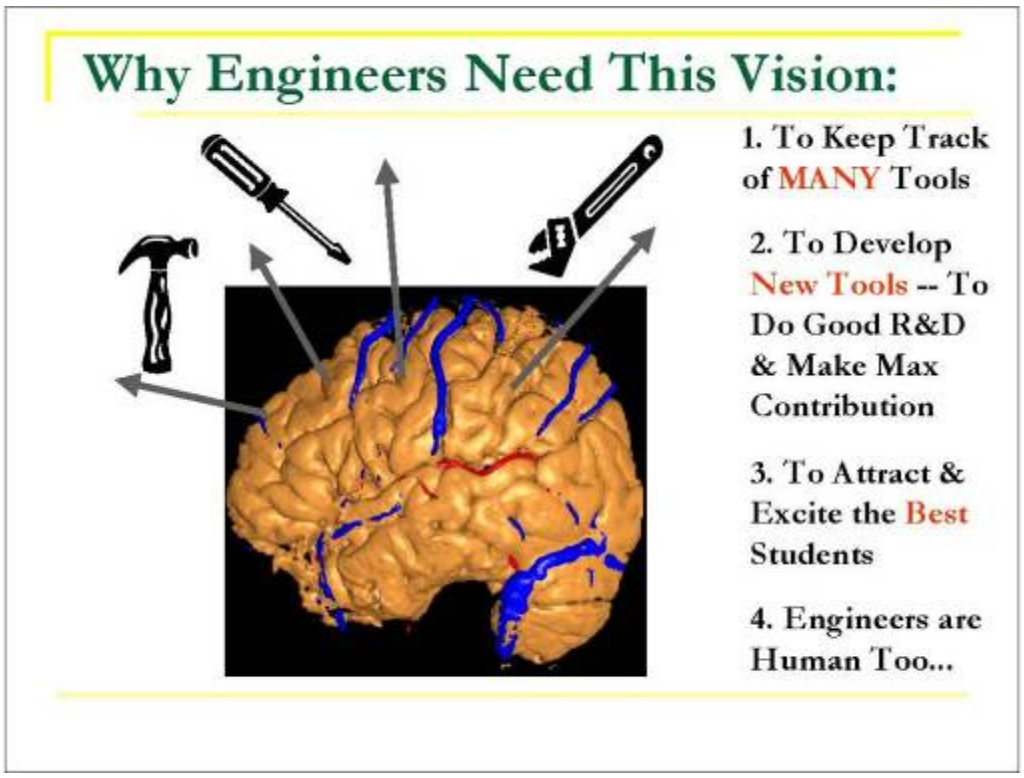
Back in the 1960's, AI announced goals very similar to what I have just discussed. In fact, the early work on perceptrons was accepted as part of AI at that time. And now, many people teach courses in "modern AI" which do allow some work on ANNs to be included.

But even today, the mainstream of hard-core classical AI addresses the question: "How can we build an artificial Einstein?" They recognize, quite correctly, that the highest level of intelligence we can all see in nature is the human level. That level of intelligence is based on some (partial) use of symbolic reasoning. And so they try to jump directly to symbolic reasoning.

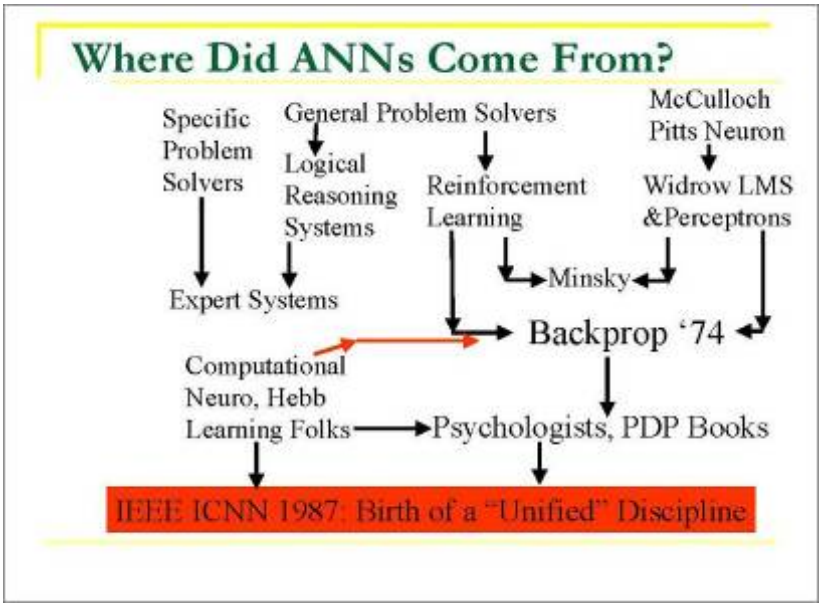
For practical purposes, the target we pursue in ANN research is SUBSYMBOLIC intelligence, the kind of general-purpose nonverbal intelligence that we can find even in the smallest mammal brain. Yes, our tools can be used as part of artificial symbolic reasoning systems, but that is not what guides our research. For now – achieving the level we see in the smallest mammal brain is a difficult enough mathematical challenge. Only recently have we begun to develop the tools which give us a practical roadmap for reaching this destination – and we have a lot of work left to do in implementing, integrating and applying these tools.

For those more interested in the symbolic level… we need to remember how 99 percent of the human brain is "homologous" to the mouse brain. Thus a true scientific understanding of the symbolic level will require a firm grounding in what we are only now beginning to learn at the next level below that.

Occasionally I hear psychologists or philosophers say: "You build things, therefore you must be reductionist." This is as untrue as the worst racial stereotypes. But still – the question mark on this slide is beyond the scope of this tutorial.



Engineers often ask “What do you need this vision piece for? Shouldn’t we just drop it?” In fact – the vast majority of neural net applications I have seen in the past two decades have fallen very far short of accomplishing as much as they could have, if they had had a better cause-and-effect top-down understanding of what they can get from neural nets. Understanding what the methods are FOR is crucial to using them effectively. When there are hundreds and thousands of papers and software commands competing for attention, it is crucial to understand the LADDER (or upside-down tree) of capabilities which exists, rising up from the simplest most specialized methods through to more and more powerful methods which capture more and more of brain-like capabilities. To obtain funding from ECS, of course, it is especially important to know “how high” on the ladder the present state of the art is, in order to plan a project to take us up one step. The ladder provides a kind of sense of direction. In addition, to attract the best students, it really helps to have a broader vision, a vision of a field which will offer new and serious opportunities enough for a lifetime, a vision which offers the creative person more than JUST a meal-ticket. (Then again, many students may also realize that learning general-purpose methods will give them a lot more flexibility than domain-dependent details which can easily go out of date.)



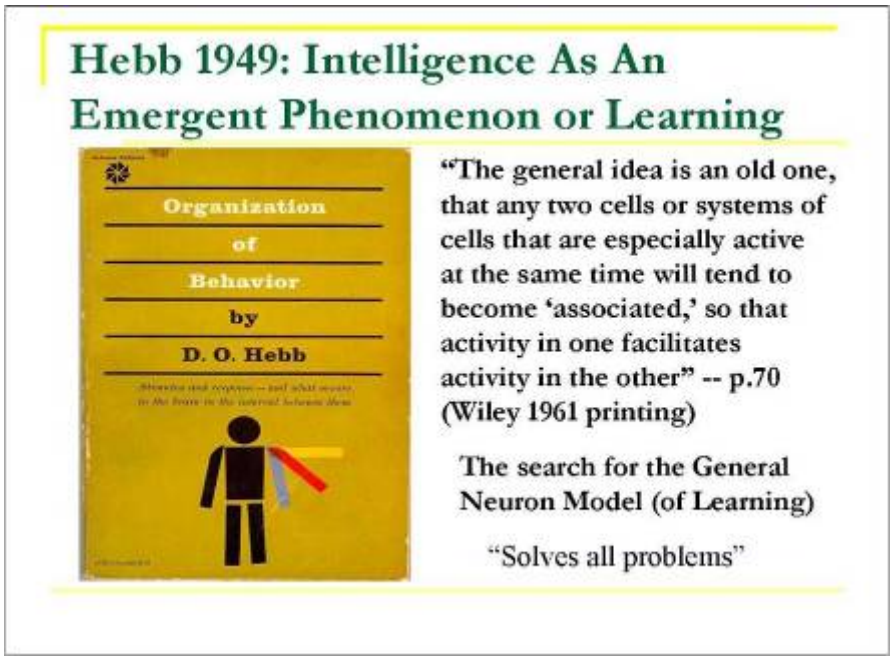
Many people think of ANNs as one kind of classifier system, for pattern recognition. But that is only one part of what they are.

Back in the 1960's, AI truly focused on the goal of trying to achieve brain-like intelligence, as such. The "perceptron movement" (including Widrow, Rosenblatt and others, inspired in part by Von Neumann!) was a kind of early neural network movement, WITHIN AI; they are the ones who tried to get started by addressing the classification problem. But, as they were unable to solve even simple classification problems like XOR, pessimism grew, and the mainstream started to view neural nets as the very worst and most unmentionable kind of taboo heresy – an "old discredited idea that could never work." Minsky's classic book Perceptrons became the Mainstream Official verdict on the field. Minsky's most famous conclusion: "You can never do XOR without a multilayer training procedure; none is in sight."

In the early 1970's I found a way to overcome that problem, and even offered to Minsky (and others) to collaborate on putting it forward. I learned a lot about heresy and taboos and the academic system. (See Talking Nets, MIT Press, for a small part of the story.) But eventually, the discovery of backpropagation – included in my 1974 Harvard PhD thesis, and generally cited as the original source – did get out. The simplified, popularized version in the 1986 PDP books stimulated the "birth" or "rebirth" of the neural net field more than any other event.

But in fact – backpropagation only halfway came from the perceptron school! I first formulated it as a way to solve the reinforcement learning problem, which had been ANOTHER mainstay of mainstream AI. (That stream was also blitzed in a paper of Minsky and Selfridge, who said they couldn't make that work either, and many people in AI still conclude that no one could make it work on a useful scale. But as you will see, times have changed. We have found some solutions.)

Just as important as AI to ANNs was the "Hebb" stream. One might even say that Hebb was the grandmother of ANNs, and AI the father.

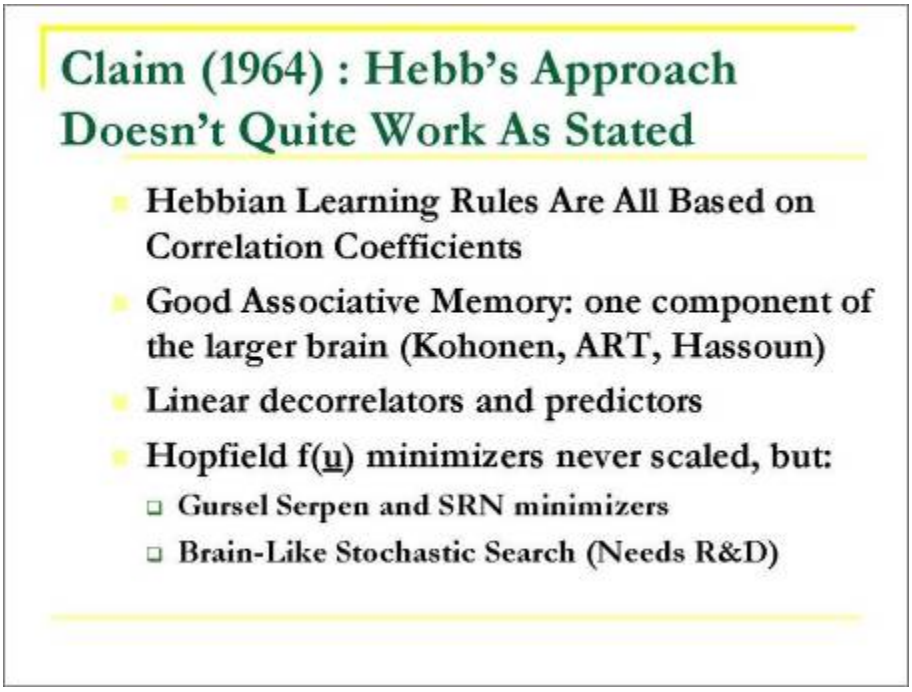


Hebb's classic book was a great inspiration to all the mains schools of artificial neural networks – including the engineering school!

I have seen hundreds of people excited by the New Vision of Emergent Behavior in Complex Adaptive Systems. But in my view, Hebb's version of that vision – back in the 1940's – was actually far more complete and real than the new reinvented versions!!

Hebb never claimed that ANY complex system would automatically evolve into becoming intelligent as a whole system! The universe is full of complex and dynamic but dead planets, and live but low-intelligence swamps – and some would say that the federal government is the best proof that maximum complexity does not always yield maximum intelligence. Rather, Hebb argued that the right kind of simple dynamics could allow complex intelligence to emerge. He was inspired in great part by the great experiments in neuroscience by Lashley, Freeman, Pribram and others, demonstrating "mass action" – the ability of any part of the higher brain to learn almost anything, if connections were in place. (For example, many creatures are born with edge detectors in the rear of their brain… but neurons in middle part can learn to be edge detectors if the rear part is damaged!)

Hebb inspired a great search for a "general neuron (learning) model" which would have the required property – the property that a great heap of those neurons, all connected at random, could learn to do almost anything. He had an intuitive idea that this rule should reflect the idea that "behaviors are reinforced by repetition;' the mathematical translation of that simple idea has come to be known as Hebbian learning, and is still a main foundation of computational neuroscience, both in its classical (continuous variable perceptron-like with differential equations) and "spiking neuron" versions.

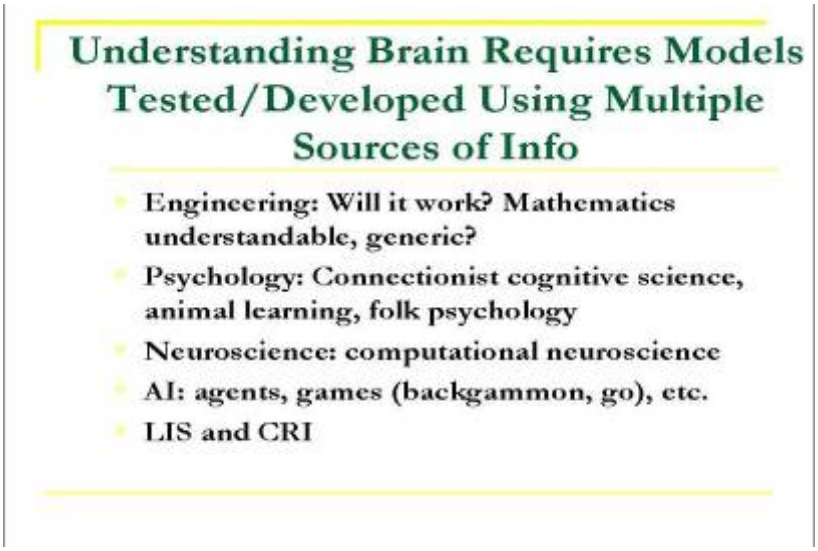


I can remember how intensely inspired I was when reading Hebb's book back in the summer of 1963 or 1964. I owed a paper to the University of Pennsylvania summer school in computers which I had attended the previous summer, showing how I was using computers in a summer job – and I decided to write a program implementing Hebb's idea, and showing how it could be used on simple problems in learning in AI. I tried, very hard – but that was one course where I had an incomplete. And I kept trying.

In the end, I concluded that ONE kind of neuron simply could not do it – could not encompass learning the whole range of capabilities that a brain can learn. Later, Stephen Grossberg (then a young assistant professor at MIT) showed me his papers from 1969, where he COULD use Hebbian learning for an associative memory function.. And that is an important PART of brain function. But that is not the whole thing.

In engineering applications, other than clustering, Hebbian learning methods and "Hopfield" or Cohen-Grossberg nets have become harder and harder to find in recent years. The real reason, in my view, is that they never were able to achieve competitive performance in the larger prediction and decision-making tasks engineers have to do well on. Certainly an all-encompassing design ought to be able to replicate what linear regression can do, more or less – but Hebbian learning essentially gives correlation coefficients, not regression coefficients. (Still, some people have tried to stretch the definition of Hebbian learning to include even backpropagation through the back door, and it is amusing to see "Hebbian" interpretations of the backwards-flowing NMDA synapses observed by Bliss etc. a few years ago, published in Science.)

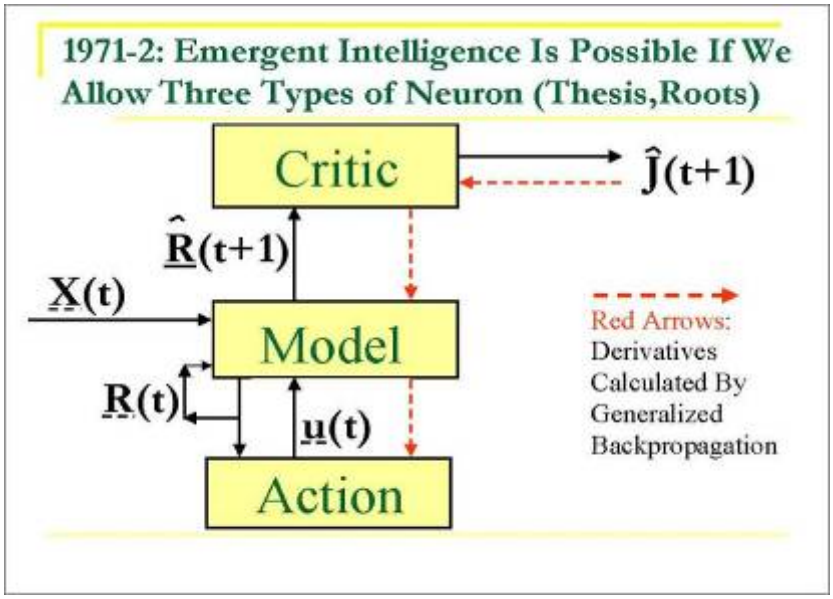
Strictly speaking, I think I can see a way that a general "neuron learning rule" could be constructed to cheat, to compress many types of "neuron" into one – but it would be a mess, not like anything either nature or a good computer designer would want.



In my (personal) view, the engineering failure of Hebbian learning by itself is a lesson of extreme importance. If we can't achieve strong functional capabilities – general-purpose abilities to handle the broadest range of tasks – by using Hebbian learning, what does that say about the possibility that the brain itself uses Hebbian learning by itself to achieve these learning abilities?

In fact, Sigmund Freud argued long ago that an addition kind of information flow between neurons would be indispensable, in understanding the basic learning abilities or psychodynamics of the brain. (See Karl Pribram's statement on the back cover of my book, The Roots of Backpropagation.) Pribram and Freeman have both reiterated often the point that today's modeling conventions in neuroscience are simply too restrictive to be able to capture or replicate those kinds of capabilities. The brain itself is a functional system, and we need to learn more about the mathematical design of functional systems, systems which can perform real engineering tasks, before we can hope to understand what is really going on there. We need to develop new types of higher-level brain models which try to COMBNE the best of what has been learned from all of the disciplines on this chart!! Engineers have a critical role to play in making this possible, by developing the kind of designs and mathematics that will be needed for this future crossdisciplinary effort.

Almost ten years ago, I worked with Joe Young and Andy Molnar from the psychology parts of NSF, to start up a new research funding initiative, ultimately called Learning and Intelligent Systems (LIS). I like to believe that LIS played a key role (along with the successes of ANNs) in reinvigorating the attention played to learning in many fields, especially neuroscience and AI and statistics. But it also became clear that we on the technology side have a lot of homework to do first. We need to develop and consolidate what we have learned about learning, and to cut across the disciplinary barriers which exist even within technology. In the long-term ---our ability to understand ourselves better, someday, will depend critically on this kind of tangible mathematical/engineering work.



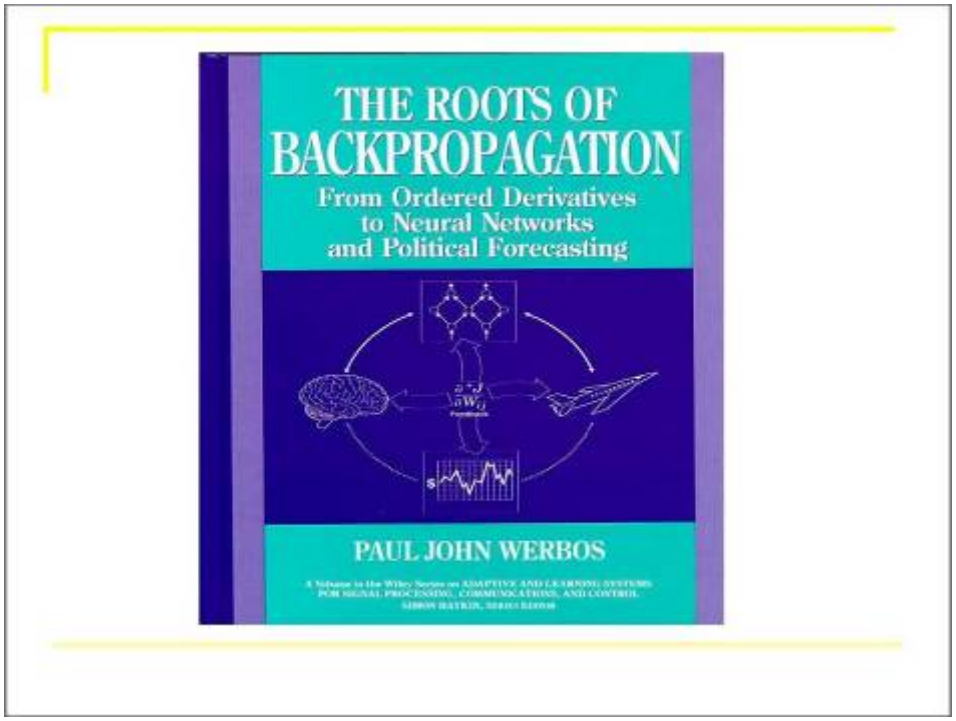
After I gave up on ever implementing Hebb's vision of "one neuron does it all," I aimed for a slightly relaxed version – which actually does work, in a formal mathematical kind of sense. It actually is possible to design a general-purpose learning system out of three types or block of neurons – a system which actually can learn to converge to the optimal "strategy of action" in the general case, in some sense.

In actuality – this mathematical design turns out to be a direct translation of ideas from Freud into mathematics. That's where backpropagation REALLY came from. (Some of the details are in chapter 10 of my book Roots.) Backpropagation is actually a general-purpose way of calculating derivatives through any large sparse nonlinear differentiable system. In fact, some of the people I sent my thesis to renamed this an "advanced adjoint method;" their work then led to subsequent work implementing this in actual circuits, to calculate derivatives efficiently through local calculations in real hardware.

The application to static classification is only one of many applications.

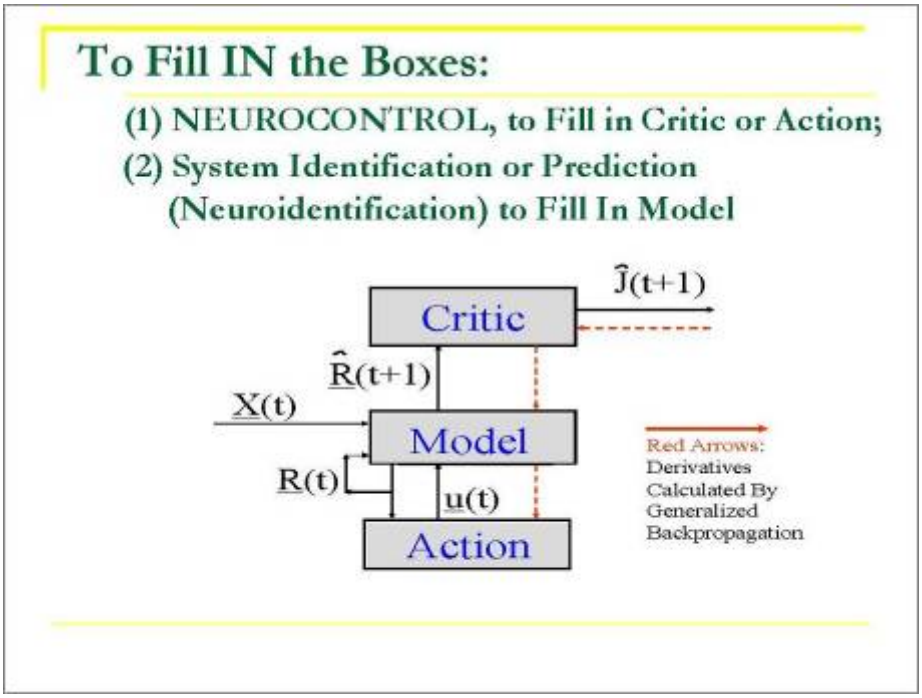
It has taken many years for engineering practice to catch up to this level or complexity of design. It is much higher on the "ladder" of capabilities than what most people are doing even today with neural networks. I will take many slides to explain the basic ideas, and it will take further citations to explain how to implement it in detail.

Back around 1987, I thought that the higher-level intelligence of the mammal brain might be explained as an emergent result of something this simple underneath – more complex than Hebb's ideas, but still in the same spirit. However, as the complexity of the sensor inputs X grow, the learning speed of this design gets slower. Recent research has extended the ladder upwards, to allow faster learning – as is needed when we try to truly integrate the management of very large systems like electric power grids!

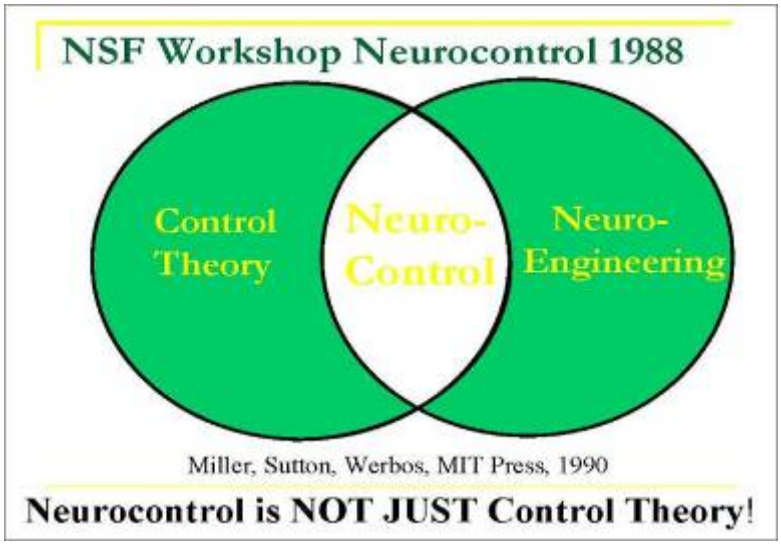


See previous slides. The entire PhD thesis is reprinted here.

Chapter 8 also gives a more modern update of how to use backpropagation through time, etc. Chapter 7 gives the 1981/1982 published paper which, in my view, actually overcame the "heresy" barriers for the initial idea – and also showed how backpropagation permits intelligent control or reinforcement learning designs far more powerful than the simpler temporal difference ideas which it also discussed. The thesis proves the validity of the "chain rule for ordered derivatives," the most general form of backpropagation for all kinds of ordered (feedforward) systems. And it includes applications to political forecasting models and to time-series estimation.



When I started running the ANN area at NSF in 1988, I naturally did NOT restrict it to funding my own old ideas! But in considering these ideas, it seemed clear to me that there are two major tasks involved in trying to reach brain-like intelligence in future large ANN designs: (1) the "control" or decision-making task, which defines the overall system; and (2) the task of learning the dynamics ("Model") of the environment or plant. Thus I would still see learning-to-predict and learning-to-decide("control") as the two main divisions of research needed to fulfill our goals. They are the two pillars of the ANN field. Areas like clustering and associative memory are important PIECES of these larger areas, when they are designed to contribute to the larger goals or systems.



How can we achieve brain-like intelligent control by designing learning systems made up of simple processing units like neurons?

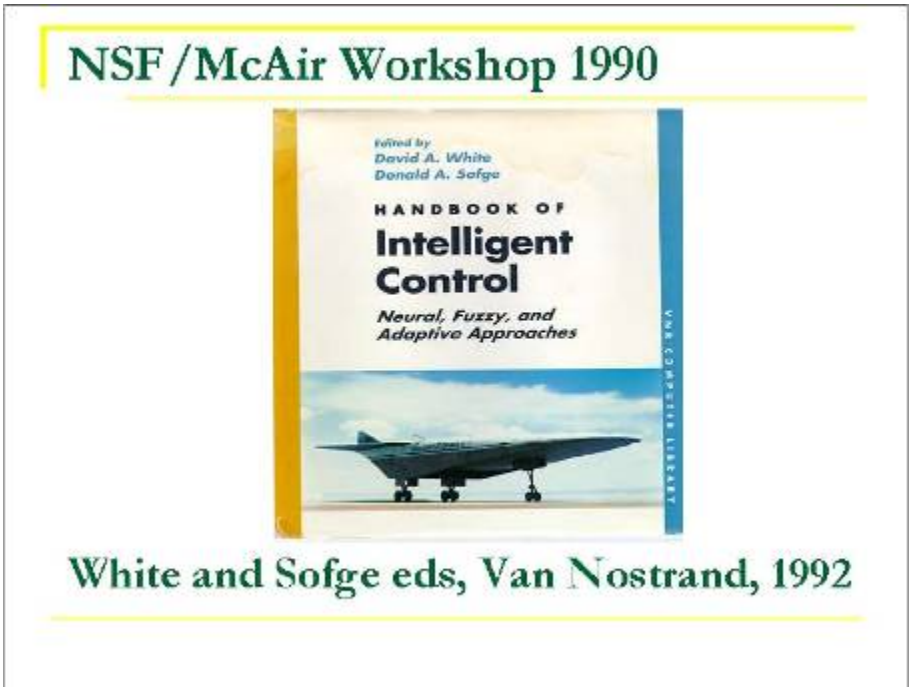
Whatever the brain does – it does do "control" (i.e.deciding on action), and it is made up of neurons. It is a neurocontroller. Thus to achieve brain-like capabilities, we need to build up the science of neurocontrol as illustrated here.

Neurocontrol is a subset of ANN design, since it does built up systems out of ANNs. But it is also a subset of control theory, since control theory addresses the general problem of all kinds of systems designed to output useful streams of control decisions over time. For best results, we clearly need to draw on BOTH disciplines, as much as possible. This is why my first action at NSF was to contact Tom Miller and Richard Sutton to organize the first workshop on neurocontrol, to try to bring these disciplines together and create a new area. The book from MIT Press was widely influential, and did inaugurate this new field.

But there have been a lot of misunderstandings and lessons learned in later years.

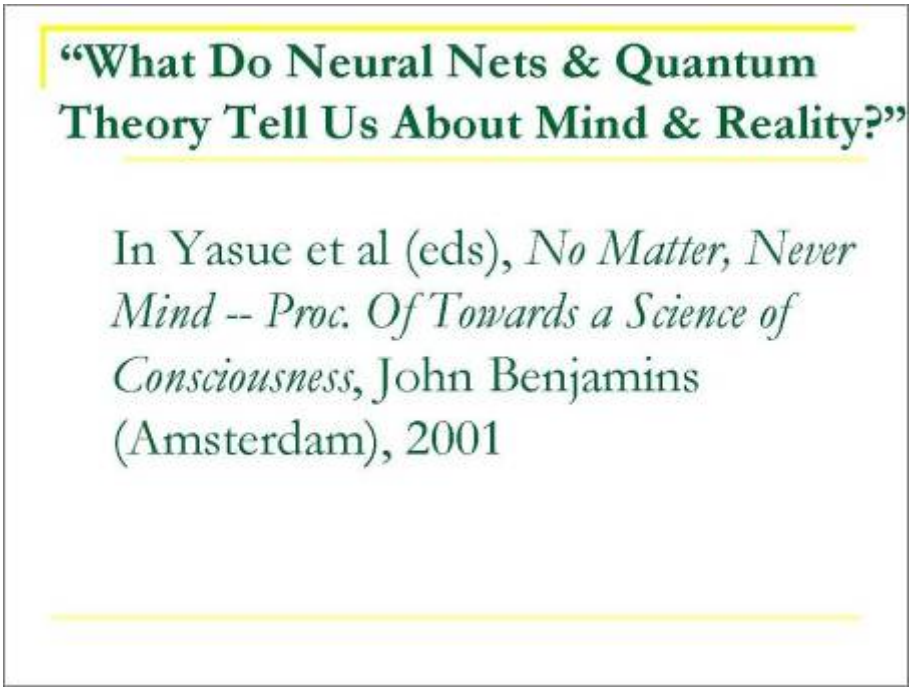
First, many people outside of engineering have a paranoid misunderstanding of what control theory is all about. "Are you turning the brain into a slave that you control, denying it all autonomy? These are not learning systems, they are just controlled systems.." This is a gross misunderstanding. We are developing general purpose learning systems which THEMSELVES can control engineering plants, for example. Neurocontrol designs vary in regards to how much autonomy they have, but our ladder rises up to as much autonomy as a human brain possesses.

Second, it is not necessary that everyone in this area possess a PhD both in control theory and in neural networks. There are few people like that. But it is critical that we do try to learn a few core ideas from each other. I hope that this tutorial will help. Very few people know enough about both fields to do the very best that ca be done in real-world applications; however, we all can learn a step or two more, and do that much better. I do strongly hope for more unified curricula in the future that will help overcome these gaps. After 1990, some people decided to use the new word "neurocontrol" for different things…



The success of the previous workshop led to a follow-on workshop in 1990, jointly sponsored by NSF and McDonnell-Douglas. The resulting book, the Handbook of Intelligent Control, is still the most advanced source now available for many of the key technologies and capabilities to be discussed here. It has served as a kind of "Bible" for my program, and has important details of implementation.

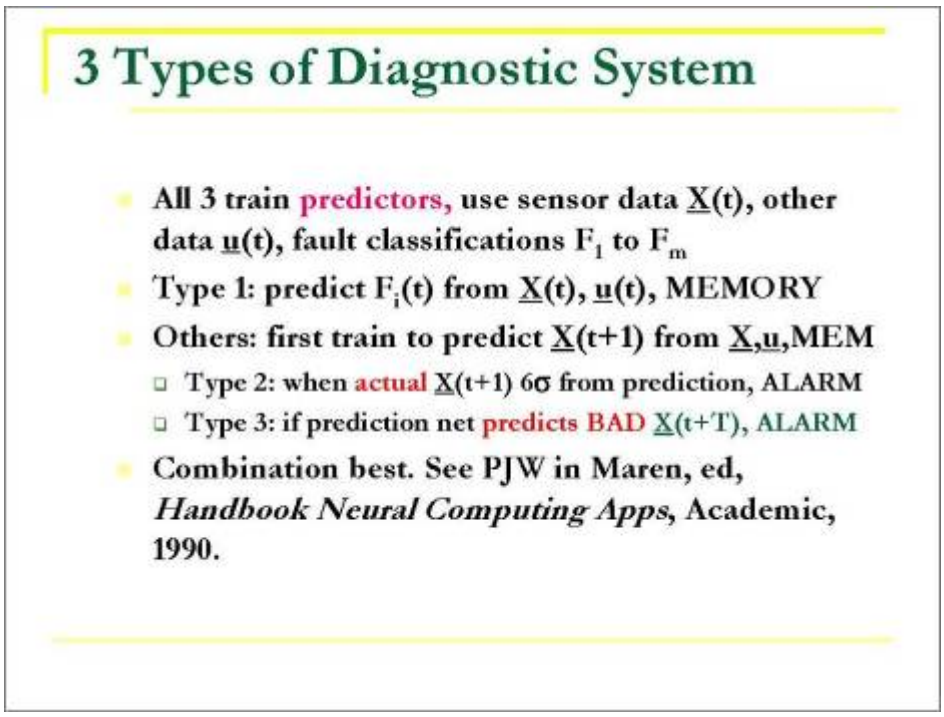
Many of the new advancements since then have appeared only as papers of various sorts. However, there is a new Handbook of Learning and Approximate Dynamic Programming in press from Wiley and IEEE Press (see www.eas.asu.edu/~nsfadp) which substantially updates and elaborates many of the core themes of the Handbook. I will refer often to the old Handbook (HIC) and the new Handbook (HLADP) from here on.



This slide completes my overview of generalities and philosophy. (I will probably be much faster on this aspect in Budapest.)

But if you seriously wonder whether this kind of mathematics might help us somehow to understand human consciousness and the like, you might be interested in the paper mentioned on this slide.

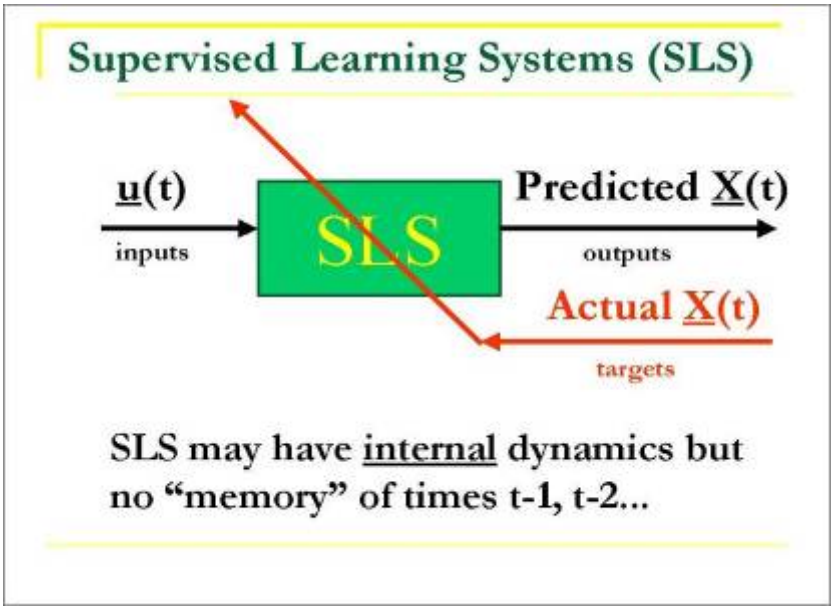
Actually, this book is the selected proceedings of a global conference Towards a Science of Consciousness, hosted by the United Nations University (UNU). My chapter discusses prior work on the connections to neuroscience and psychology, mainly in various books edited by Pribram but also by Levine and others. The chapter may be found on the web, at arXiv.org, in the q-bio series for 2003. My views here do not fit into any of the extreme positions widely publicized in the public debates on these complex issues.



This is my one slide on diagnostics!

Diagnostics are extremely important in engineering. But the neural network methods useful in diagnostics all boil down to methods for prediction. This slide describes the three main ways that general-purpose prediction tools can be used as part of a fault diagnosis system. There are many applications out there in the field, and I will say much about the art of how to apply prediction methods well for diagnosis. The book edited by Maren (Academic Press, 1990) does contain a lot of practical discussion of how such systems can be set up. In this tutorial, I will discuss how to do prediction better, which is important to all of these uses of prediction.

I do have a couple of advanced suggestions, however. For optimal detection of "novelty events" (X(t+1) very different from trained predictor)… it is usually best to first train a TLRN predictor network robustly, as I will discuss below. And then the ERRORS and the recurrent nodes, or just the sensed data and recurrent nodes, can be plotted using one of Kohonen's self-organizing maps. Either system alone has worked well in diagnostics, but the combination should work better. Alternatively, Principe's new learning rule based on comparing pairs of observations could be used, perhaps to replace the SOM part.



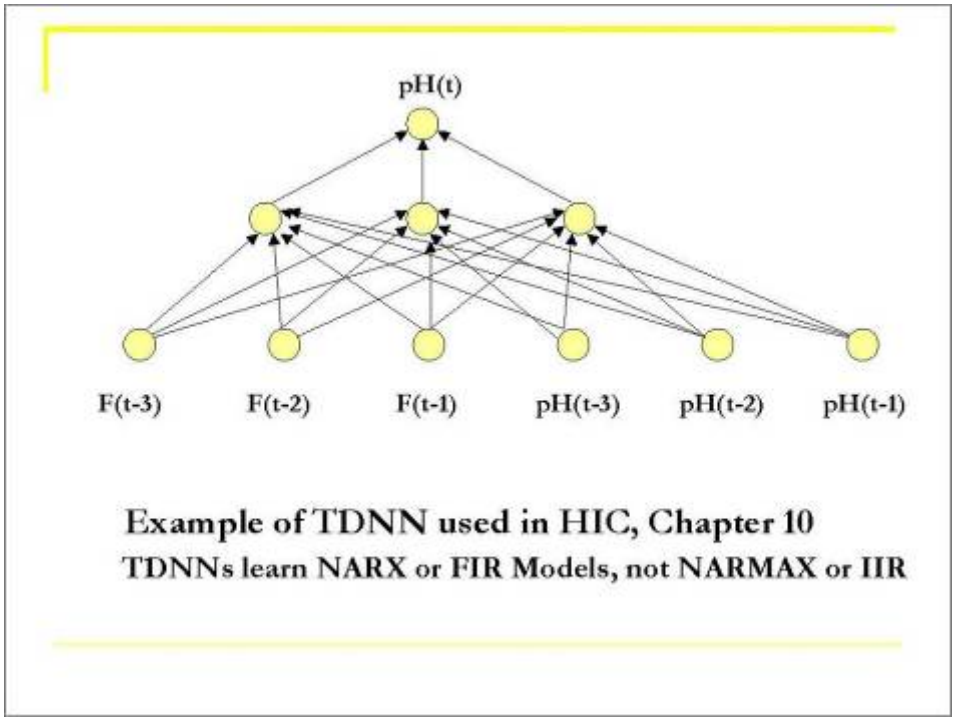
The term "supervised learning" is very precisely defined now. Supervised learning refers to the general task of trying to learn the mapping from a vector (or set) of inputs to a vector (or set) of desired outputs, based on previous examples of inputs and desired outputs. The idea is to learn the mapping based on a past history or database or "training set" of examples. The idea is to learn the mapping IN SUCH A WAY that the future predictions are likely to be good predictions of the actual or desired value, IN THE FUTURE, for new input vectors X not seen in the training set. (This is called "good generalization.")

Some people in AI now call this "learning from examples," but we should not use that term because it is very ambiguous.

If the desired outputs are binary or discrete, we call this a static classification task. That is a special case of supervised learning.

Normally, the learning system contains parameters or weights W which we keep changing, in order to make the predictions fit the actual data better and better. Sometimes we examine the observations one-by-one, and change the weights based on observation-by-observation learning; this can be done in an offline simulation, as a way of training an engineering system that will be fixed in its operation. Sometimes it is true real-time "on the fly" learning. Sometimes we wait until we cycle through an entire training set in a "batch," and only then update the weights, in "batch learning." All have their place in engineering. The brain seems to be a real-time learning system – but that actually turns out to be an oversimplification!

Universal supervised learning is a hard enough challenge by itself for today's research! ANNs, statistics, AI, philosophy, psychology, etc., all have important contributions to make in helping us achieve more brain-like capability even in this "narrow" task. This task is a kind of subset or starting point for the more general dynamic prediction that the brain or an intelligent controller needs to perform.



This slide is an example of how people have used neural networks in the past to do prediction over time, in chemical engineering. Basically, we convert a problem of prediction over time into a simple supervised learning task. We define the INPUTS to the SLS as the observations (pH) and decisions (F) over the last three time-periods, and then try to predict the current pH level as the desired output. When we use a static input-output neural network to make predictions over time, by "lagging" or "delaying" the inputs like this, it is called a "Time-Delay Neural Network." (TDNN).

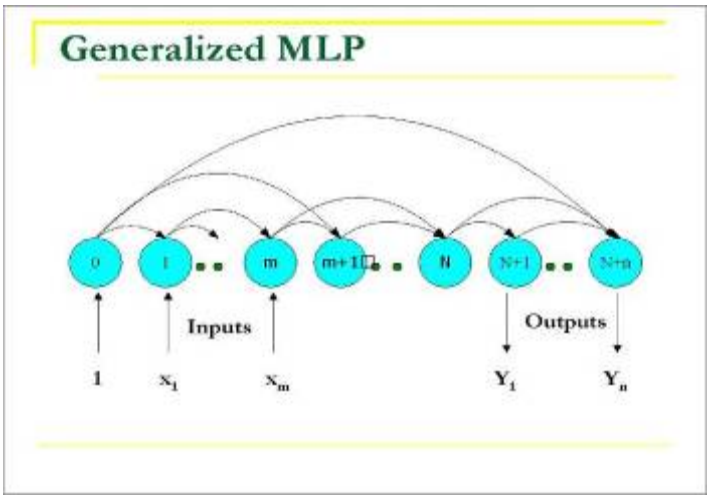
In this example, the circles and arrows basically indicate WHAT KIND of static input-output network we used here. We used a three-layer "Multilayer Perceptron" (MLP), the most common ANN across all engineering applications. Some people call it a "backpropagation network," but that is very confusing terminology, since MLPs were used back in the 1960s (before backpropagation even existed!), and also because backpropagation has been used on much wider classes of networks. (Some people like to use confusing terminology because it gives them a chance to invent dozens of new names for the same algorithm.) See www.iamcm.org (publications) or my book for the equations of the MLP; in essence, each circle represents a two-step computation – a weighted sum v followed by calculating tanh(v). The weights and intercept of the weighted sum form the weights to be trained or tuned.

If we want to do prediction by using simple TDNNs, we face two decisions:

(1) Should we really use an MLP, or should we use some other static architecture?; and (2) how should be train or tune the weights? The next slides address that.

In practical engineering, people usually agonize over whether they should use an MLP (a "global" network) or a simple nearest-neighbor kind of network (a "local network"). This slide summarizes the practical tradeoffs. In essence, an MLP gives better generalization, if the input data really varies over a space of more than three or four dimensions. (But if the inputs are all heavily clustered around a few dozen strong attractor points, it may be good enough just to cluster the data and use a local network.) A local network – like a lookup table – gives much faster learning. People who want to prove that their local network is "faster than a backpropagation network" (??) will often use a trivial, slow implementation of backpropagation to try to prove their point; however, modern implementations such as the RPROP variation in the SNNS shareware package (see www.ieee.org, Neural Networks, for pointers) or the DEKF variations used by Feldkamp et al, or even the simpler Adaptive Learning Rate algorithm defined in HIC chapter 3, tend to be much faster. But even with acceleration – dumping an observation into memory is still faster.

As a rule then, applications demanding high accuracy or complexity tend to use MLPs (or complex extension of them), and may use offline learning with structures that allow high offline adaptive behavior from fixed weights. Many real-time systems use local networks. Mixture of experts models (Jordan et al, Weigand, Principe, etc.) provide a partial hybrid solution, using ensembles of global networks. Likewise, James Lo has a hybrid solution (which could be implemented simply by applying different ALR rules to different layers). A more brain-like solution would be to use a concept of "syncretism" (also to be found in Freud, and in HIC chapter 3); the closest thing to an operational version of this is Atkeson's "memory-based learning," combining fast memorization and slow learning/generalization from memory. Atkeson (CMU) and Schaal (USC) have beautiful demonstrations of physical humanoid robots learning agile movements like tennis backhands, by first learning to emulate a human role-model and then refining (optimizing) their performance. But that is the top of (one trunk) of the "tree" of designs, beyond what most applications need; there is also need for research to take this still further.

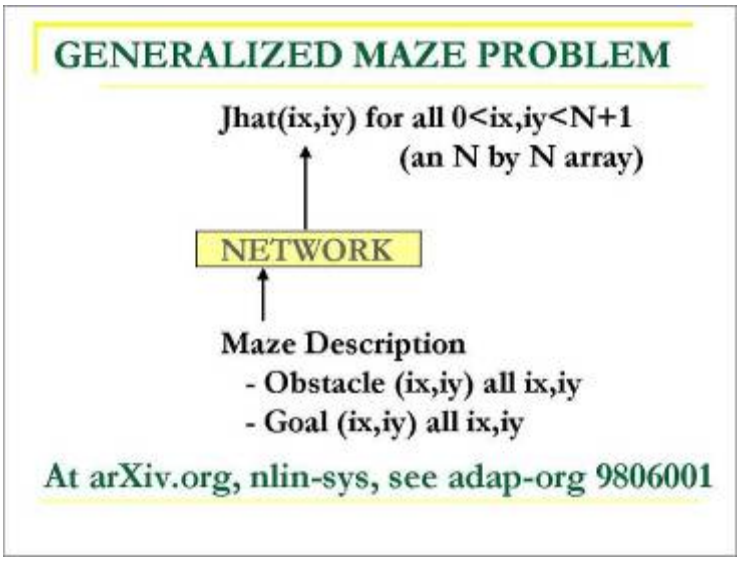


Based on the important theorems of Andrew Barron, many now believe that the MLP is a kind of practical, well-scaling universal approximation machine. When the task is to learn smooth functions of many inputs, it is clearly better than the ever-popular linear basis function methods (which include the easier local networks, Taylor series, and feature-based classification and its "new" kernel function versions). Not all functions are smooth – but don't we have to make SOME realistic assumptions like smoothness in order to have any hope at all in learning?

In fact – it is extremely important to basic research to reconsider – what DO we have to assume in order to make learning possible at all? What are the minimum apriori assumptions we have to make – or the most general class of worlds we can really learn to adapt to – consistent with learning? One key difference between the eld of statistics and the field of neural networks is that we absolutely MUST face up to this question, in order to have any hope of explaining how the brain copes with so many millions of input variables. We must face up to the kinds of questions which philosophers like Emmanuel Kant wrestled with – which turn out to be critical to performance in real-world applications like image processing, where Germans trained in Kant have in fact achieved outstanding performance!

It turns out that many of the functions which the brain needs to learn, in order to survive in our world, are well beyond what MLPs can learn!! Intuitively, these are applications where there are many inputs that tend to have a kind of entangled sort of interrelation.

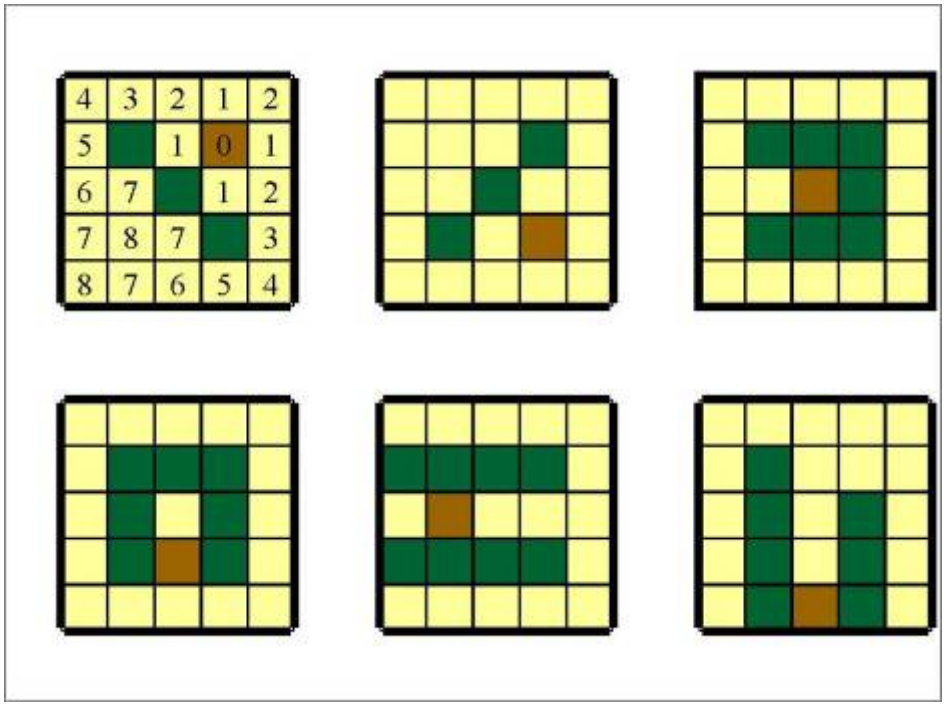
The GMLP (above) is the most general, all-inclusive topology available for feedforward networks made up of the usual simple "sigmoidal" (e.g. tanh) neurons. It is an example of a universal design for what it does; ANY feedforward topology made up of sigmoidal neurons is just a special case of a GMLP, a special case in which some of the weights may be "cut" (fixed to zero). At the IJCNN1999, in the session led by Pribram, Russian mathematicians showed how such architectures give more general function approximation capabilities than MLPs. See www.iamcm.org for equations, etc. – and some discussion of the well-known issues of pruning and penalty functions important to optimal performance in ALL global feedforward networks. (Penalty function tools should also be in good ANN packages, to address the well-known "overfitting" problem, which I will discuss further.)



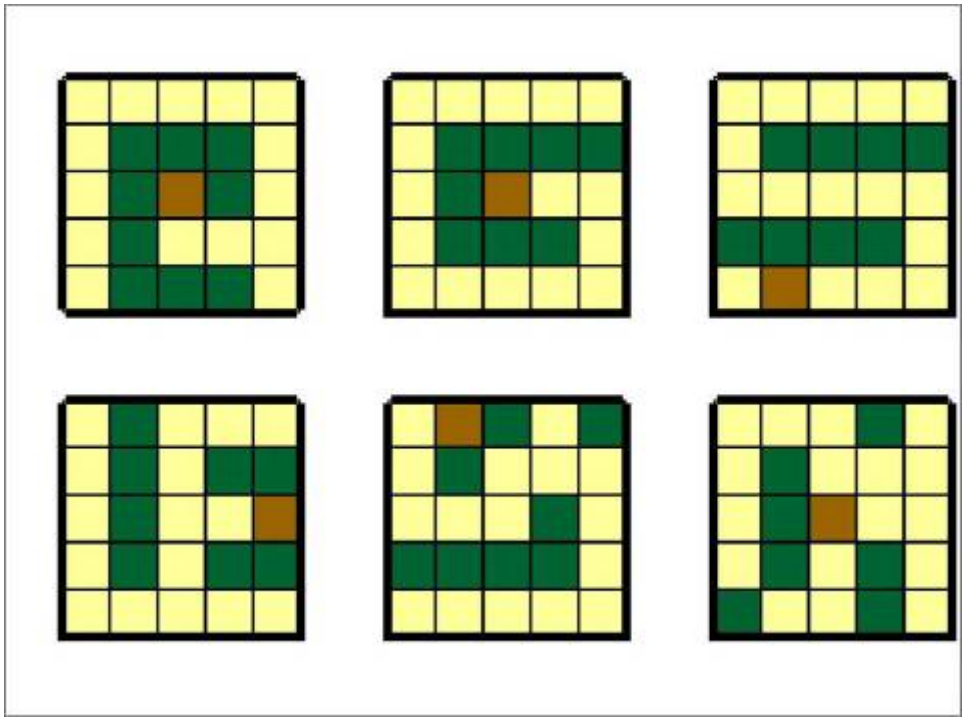
But… certain supervised learning tasks simply cannot be performed by ANY feedforward network, global or local. (Or by any network trained by Hebbian methods, whose occasional success is based on nearest-neighbor prediction.) The theory behind this was discussed in HIC – but a practical example may be more useful.

Consider the "simple" problem of trying the learn how to navigate through an N by N square maze. Many AI people have studied the problem of "learning the maze" – of sending a robot into an unknown maze, and having it bounce off the walls a few hundred times until it finally blunders its way through. But this is not what we want real robots to do. It is not even what we want unmanned ground vehicles to do. We want systems which, like humans, can learn the GENERALIZED ability to LOOK AT THE MAZE and see a way through, as fast as possible. (And, yes, humans may ALSO use mental simulation, but not ONLY that.)

This leads immediately into the Generalized Maze task illustrated here. Usually people describe this as the task of learning to map from an "input vector" which is actually a 2D description of where the goals and obstacles are, to a target which is the direction to move to get to the goal as soon as possible. But for a simple experiment, it is easier to define the targets as the dynamic programming "map" or evaluation of different squares in the maze, which immediately shows the robot which neighbor to move to. It turns out that MLPs fail miserably in either version of the task. Likewise, "Simple Recurrent Networks" (a version with simple training popular amongst psychologists) fail miserably. But the web page above explains how Pang and I were able to perform this task, using another kind of neural network topology, called a Cellular SRN. (This generalizes the older type of "SRN," as I will discuss.) There is reason to believe this is essentially the most general useful way to represent nonlinear input-output mappings – and that it is critical to brain-like success in applications such as autonomous navigation, segmentation and connectivity analysis, success in strategic games, etc. Thus if an MLP is not good enough… look up the web page…



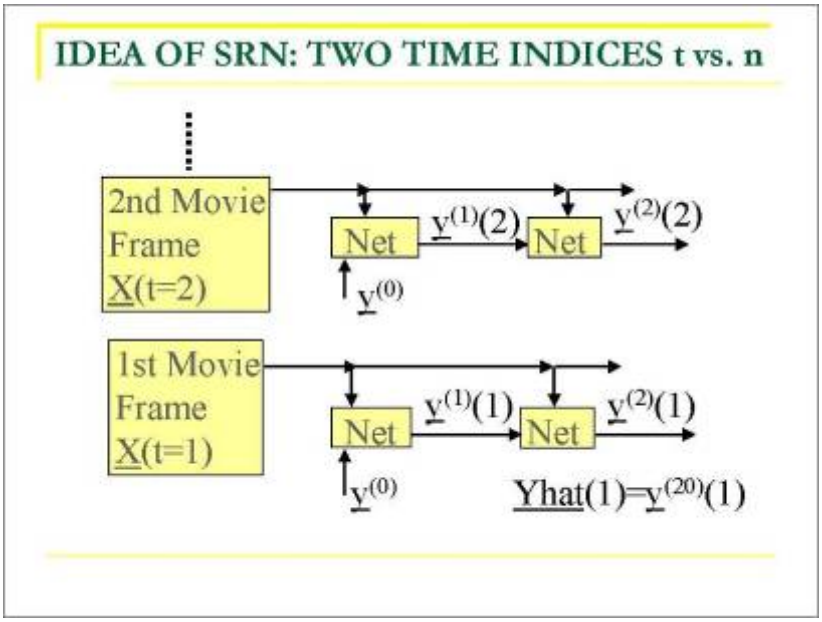
Why do I claim MLPs would not work on Go, and what alternatives might exist? This slide and the next slide depict a study I did with XiaoZhong Pang several years ago, on neural networks to approximate a deceptively simple-looking value function. (See xxx.lanl.gov/abs/adap-org/9806001 for the original study. These slides represent an extension I later presented at IJCNN.) The issue here is maze navigation --a problem often discussed in RL papers. But conventional RL papers discuss “learning a maze.” This is not how humans do obstacle avoidance. We do not just bump against the walls and “learn the room” every time we enter a new room. We learn a GENERAL skill, the skill of LOOKING at a room, of INPUTTING a visual image, and then seeing our way through. The goal here was to train a neural network to input the 5-by-5 pattern of obstacles (in black) and goal (in yellow), and then output the value function. In my final study,I trained a neural network on all squares in six easy mazes (shown here), and showed that the learning generalized to better performance on six hard mazes (the next slide) which the network was never trained to. But an MLP was absolutely unable to learn this task, or even to come close. The problems here are very similar to the problem of learning the “connectedness” map which Minsky talked about, years ago, in his book on Perceptrons. If ADP systems using MLP critics cannot even learn such a simple general obstacle-avoidance task like this, how could they ever learn more complex problems, like how to extract a car from a parking lot? Or how to navigate a dangerous outdoor field -something the mouse is very good at? Or how to play a game like Go, where patterns of connection are crucial to the strategic situation?



After the work with Pang, I actually trained a simple cellular SRN on the six easy mazes (above), and then used these 6 as a test set. No matter how long I trained on the training set, performance on the test set kept improving.

That was reported in several brief papers, including Proc. IEEE-SMC (Beijing) and the Proc of the Yale Workshop on Learning and Adaptive Systems (available from Narendra at Yale).

In all fairness, however, the learning was slow. I believe it can be made much faster, just as basic backprop with MLPs was accelerated after some effort. I have a project in the works to try to show this (though my time is very limited). Certainly the brain does this somehow, and the web page describes several learning methods hardly explored in our brief study. For the time being, evolutionary computing provides one way to train such networks – especially if one uses methods (like particle swarm optimization) which are better than discrete genetic algorithms for tuning continuous variables. But their scaling abilities are limited, and it will be critical to work towards a more brain-like solution to faster training here.

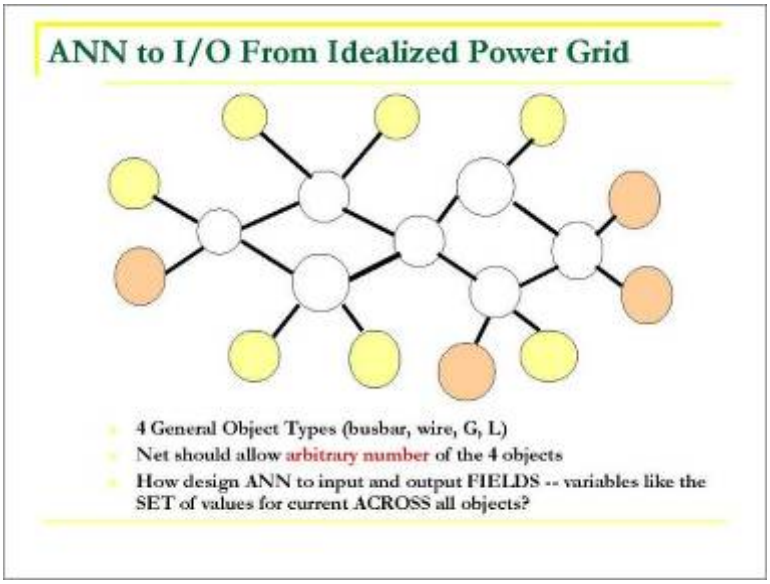


This slide depicts one aspect of the neural network which Pang and I used to solve the generalized maze problem.

We used a special type of Simultaneous Recurrent Network (SRN). In a paper on recurrent networks and natural gas models (in Neural Networks, Oct. 1988), I proposed a more general version of SRN, discussed at great length in the Handbook of Intelligent Control (1992). For an ultimate, general-purpose time-series identifier (like a system to process video data), one needs two kinds of recurrence or feedback, and two time scales: one to provide "memory" (time-lagged) and one to provide "relaxation" or "iteration" (simultaneous). For the maze, we only needed iteration, because it was a static mapping problem.

Psychologists have since used "SRN" to refer to simpler, less general networks trained in a simpler way. In our study, we found that such simpler recurrent nets worked as poorly as MLPs! In our work, the learning and the code was as slow and as crude as the early Rummelhart-et-al demonstrations of MLPs to learn XOR; however, there is lots of room for others to do better, even using methods described in that paper. Or one could use evolutionary computing, like Fogel, to demonstrate tangible results even before these learning-speed issues are mastered.

Yang and Chua (Int'l J. Bifurcation and Chaos, June 1999) replicated the work with Pang, and showed it is
compatible with the type of high-throughput CNN chips they have designed and had manufactured. Another key aspect of the design here was that the underlying feedforward component net was "cellular." It exploited symmetry with respect to translation and rotation to drastically reduce the effective size and complexity of the network --allowing a simple network to handle a complex problem. There is no way to handle brain-like complexity without exploiting symmetry in some such way. But a more general version of symmetry will be needed to handle more general situations.



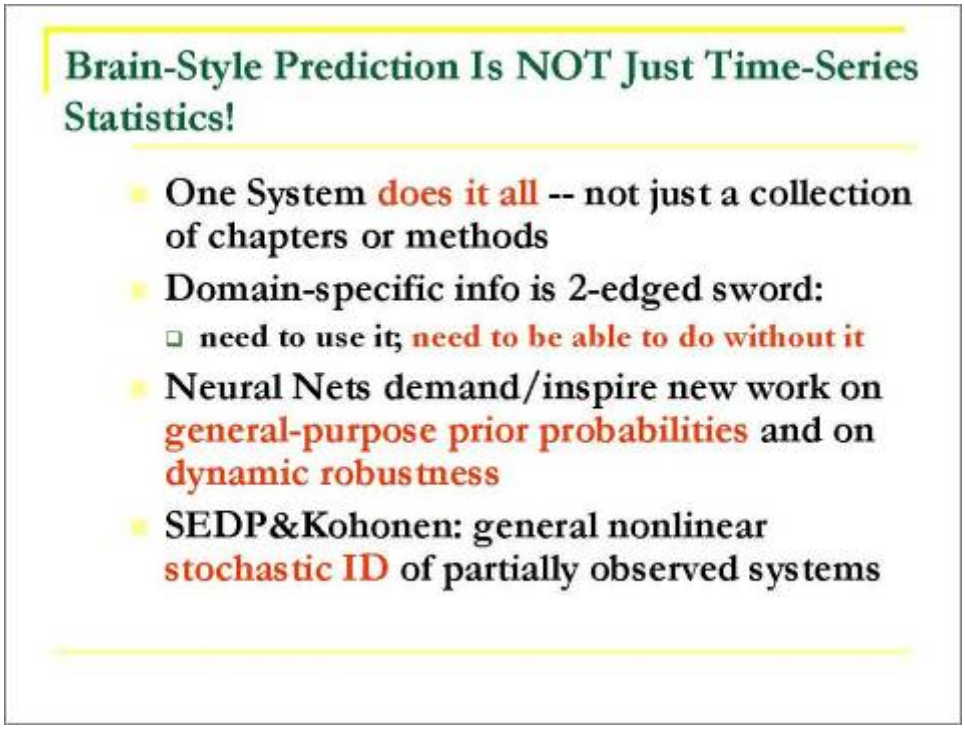
The electric power grid provides a beautiful testbed or example of a complex problem which requires us to exploit a more general type of symmetry. (The cellular SRN is a special case of this.)

Many classical AI people would argue that the maze problem and the electric power grid are systems which simply prove that a general-purpose learning approach is impossible. Some would even argue that general-purpose learning is impossible. But we know from biology that this is wrong. Some organisms may be born with specialized units like edge-detectors hard-wired into their brains; however, when those areas are damaged, even in organisms like frogs, researchers like Lettvin have proven that other neurons (in areas parallel to our "language" regions of the brain) can LEARN to be edge-detectors.

The real challenge, then, is how to assume as little as possible apriori, so that we can LEARN all the rest. How to do this?

In the book on complexity edited by Karny et al, I suggested one crude possibility. Perhaps the notion of a network of objects and relations would ITSELF be powerful enough to achieve brain-like capability, if embedded in a powerful enough framework. Certainly the electric power grid is an example of a system which can naturally be described as a network of objects and relations, and variables attached to objects. There are many other such systems, like communications grids, transportation networks, and so on. AI has certainly used relational nets.

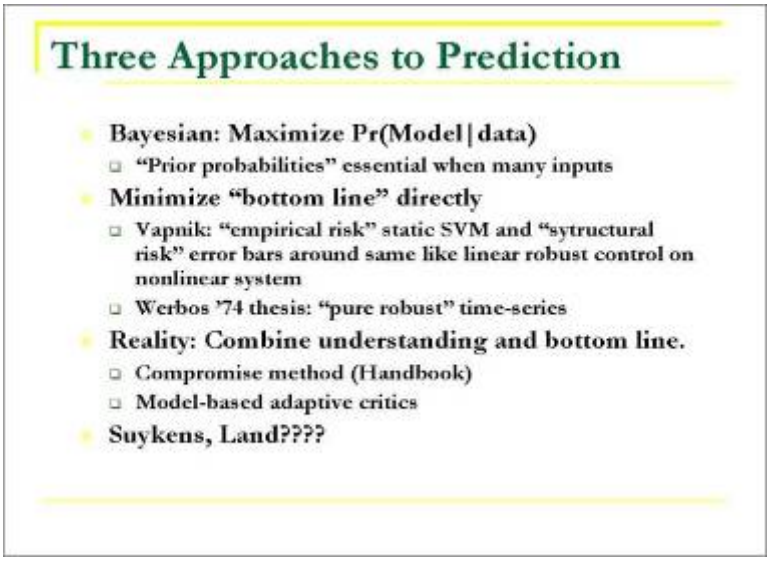
There are many difficult research questions which we need to explore here. But there are also many near-term research opportunities and applications involving those special cases where we already know what the objects and connections are, in the grid, and our goal is to learn how to make predictions for that grid. For example -given a training set made up of "input vectors" which describe the state of DIFFERENT electric power grids, with different topologies, how do we train a neural network to predict the next fault? (Or to approximate "J" over the entire grid?") No traditional neural net could be trained on such an irregular training set. But there is a new kind which can. (See later slides) I have said enough for now about the "ladder" of neural network structures for supervised learning. To summarize – there are simple designs which work well enough on simple tasks, and more powerful designs which work on the simple tasks but also on more general kinds of tasks.



Some statisticians have argued that "neural networks are just a branch of statistics." Textbooks are sometimes written which contain dozens of methods which are essentially just a choice of canned topologies. The user is sometimes urged to remember the "law of no free lunch;" the idea is that different methods work on different problems, that there is no universally better method, and that serious people always operate by studying the specific domain problem, picking a chapter, and developing expertise relevant only to that narrow domain. General purpose competence is impossible. Thus to build robots for space application, we may identify 2,000 standard tasks, and propose that NASA fund 2,000 back-to-back 3 years studies to develop specialized methods for each of the tasks. Some theorists call this "practical."

The "law of no free lunch" (as normally interpreted) is only HALF true. It is true that statisticians have the right to complain when a person tries out tweak number 166 on one narrow test problem, and draws universal conclusions from one example. But is possible, for example, for the domain of competency of one topology to be subset of another, more universal topology; that's what the Turing theorem is all about, and the capabilities of Object Nets are closely related to Turing concepts.

But in the end, as I described before, we cannot escape from the issue of analyzing the prior assumptions like smoothness, symmetry and robustness which make it possible to develop universal brain-like capabilities. That research challenge is at the core of the neural network field, and we have already learned a lot that goes well beyond what classical maximum likelihood statistics can do.

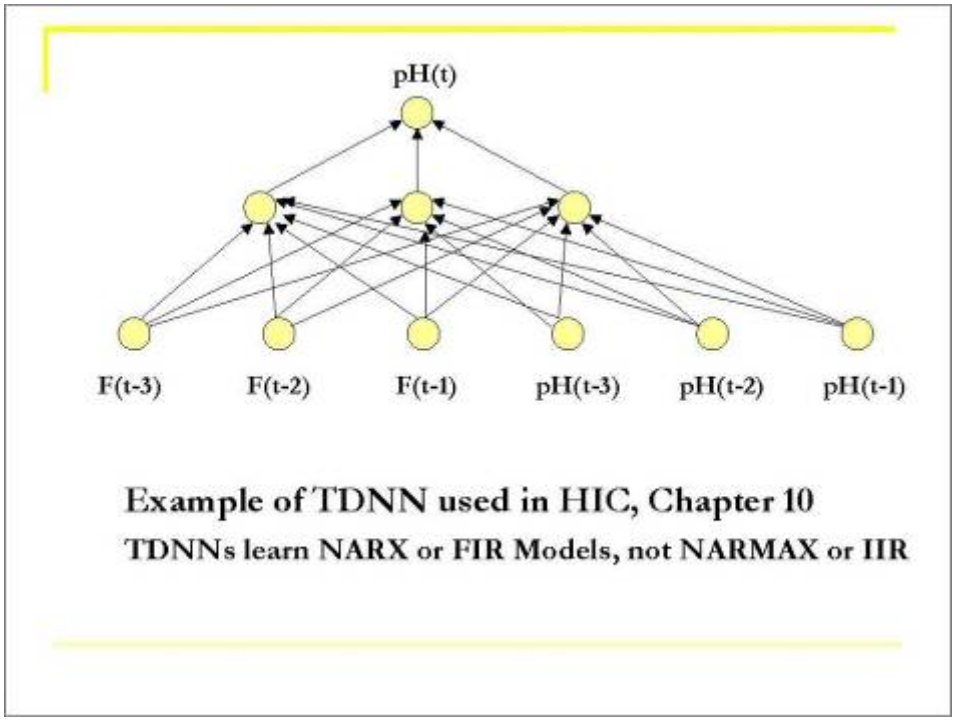


What does all of this theory tell us about practical issues, like how to train an MLP or linear classifier for best performance?

In most ANN work today, people train MLPs to do supervised learning, based on a very simple concept: they train the weights so as to minimize the square error of the predictions over the training set. The equations for doing that, using backpropagation, are widely published in simple texts (and my book). The capability is available in many canned packages as well. Statisticians have rightly pointed out that this is basically an extension or example of the concept of "nonlinear regression," known for decades.

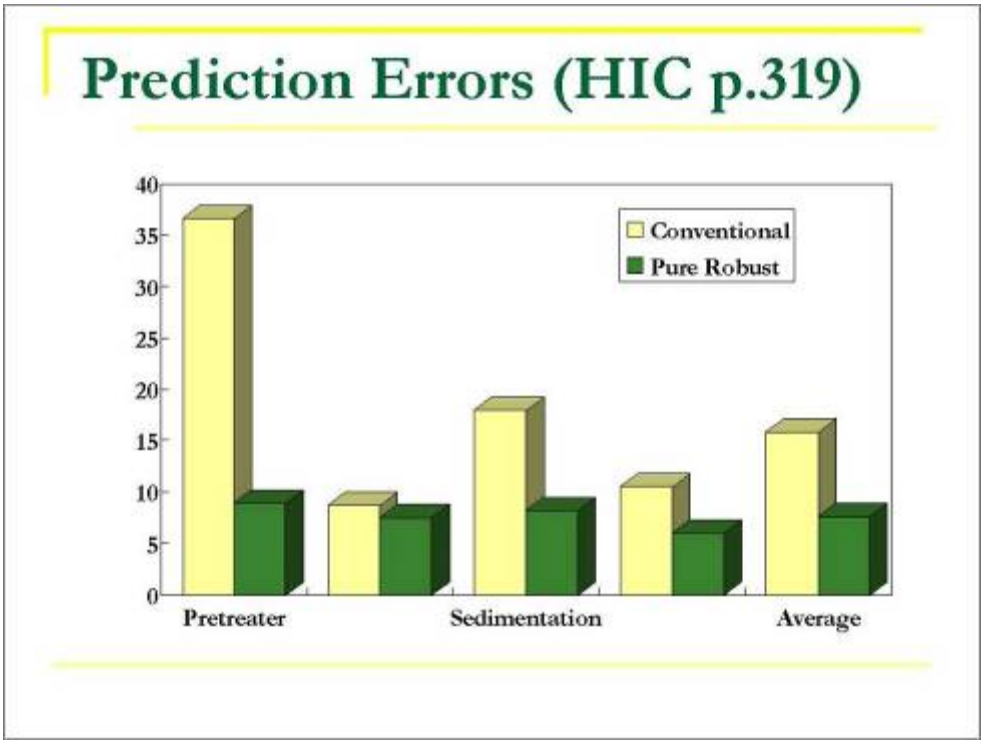
But practical statisticians have come to accept that minimizing square error isn't always the best way to go when there are many, many input variables. Instead of minimizing square error, many minimize a weighted sum of prediction errors plus the size of the weight vector. This is called "ridge regression" in the linear case. In 1987, I proposed that we can apply this to what we minimize in training MLPs. (This is different from an earlier suggestion by Hinton that we use an equivalent term on a temporary basis, called "weight decay," in order to speed up convergence to the weights which minimize square error.) But more sophisticated "penalty" terms have been developed in the meantime. For example, Phatak of UMBC has developed penalty terms which prevent statistical problems WITHOUT reducing the number of neurons or connections; this allows a kind of redundancy and fault-tolerance closer to what we actually see in the brain. More research is needed, of course and many approaches need to be pulled together.

Recently, Vapnik, "Father of the SVM," has elicited great excitement with the idea of minimizing losses directly, instead of using Bayesian concepts grounded in the idea of "probability of truth." I used an extension of that general approach to the time-series case to improve political forecasts in my 1974 thesis – but found later that we really need a kind of compromise for the general case.



In fact, in 1990, I joined with a famous chemical engineer, Prof. Tom McAvoy of UMCP, to use the same statistical concepts I used in my thesis, in order to improve the training of a simple TDNN to predict data form simulated and real chemical plants. Notice that there is no advanced topology here, and no penalty functions.

The example published in HIC begins with a review of neural networks for the chemical industry, and describes some of the application tests in detail. But after that part, it describes the ladder of more advanced methods in far more detail than I can here today.



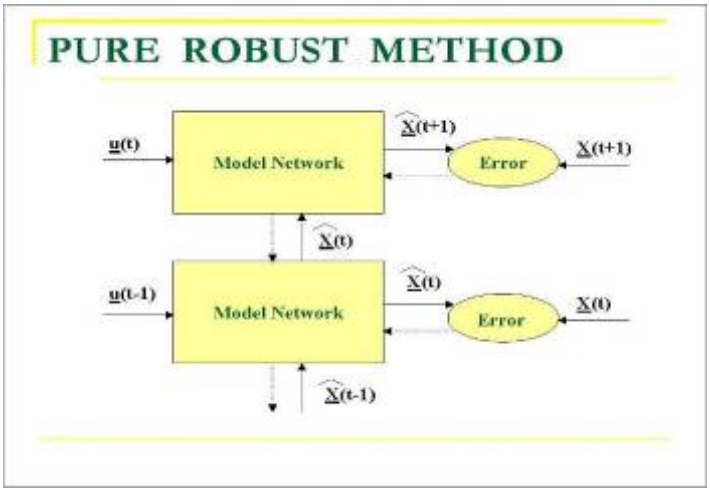
At the time of publication of HIC, we had done four test studies on simulated and real data, including a proprietary mulation of a polymer plant and real data from a wastewater treatment plant.

This is a case where improved prediction can translate directly into improved control, and a factor of two reductions in pollution (waste).

On average, when we trained the TDNN using the "pure robust" method of training, we cut errors by two-thirds, in the kind of multiperiod backcast testing from the training set to the test set that really represent the value of the predictor to the application.

Subsequent 5 studies by Ted Su, a PhD student of Tom's, strengthened the conclusion here. There was one interesting technical aspect. Training tended to be easier, but the benefits smaller, for simulated plants. Training was harder, but far more valuable, for predicting real plants. Tricks like using good initial values and "shaping" turned out to be critical to the training.

The key idea in shaping is to train one network to perform an easier task – and then use those weights as the INITIAL VALUES in training a network to perform a harder task. The new networks could have the same structure as the older one, or it could have some (zero) new connections added. Shaping can be done in stages, task by task, step by step. In fact – even humans need to use step-by-step learning, in order to learn very complex tasks.

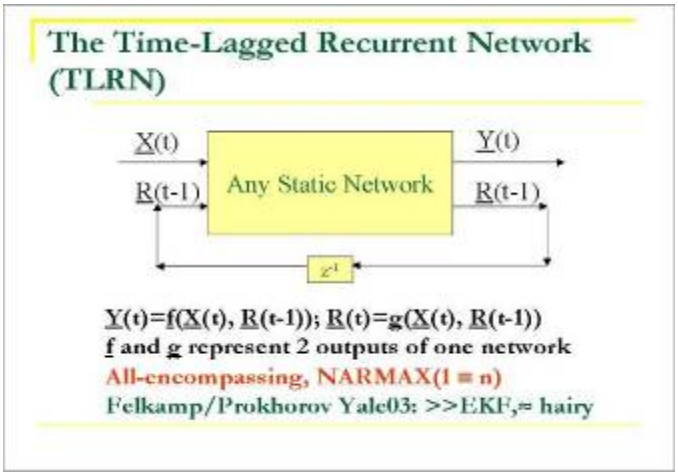


This slide basically describes what the pure robust method is like. Some people in control theory would say “this is just parallel training” – but it is drastically different in practice. (See HIC for a discussion of why.)

In ordinary statistical training, one minimizes the square error between the observed X(t+1) and the prediction for X(t+1) which results when the ACTUAL, OBSERVED values for X(t) and u(t) are plugged into the model. But in the pure robust method, for any starting X(1), one builds up an entire chain of predictions, by plugging in the PREDICTION for X(t) instead of the actual value. One minimizes the square error in this multiperiod approach. In fact, a number of very competent engineers have quietly called this “the multiperiod method which we all know about” – without referring back to the original theory, which warns about its limitations.

In 1977, I published a paper in the IEEE SMC Transactions, applying this method to the prediction of social trends (conflict and economics) in Latin America. The method failed very badly, for theoretical reasons I discussed beforehand. The pure robust method – the method of “just minimizing the bottom line,” is not a good universal approach. The theoretical reasons(discussed in HIC) are a bit subtle for today; crudely, I would say that reality involves a COMBINATION of hidden order (what forces the robust approach) and hidden mixing or noise (which forces stochastic approaches); only by finding a method which unifies the two can we have a universal system, or a system able to handle plants which combine both aspects. The 1977 paper demonstrated a “compromise method” to do that – a special penalty function, in effect – which was able to cut in half the errors in predicting GNP in Latin America. It worked better both short-term and long-term. HIC discusses the neural net version and possible extensions – and the need for further research to do even better.

The pure robust and compromise methods both require the use of true backpropagation through time (BPTT), to work. My original definition of BPTT as a specific exact, low-cost closed method was unambiguous; however, some psychologists have used the term as a kind of buzzword for inexact methods, leading to the false mpression that it is harder to train such recurrent systems than it really is. Ford and Principe, among others, have irly bulletproof implementations of BPTT available.

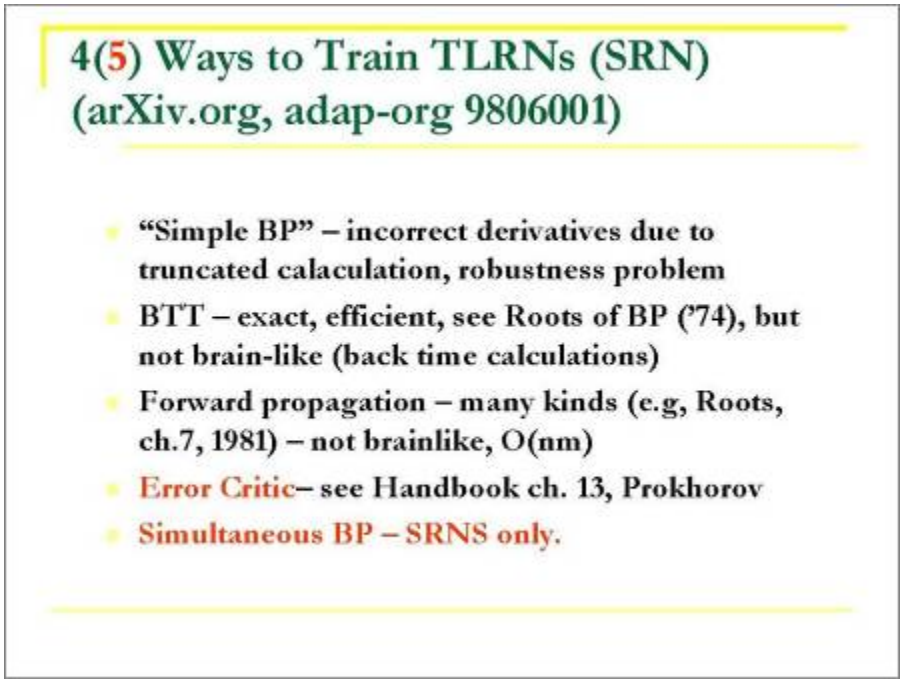


The TLRN is essentially The General Case for neural networks with "short term memory" or "working memory" in a certain sense. Everything else is a subset, a special case. In the extreme case – if an ObjectNet is used as the Static Network inside this diagram, one winds up with a kind of universal (deterministic) system, a universal approximator for time-series dynamic relations. (Sontag has in fact proved some universal approximation theorems for the dynamic case.) Note that this flow chart does NOT require that R(t-1) be fed back only to the "input layer" of the static network inside; it does not even require that the static network HAVE layers at all!

Some have misunderstood, saying "You are only inserting recurrence outside the network. Others insert it inside." But this is a matter of mathematical convention. The arrow refers to a flow of information, reflected in the equations. For example, I give examples of this using the "sticky neuron," in the book edited by Maren (1990), in which the time delay is from the "membrane" of a neuron to itself in the next time period, bypassing the tanh function. (Schmidhuber and Ford have presented examples of how this kind of placement may or may not be useful, in different applications. But the brain does appear to contain sticky neurons, in my view!) Chapter 8 of Roots gives the actual equations of a TLRN for the case of nonsticky neurons.

Others have said "TDNNs represent NARMA functions, which are the most general case and much easier to train." Unfortunately, some strands of control theory have misconstrued the original literature on autoregressive moving-average (ARMA) processes. Even neural network people should at least read the classic text by Box and Jenkins (the multivariate case of which is discussed in chapter 3 of Roots), which explains in clear practical terms why the TDNN (ARX) kind of design is NOT universal. In fact, there are many other reasons. Linear ARMA models are a special case of TLRNs.

The TLRN (with MLPs inside) is the crucial workhorse for all the most impressive real-world applications of ANNs today, for many reasons. HLADP explains some of the reasons. For example, Feldkamp et a give examples where simple TLRNs dramatically outperform extended Kalman filters in nonlinear state estimation, and do as well as far more expensive less brain-like particle filter designs now growing in popularity. I will say more in later slides.

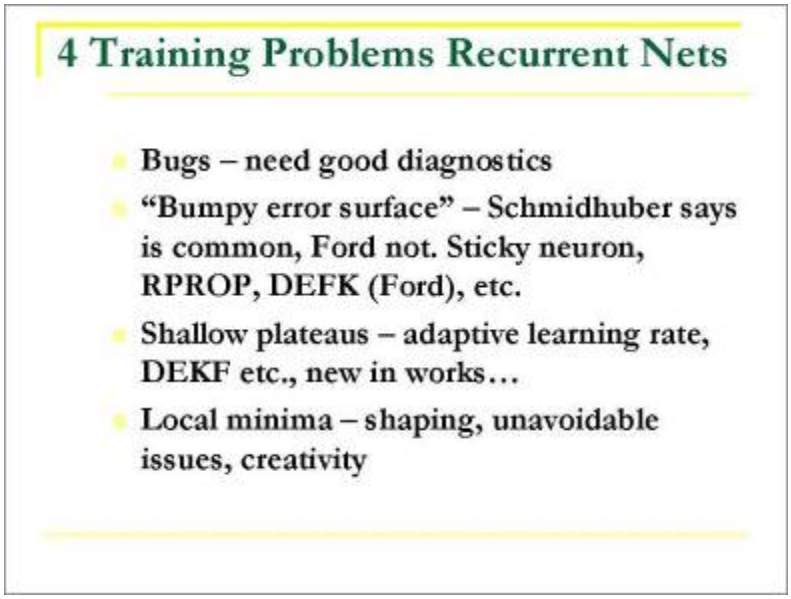


This slide summarizes several ways to calculate the derivatives needed for any derivative-based method to train TLRNs or SRNs or hybrids. See the web page for details.

In this tutorial, I have given a kind of top down view of what a TLRN is – the general case. For a more concrete bottom-up view of the same thing, see the book by Principe and Euliano, and the software at www.nd.com. That software provides a kind of "tinker toy" kit for building up all kinds of examples of TLRN. (Of course, it can also build TDNNs and MLPs, which are a special case.)

In other words – there is a GENERAL TLRN structure. To define the specific TLRN which does what you want, you can simply look for the right weights. In that general structure; you can set of them to zero, which is the same as "cutting the connection." But you can reach the same set of possibilities in a bottom-up way, by starting from zero connections and ADDING neurons one at a time.

It is important to understand that both viewpoints are equivalent. It is also important to understand that the growing, pruning and weight adjustment can all be automated. The easiest way, conceptually, is to start from the general structure, and use automatic learning systems to adjust the weights.

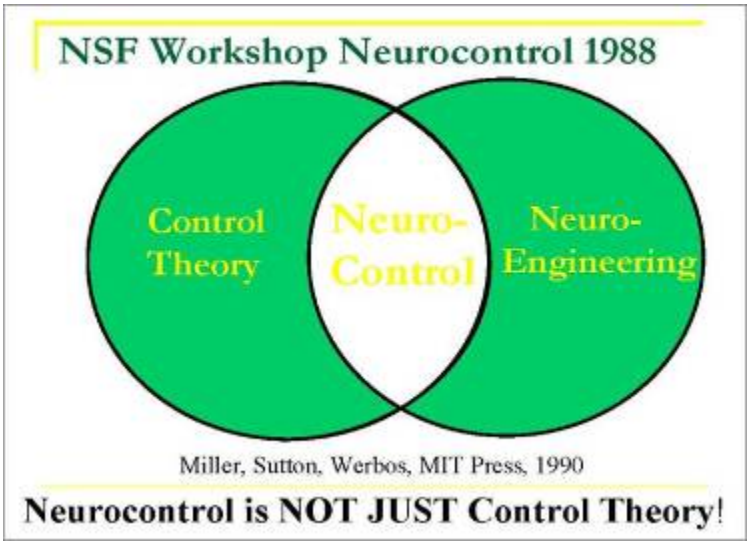


At least four kinds of problems explain why some people give up prematurely When trying to use recurrent networks. This is a tragedy, because recurrence is usually essential to the kind of power one can get from neural networks.

Bugs may actually be the biggest problem. It's easy to say "I never do bugs," when it's not true. It's critical in the early stages to have sensible debugging – for example, to use reductions in error as predicted by new versus old gradient and weight changes, versus actual error changes, versus learning rates, to get a handle on where problems are coming from, in the early stages of software development (with an option to turn on again). Many "plateau problems" (low slope) have been misdiagnosed as"local minima." With plateau problems, better ways of using derivatives 9as discussed above) are generally needed.

Local minima are important but are widely misunderstood. Anyone who claims a universal infallible easy magic wand for local minima is misconstruing reality. With MLPs and simple systems, local minima are far rarer and far less harmful than often believed. (Typically it is useful to IMPROVE performance, and not strictly necessary to guarantee that nothing better is possible!) There is a kind of Murphy's Law at work – the more precise the training method, the sharper the error surface, and the greater the danger of local minima; thus the number one practical approach to local minima is "shaping" – training weights on an easy problem, using the results as initial values on a harder problem, and so on. Even for training humans, I would argue that this practical approach is "the right way."

The more complex the task, the more unavoidable are local minima. Thus in living human life, we are ALL stuck in local minima. That simply means we can do better, if we exercise more creativity. Symbolic reasoning has helped humans get out of ruts that other mammals have remained deeply stuck in; once again, I would agree with AI that symbolic reasoning offers a higher level of intelligence – when we have the foundations built to do it right.



How can we achieve brain-like intelligent control by designing learning systems made up of simple processing units like neurons?

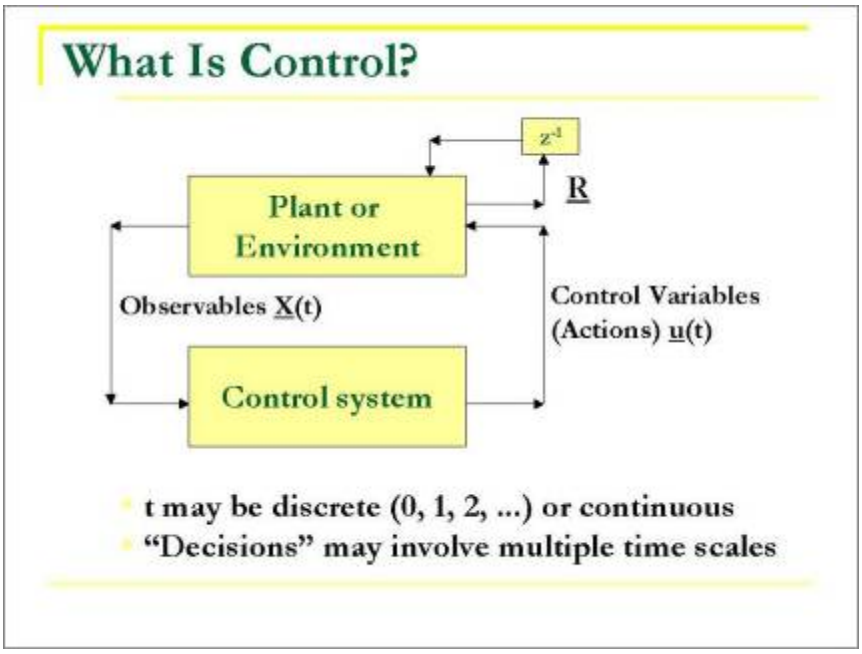
Whatever the brain does – it does do "control" (i.e.deciding on action), and it is made up of neurons. It is a neurocontroller. Thus to achieve brain-like capabilities, we need to build up the science of neurocontrol as illustrated here.

Neurocontrol is a subset of ANN design, since it does built up systems out of ANNs. But it is also a subset of control theory, since control theory addresses the general problem of all kinds of systems designed to output useful streams of control decisions over time. For best results, we clearly need to draw on BOTH disciplines, as much as possible. This is why my first action at NSF was to contact Tom Miller and Richard Sutton to organize the first workshop on neurocontrol, to try to bring these disciplines together and create a new area. The book from MIT Press was widely influential, and did inaugurate this new field.

But there have been a lot of misunderstandings and lessons learned in later years.

First, many people outside of engineering have a paranoid misunderstanding of what control theory is all about. "Are you turning the brain into a slave that you control, denying it all autonomy? These are not learning systems, they are just controlled systems.." This is a gross misunderstanding. We are developing general purpose learning systems which THEMSELVES can control engineering plants, for example. Neurocontrol designs vary in regards to how much autonomy they have, but our ladder rises up to as much autonomy as a human brain possesses.

Second, it is not necessary that everyone in this area possess a PhD both in control theory and in neural networks. There are few people like that. But it is critical that we do try to learn a few core ideas from each other. I hope that this tutorial will help. Very few people know enough about both fields to do the very best that ca be done in real-world applications; however, we all can learn a step or two more, and do that much better. I do strongly hope for more unified curricula in the future that will help overcome these gaps. After 1990, some people decided to use the new word "neurocontrol" for different things…

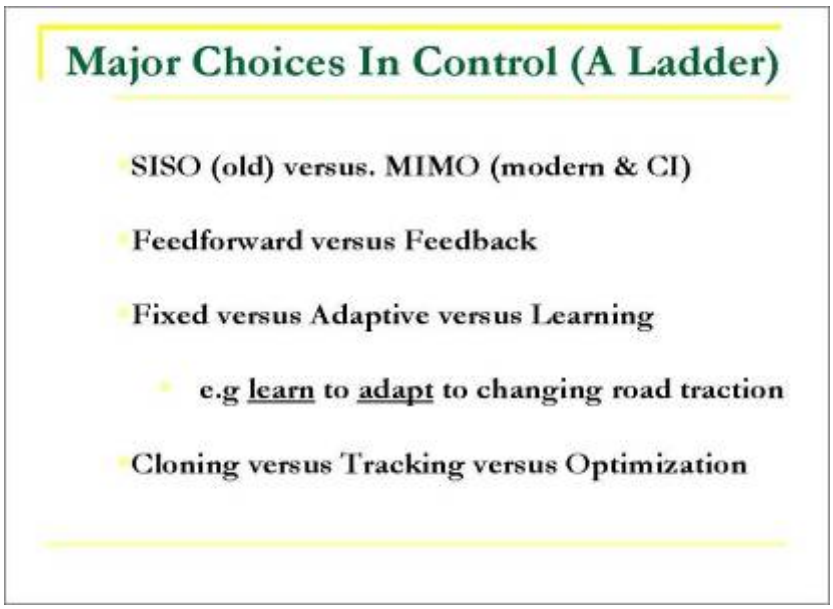


This slide summarizes what control theory "is." Some people in AI have told me that I should not call it "control theory," after I explain that the field is not a narrow effort to control or enslave people; however, I do not have the power to change this standard terminology! Still… I should note that the premiere theoretical conference in that field is the IEEE Conference on Decision and Control (CDC); it really is decision AND control.

Control theory is basically just the principled design of control systems, systems which are supposed to send out "control signals" (the vector u) at each period of time, in order to improve the state of some kind of external environment or "plant." (Like a manufacturing plant.) Throughout control theory, it is understood that the vector we observe (X) is only a small window into the state of the larger external reality or state vector, which I show here as R.

My choice of letters here is a little different from what they generally use – because we in the ANN field have to try to unify ideas from many different areas, where different ideas appear, and also because it helps to use letters that stand for something. For example, "R" stands for "reality" or'"representation" or "recurrent" – which end up being the same thing here very often!!!

Of course, this is a very broad and inclusive concept here. There are many different types of control system. For example, there are different ideas for what we want to do to the external environment; those lead to different types of control design.



I don't have time to say all that can be said about this. It is said that no PhD program in control theory in the US guarantees that all its graduates have a background broad enough to embrace all the aspects of control theory shown here! (Thus few people in the US can legitimately claim to "know control theory" in an inclusive sense. It is said that Sweden tries harder.)

The last two bullets are most essential to real-world applications of neural nets. Pure mathematicians sometimes have problems with "learning" versus "adaptation," but this distinction is essential to success in real-world applications. Learning is cumulative and basically universal; adaptation means changing to adapt to changing conditions. You can see the difference when a new driver first encounters a patch of ice; if he/she tries to learn how to survive – often the learning is just too slow, and he/she hits a tree. But as he/she learns to drive better, over a variety of conditions, he/she learns to adapt quickly – by habit or reflex – to slippery feelings on the road. Adaptive control systems (either classical or neural) tend to be very slow in adapting to changing conditions, leading to "bad transients" (high waste, pollution, an explosion on occasion). They do so because their designs essentially try to use learning methods to adapt key parameters. But if TLRNs are trained on a variety of possible conditions, covering the range of likely types of divergence, the recurrent nodes can implicitly detect the changing conditions, and yield fast adaptation; this is why the Ford diagnostic systems can adapt easily and automatically to varying engine types, without a need to retrain the neural network! In fact changing conditions and parameters and other unobserved variables are crucial in most real-world applications.

In control – including neurocontrol or AI control – there are only three relevant families of basic design types: cloning designs (to imitate or model a human), tracking designs (to make something follow a path ordered "from above"), and designs for optimizing a performance or utility measure.

This slide elaborates on the previous trichotomy. There are two versions of cloning – copying what a person says (as in traditional AI or fuzzy control), or copying what they do. For copying what they do, effectively, it is essential to think of this as predictive modeling of what the human would do. Thus, as in any dynamic prediction, use of a TLRN structure is essential to max capability. Cloning can be used to generate initial weights for an optimizing controller.

Many classical controllers focus on tracking tasks, without analyzing how to maximize performance. Many will say that the folks who fund them don't care about performance. Many people who fund them say they are looking for new support services. When tracking is the only consideration, the only real challenge left is to try to stabilize a plant; however, it turns out that optimization can give often greater stability than more direct, brute force tracking methods! I will discuss that more.

People have claimed at times to develop hundreds of optimization methods… but all of them boil down to the three core methods noted here. There is LQG for linear plants, of some historic importance and a useful starting point for some of us. For the nonlinear case, there are a host of methods for considering an action plan, looking ahead N steps to see how well it does, and then updating the plan over and over again to get better performance in explicit N-ahead lookup.

And then there is learning and approximate or adaptive dynamic programming (ADP) which, in my view, is the only family of methods even remotely plausible as a model of how the brain does it. Still, explicit N-period lookaheads and ADP both provide great capacity; they are the "top of the tree"(or of the first staircase?) in developing more powerful neurocontrol capabilities.

Cloning has had many amazing applications important to the aerospace sector. I usually have time to discuss how Accurate Automation developed a robust nonlinear controller for the model National Aerospace Plane in only two weeks using cloning, after others had spent many millions. But the Atkeson and Schaal example I mentioned, with robots, may be more critical to some new directions we are now pursuing.

One potential testbed area for cloning followed by optimization design is depicted in this slide. NASA, NSF and the Electric Power Research Institute (EPRI) have join together in 2002 to support a new research thrust related to Space Solar Power (SSP).For more information, see the NSF web page and the URLs which appear in publication NSF 02-098.

This initiative was partly an outgrowth of an earlier workshop with NASA, which explored the possible role of computational intelligence in making SSP a viable approach to the world's primary energy problems. Many of the old roadblocks to SSP have been solved, in principle, or are now being solved. Possibly the number one roadblock, in terms of underlying technology, is the need to better automate the assembly of large structures in space, using large numbers of semi-autonomous robots. This will require breakthrough capabilities in robotics, beyond what conventional robotics has given us. A few advanced people in robotics, like Fukuda in Japan and Hirzinger in Germany, have exploited computational intelligence to achieve real-world results far beyond the mainstream of what is available in the US. Yet we could do even better, if we could build on their success and use the more powerful types of learning algorithm discussed here.

Some of Hirzinger's work is reviewed in the article Neurocontrollers in the Encyclopedia of Electronics and Electrical Engineering.
Some of the new possibilities here are mentioned in my paper on energy in the State of the Future 2004 CDROM (www.stateofthefuture.org), and new NSF-NASA partnerships seem promising.

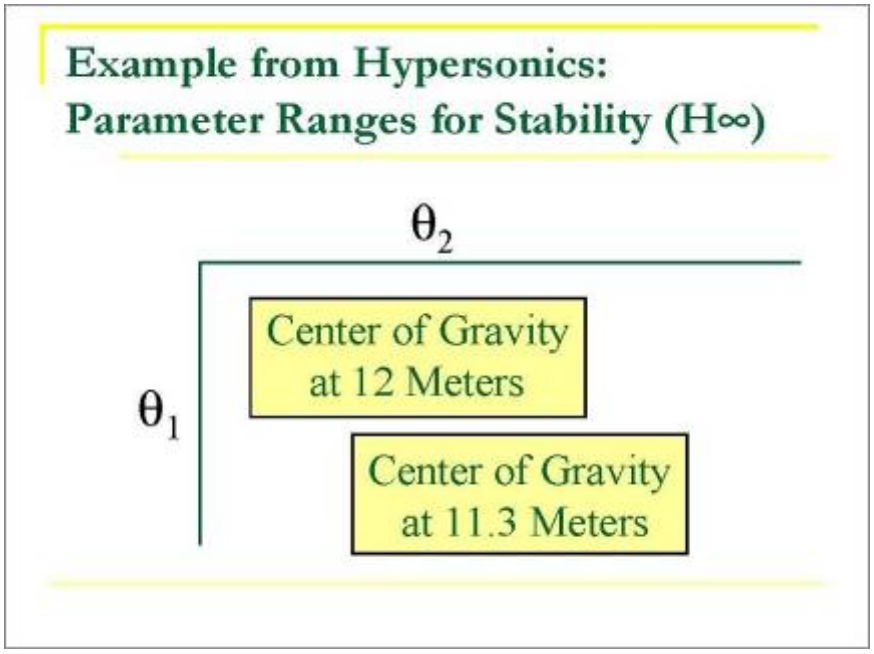
There are three major pillars in "modern control theory" – and few people who really know all about them all. Neurocontrol people should at least learn the big picture of what they are, however.

The big three are: (1) optimal control (including LQG and dynamic programming and N-period ahead optimization, discussed in classic texts by Bryson and Ho or Stengel); (2) adaptive control; (3) robust control. I have already stressed optimal control as an approach, but many people rely more on robust or adaptive control, which focus on tracking problems.

The distinction between adaptive and robust control is actually like a famous versus from the Chinese classic, the I Ching. They ask: "Which is stronger, the oak tree or the grass? The oak tree looks so much stronger – massive an unmoving. The grass looks so weak. But when the typhoon comes, it is the oak tree which falls." Robust control, like the oak tree, tries to find control designs AND parameter values which stay fixed in application, and are able to keep a plant stable under a wide range of external conditions. Adaptive designs try to change the parameters as conditions change.

Yet engineering today does not seem to follow the I Ching. The adaptive control designs used today are famous for being much less stable and reliable than the robust control designs. How can that be?

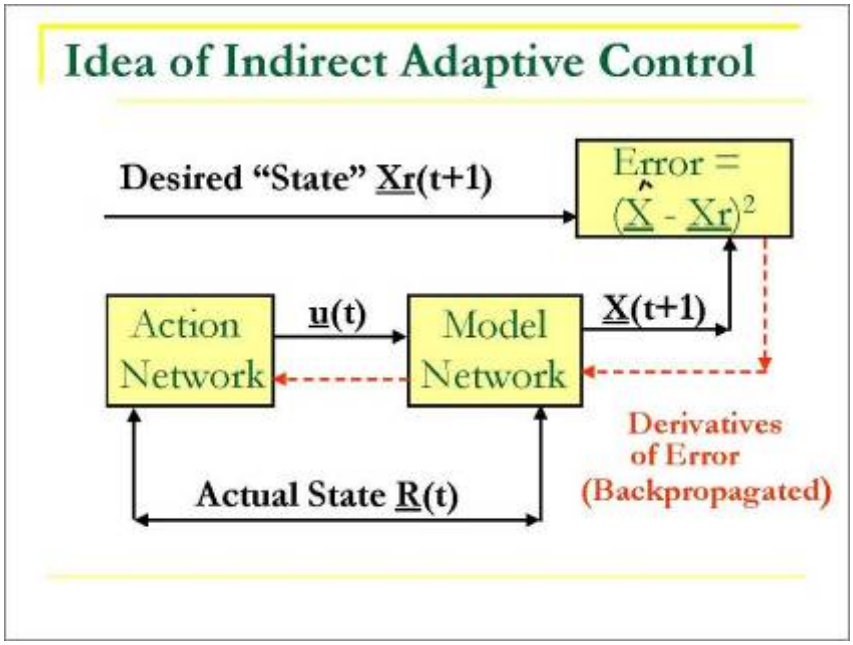
The usual form of robust control popular today are linear feedforward methods, easily constructed in MatLab. But there is a third way to solve tracking problems, where neural networks
(e.g. ala Ford but with ADP) and advanced, nonlinear robust control converge to the same design. It is learning offline to be adaptive online. Or the ADP version could be called "using neural nets to provide the most accurate available approximation solution to the HJB of the nonlinear robust control problem with feedback loops." This is the stablest method, and it allows integration of performance and stability together as goals.



This is an example of why today's ordinary best linear multi-input multi-output robust control is not so universally good enough as sometimes claimed. It is taken from a world-class paper on the robust control of a model hypersonic aircraft, from the AIAA conference in Norfolk on that topic.

The graph speaks for itself. With two control parameters, there is a range of values which can stabilize the craft if the center of gravity is at 11.3 meters. A different value if it is at 12 meters.

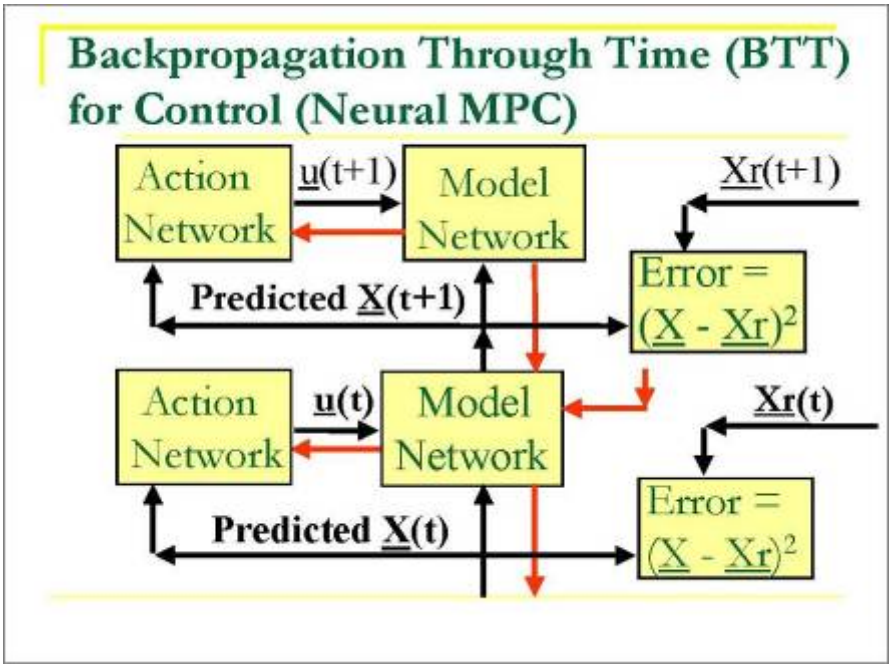
One might ask: "Is this an airplane or is this canoe? Walk two feet and it keels over…" The authors may recommend, reasonably, that there should be tight effort to control the center of gravity. But is that really the best we can do? Common sense would agree with the I Ching here. Somehow, it ought to be possible to achieve more stability by somehow sensing where the center of gravity is, and adapting to its changes. Why doesn't it seem to work out that way in practice?



The reason why adaptive control doesn't beat out linear robust control here can be seen easily by looking at the mathematics (flow chart) of how we actually DO adaptive control today. The idea of being adaptive is not the problem; the problem lies in HOW the adaptation is done today.

This slide really specifies exactly how the best form of adaptive control is implemented today with neural networks. Basically, the weights in the action network ("controller") are trained by derivatives of tracking error backpropagated through the trained model of the plant. In other words, the controller is trained to minimize tracking error AT TIME t+1. It only looks one period of time ahead. It is myopic. It has been well-known since Wiener and before that myopic control can easily lead to instability – especially when the actions we need to get us back on course ten times ahead lead to no effect or temporary negative effects one time ahead. These problems have fancy names (deadtimes; nonminimum phase; sign reversals or indeterminism), but they are reasonably straightforward – and extremely common.

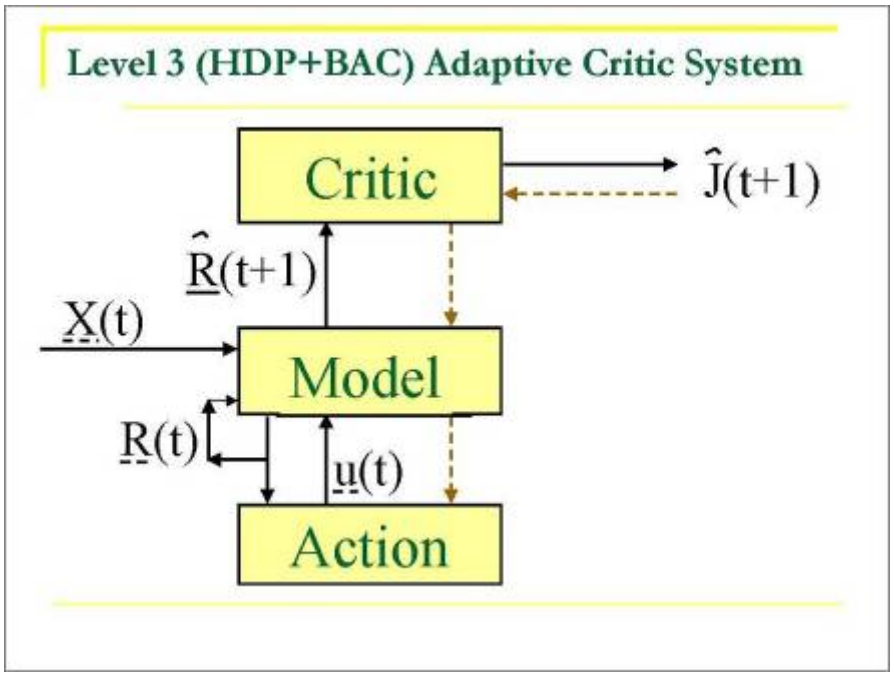
The design here does sometimes work, especially in the hands of those who cultivate the art of tweaking it. But I have many times seen people try it out, uncover vast explosions or transient problems, and then say "neural nets don't work" – when all they had to do was add one arrow in the flow chart to solve the problem and beat out everything else they have! They often feel compelled to use this design because of all the many stability profs proven for classical and neural adaptive control – but those proofs all assume conditions about the environment which are very strong and rarely met. There are two easy ways to get much better performance and stability both.



This is the easiest of the two upgrades. It needs only the same kinds of nets as the previous design –but now there is an explicit N-period ahead lookahead, and a minimizatoin of error OVER TIME.In the Miller, Sutton and Werbos book, we noted four implementations of this kind of design by that time. One, a chapter in the book, was Widrow's truck-backer upper, a truly remarkable system which was once very widely publicized – and deserves to be remembered more than it has been. The agility of that system in nonminimum phase maneuvers, with a fairly large physical truck, competing against a human truck driver… was very impressive. And it was all due to a use of BTT.

Pseudocode for BTT as a control method can be found in chapter 8 of Roots. It should be noted, however, that use of TLRNs in this design is crucial to the kind of capabilities that Ford has been able to achieve here. A thorough understanding of BTT is essential. Portions of the Ford software are said to be posted soon at the web page of Prof. George Lendaris of Portland State (though I haven't checked). Suykens et al (Springer) have a book Neurocontrol which proves that the stability guarantees for thiskind of design are far stronger and more universal than the guarantees for indirect adaptive control.

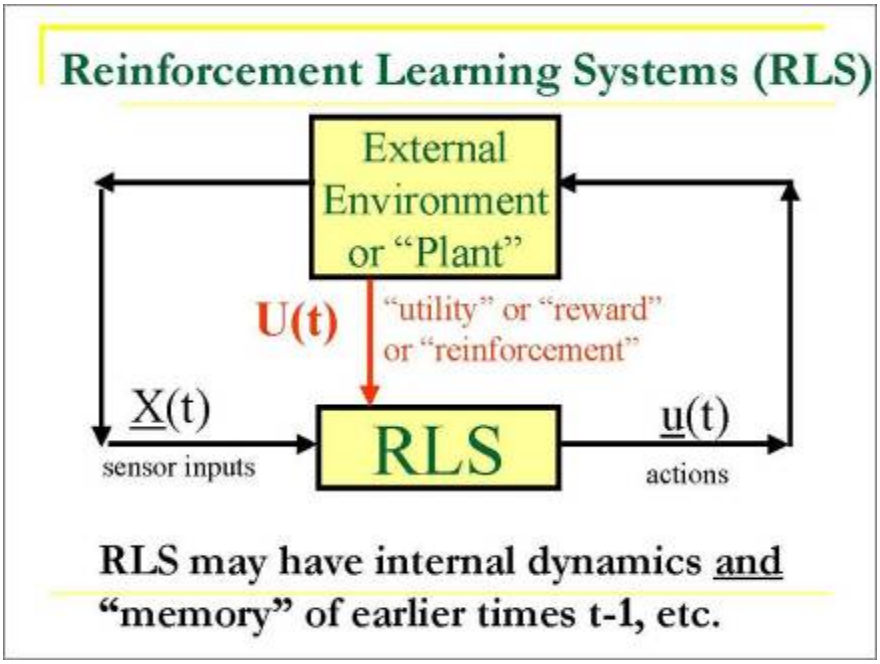
In principle, this design gives the right answer only for deterministic plants. However, I have pointed out for many years (e.g. ICNN 88!) that the method is easily extended to incorporate those features of "Differential Dynamic Programming" which address the stochastic case. BTT control has turned out to be unusually robust in practice, and still hard to beat. In the long-term, however, theory tells us that ADP is the more powerful method for the more general case, and a few relevant examples have begun to clarify some of the issues.



This is an alternative way to try to "add stability" to indirect adaptive control. Here we use the SAME design as IAC, except that we replace the previous ad hoc arbitrary measure of tracking error with a trained critic network. Control theory tells us that SOME choice of the Critic (a "Liapunov function") should be able to stabilize the plant, for ANY controllable plant, not just the limited kinds that standard adaptive control can handle. This can even be viewed as an extension of adaptive control Back in 1994, I proposed that we could constructively find a Liapunov function, by using ADP methods. In the ensuing years, control Liapunov functions have become kind of growth industry, but most of the papers today do not address the general case.

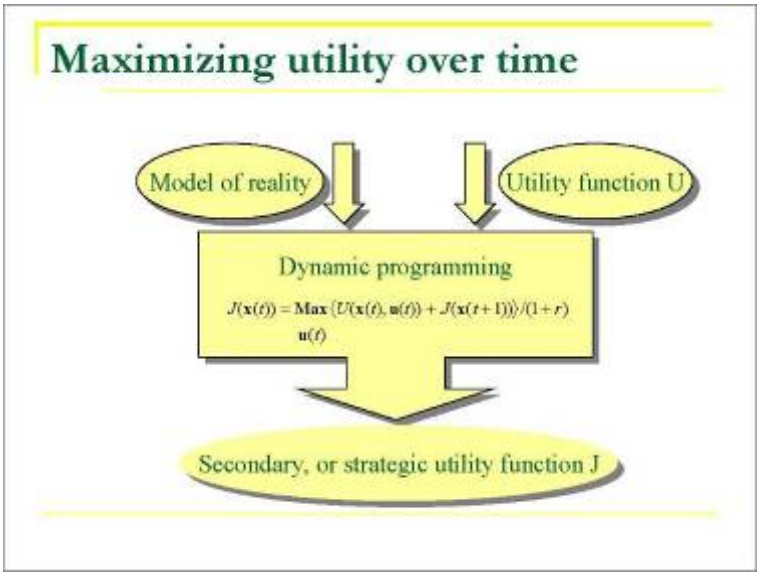
In 1998, I showed how proper ways of training the critic will ALWAYS converge to the right "J" function, which happens to be a Liapunov function, for any controllable linear multi-input multi-output plant. This strongly suggests that these new training methods, combined with the design in this flow chart, yield the "holy grail" of classical adaptive control – universal stability in a truly adaptive design. See arXiv.org, adap-org 9810001 for details.

This also strongly suggests that the nonlinear versions of these new training methods could be useful in applications where rock-hard stability is desired. But for today's applications, offline training using simpler methods may be the best choice; once these methods converge, we get a stability guarantee after the fact.



I have said enough for now about tracking problems and brute-force look-ahead. Now I will move ahead to the ADP class of control designs, which some people call "simple reinforcement learning." In fact, it is not quite so simple as many imagine, especially when you try to get the full performance capability out of them.

People in AI and neural networks have tried to build "reinforcement learning systems" for decades. The term has become very fuzzy over the years. But this slide shows the basic idea in the modern version. Supervised learning systems are told exactly what their output is supposed to be. They are given a "desired output" for use in training. But RLS are not told what to output. They are given a measure of "how good their output is," and they try to learn how to make their output better. At least, that's the older way of thinking about it. At each time t, they receive a signal, U (t), which tells them how good their output u(t) was (for the current situation of the plant). But in the modern view, we think of this as a "dynamic problem." The goal is to pick an output u(t), at each time, in a way that increase the TOTAL utility, the sum of U(t)+U(t+1)+U(t+2)+… A true RLS should be a kind of universal system. At each time t, it is given sensor information, a vector X(t). It outputs actions u(t). Starting from no knowledge at all about the external environment or plant, it is supposed to LEARN how to generate action decisions which maximize U(T)+U(t+1)+…, the sum across all future time. As a practical matter, RLS designers usually imitate economists by saying that utility in more distant times should be "discounted" according to some kind of interest rate, r; this says that we want to maximize utility over all future time – but that time t is more important" or "worth more" that time t+1. We want to maximize a WEIGHTED sum of future utility. See HLADP (or HIC) for more details. Researchers from psychology often write r(t) instead of U(t), in order to show respect to B.F. Skinner's idea of reinforcement" or "reward" in training animals.



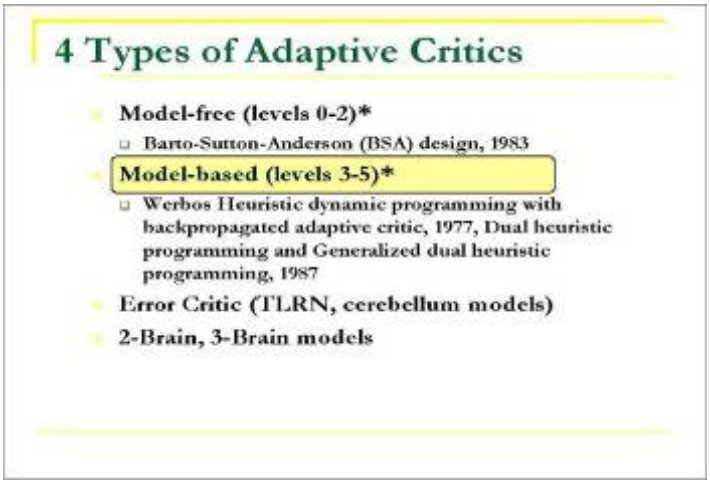
Back in the 1960's, many in AI had given up on ever developing a truly general purpose RLS system. Efforts to guess or hack or intuit a good design never got anywhere.

But in 1967, in "Elements of Intelligence," (Cybernetica, Namur, No. 3, 1968), I proposed that we try to use more fundamental mathematical principles to address this design problem. I proposed that we design RLS by trying to APPROXIMATE dynamic programming.

After all, when we try to maximize the sum of U(t) over future time, we are trying to solve an optimization problem. Dynamic programming is the ONLY exact and efficient method for solving this type of problem in the general case. The "general case" means that your environment or plant could be any nonlinear stochastic system.

This slide illustrates how dynamic programming (DP) works. The user specifies a utility function U(x); this function is simply a statement by the user of what he/she wants the control system to maximize. (It could be profit or throughput or a complex function like output minus cost minus energy use minus pollution minus wear and tear, minus tracking error. The user gets to decide what he/she wants the ultimate performance to be. The user also gets to decide whether he/she cares less about future times than the immediate present and, if so, what interest rate r fits his/her goals.) In classic DP, the user also specifies a stochastic model of how the plant works over time. These two pieces of information are then fed into the Bellman equation, illustrated in the middle box. The mathematician or engineer then tries to SOLVE the Bellman equation in this case; in other words he tries to find a function J(x) such that the Bellman equation is satisfied. The key theorem in dynamic programming is that such a J function exists almost all the time; also, the strategy of action which maximizes the expected value of future U over al future times is the same as the strategy which maximizes J(t+1). DP converts a hard problem in optimization over future times into a much easier problem in optimization one time period ahead.

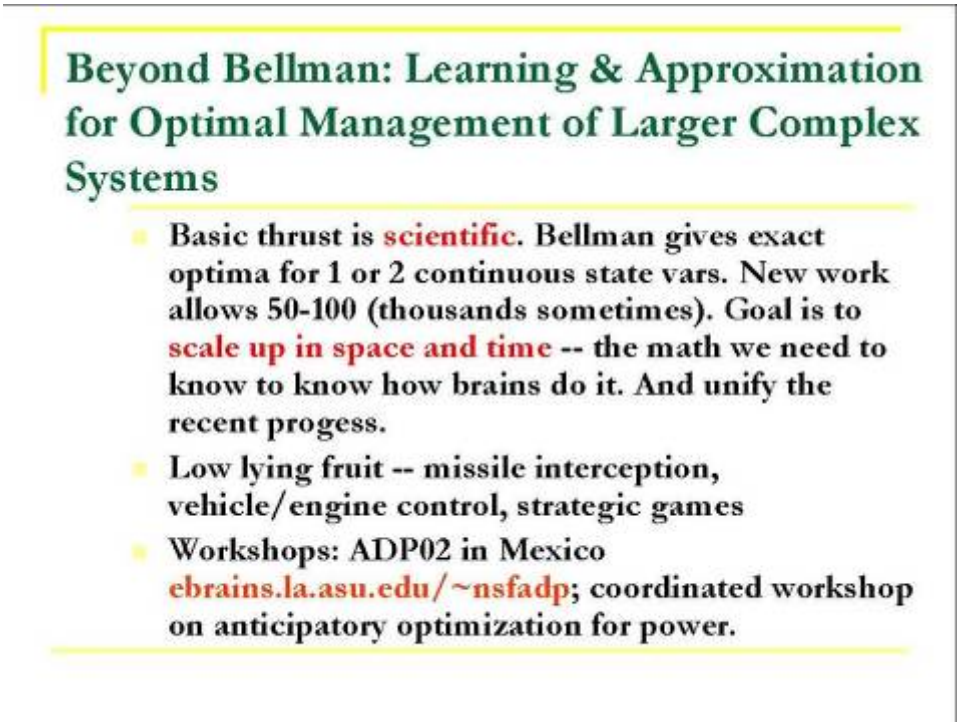
We cannot use DP proper on large problems in the real world, because we encounter a "curse of mensionality" in solving the equation. ADP, the learning-based approximation of DP, is the answer.



Modern approximate or adaptive dynamic programming (ADP) approximates DP by training or tuning a "Critic" – network or function approximator which tries to approximate the function J or something very similar to J. Sometimes we call this field "adaptive critics," to make it very clear that we are not talking about older psychologists' notion of reinforcement learning to train animals, or about optimization without a time aspect, or even about older approaches to approximating dynamic programming. The term "adaptive critic" is really the most precise term for specifying this new field of research. For now, however, I will call it "ADP."

There are many many ADP designs available. In a way, this sounds strange. After all, if two designs are trying to solve the same universal task, in a universal way, how could there be different methods? However, it is the same situation here as in supervised learning. There are very simple designs here based on lookup tables (like the original Barto-Sutton-Anderson design or like "Q learning") which are easy to use, widely proliferated – and mited in their power. Many AI people, familiar with only those methods, have gone on to assert even recently that "reinforcement learning can never solve problems large enough to be realistic." Likewise, some assert "no one has found a way to deal with continuous variables."
But in fact, there is a whole ladder of designs, of ever-increasing capability and complexity, which have in fact worked better than all previous control methods on a number of important real-world physical control challenges. Level 2 is mainly a model-free design which I proposed in 1989 (IEEE Proc. CDC) called action-dependent HDP, which has been reinvented under many other names (e.g. "policy-Q). It was very successful in 1990 in various applications at McDonnell-Douglas, described in detail in HIC. (Most notably low-cost continuous manufacturing of high quality carbon-carbon parts, and relearning of the control of simulated F-15s so as to prevent crashes after damage. The latter success was the starting point for a huge program in reconfigurable flight control, centered at NASA Ames, which is now a kind of complex empire with many variations and deployments.)

Levels 3 through 5 are all model-based designs. HIC and HLADP give much more detail. Some psychologists object to model-based design – but Grossberg has countered that learned expectations systems are a core part of animal learning which needs to be a major part of any realistic design.



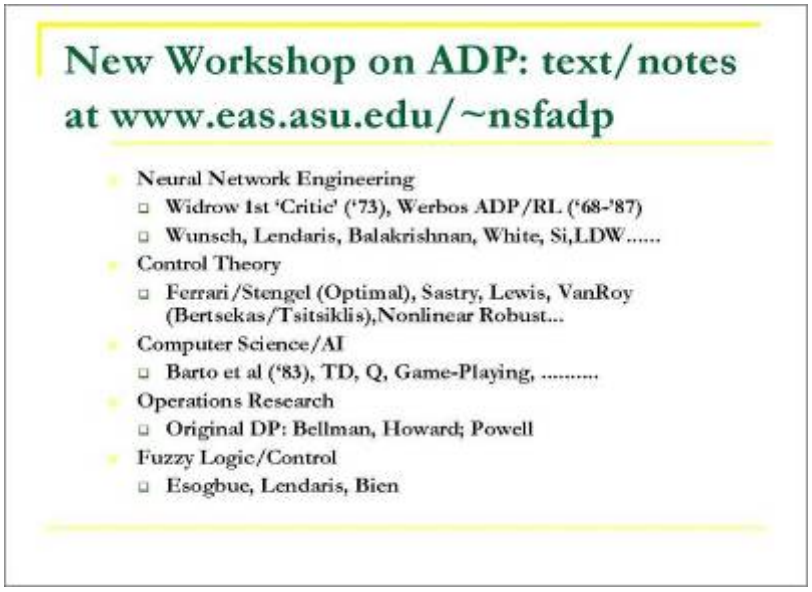
This slide summarizes some new unmet opportunities which we discussed at great length in a workshop in 2002, which led up to the new book HLADP.

This workshop addressed the general-purpose mathematical tools for optimization with foresight in the general case (where noise, nonlinearity and complexity appear).

Thousands of learned commentators have talked about management of complex systems in recent years. Hundreds of technologists have begun to talk about it, and even try to make it real. But in the end, these thousands of angels are dancing on the head of a single pin – the Bellman equation. In mathematics, there is only one exact way to solve problems in optimization over time in the general case, and that is to solve the Bellman equation.

Perhaps a decade ago, true general optimization – dynamic programming – really worked only on very special systems or systems with one or two state variables. But little-known subgroups of SEVERAL disciplines have developed new general ways to use learning and approximation to provide useful general-purpose optimizers. Our workshop was intended to bring these groups together, to learn from each other, and to chart the way to the two key opportunities: (1) greater understanding and dissemination of how to get up to 50 variables, in effect; (2) how to follow up on new ideas for how to scale up – all the way to systems as complex as small mammalian brains.

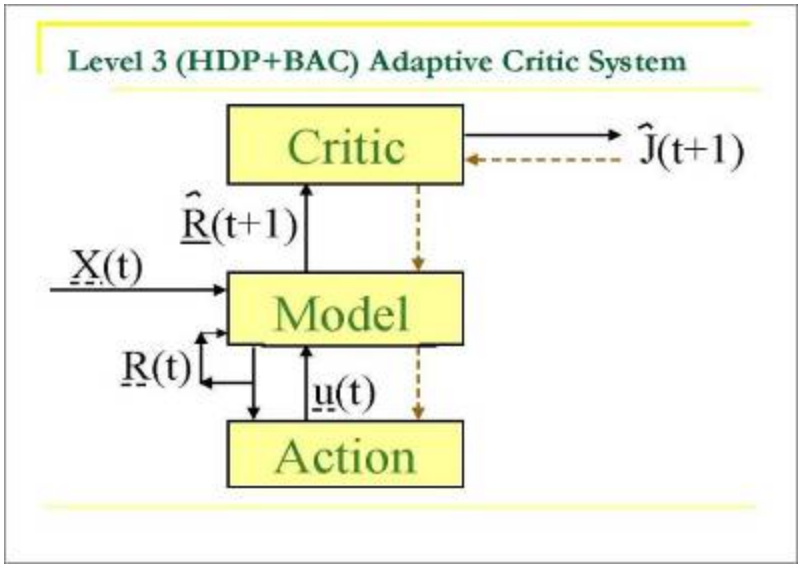
ECS has funded some work, but we believe that even an interagency initiative calling for more integration across disciplines, etc., could go further and faster, IF well-grounded in the kind of mathematical principles and issues which we have focused on. Actually, about half the effort should go into the “ADP” designs, half into critical subsystems like relevant pattern recognition, etc. Aliases for ADP: adaptive or approximate dynamic programming; reinforcement learning; adaptive critics; neurodynamic programming; HJB solvers.



This slide was discussed at very great length in Mexico, because it provides an early introduction to these 5 streams of work. It was intended to help orient people in each of the 5 disciplines to appreciate how the work in related disciplines complements their own, and fits together. There was even some time to mention the important other work depicted by "..." on the slide. Speakers in each area were more complete, of course, in reviewing their area.

Later slides in this talk (which I did not reach) reviewed the neural net area and parts of control. In control theory --Mike Athans of MIT, a leader of hard-core nonlinear robust control, has sometimes said "what we need most of all are better general-purpose numerical solvers of the HJB equation" --and that is another way of stating the goals of ADP. Adaptive control has long been interested in learning and neural networks; there is reason to believe ADP could solve important unsolved problems in that area --including tot system stability without the usual restrictive assumptions. Optimal control might even claim ADP as one of its children. Cao at this conference discussed discrete event aspects of ADP, in work which has parallels in computer science and in operations research (e.g. Melamed); it will be important to bring together these strands, to better cope with discrete variables, which are part of what the brain can deal with. Hybrid systems control al Sastry provides another window into the discrete/continuous issues, and also provides effective multiple-time-scale methods for solving and factoring the Bellman equation, important to handling complexity in the time dimension.

Computer science has done a magnificent job of educating large numbers of people about basic ADP designs, and making software available. It has also developed demonstrations of ADP on strategic games which make it clear that the word "intelligent" is not being misused when it is applied to this class of methods. It has addressed many core long-term issues. Yet many people who only know the AI part of the story believe that ADP only works on small, discrete problems, and that there are no complex real-world applications; collaboration with engineering would be crucial to dispelling such widespread misconceptions, which are part of what has hurt funding for AI in recent years.

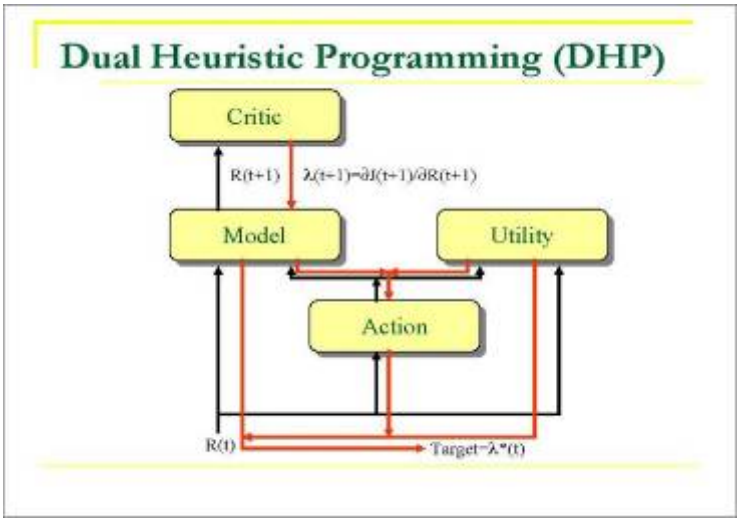


This slide is discussed in far more detail in HLADP and HIC, which give implementation details and equations. Control theorists who find the style of mathematics unfamiliar should recall that this is essentially the same design as IAC, except for the presence of the Critic at the top. A more complete control theory view is given at arXiv.org, at adap-org 9810001.

Note that I show "R," not X, going out to the Critic and Action net. This is an extremely important detail which many have missed. It is essential that the model network be a TLRN, when we are asking our controller to demonstrate "adaptive" behavior, as I discussed in previous slides. Alternatively, if you have a very complex, nasty simulation model of the plant you are trying to control, you could train a TLRN to be a "neural observer" (Frank Lewis) or state estimator (Feldkamp, Prokhorov); this means that you train the TLRN to predict the variables you won't be able to see in the real plant, based on inputs of what you can actually sense. You can then use that augmented state vector as an input to the Critic and action net.

Training a controller offline, using a nasty simulation model, is a very important tool in today's engineering. A nasty simulation model is actually easier to develop than the complete deterministic models that most control theorists ask for. A nasty model should use random numbers at each time t, and also at the start of each string of simulation; it should use them to represent not only random disturbances but possible variations in parameters, couplings, and all kind of problems that might occur in the real world. Only with a TLRN feedback structure can a controller hope to work well acdross all the data produced by such a nasty simulator. When you train the action net, you could backpropagate through a trained neural network model, trained over the nasty data – or, in theory, you could backpropagate through the model itself, if you understand generalized backpropagation and DDP well enough.

See my chapter in HLADP, the discussion of partially observed systems, for more detail.



This is “level four” of ordinary adaptive critic designs.

On the previous slide, you may have noticed that we never really use J directly there. So why calculate J, when we only need to calculate the derivatives of J with respect to components of the vector R? In fact, this slide illustrates a way of doing that.

Again, see HIC and HLADP for more details. Here I will just touch on a few key points.

Here, unlike HDP, the critic has many outputs. This means that the critic receives a lot of feedback in every time period, for its training. This means that it should have more and more advantage in learning faster than HDP, as the number of state variables grows. I developed DHP long ago as a way to overcome the obvious scaling problem with HDP. A benchmark comparison by Wunsch and Prokhorov, published in Neural Networks a few years back, shows that DHP generally performs better than HDP – and just as well as GDHP, more complicated (“level 5”) method I also developed.

Most of the best successes in real-world control problems using adaptive critics have been based on DHP, in recent years. HLADP gives many examples.

The outputs of a DHP critic actually have very deep connections to other intellectual fields. Some economists may wonder: “Are these lambdas related to Lagrange multipliers or price signals?” Indeed, they are – exactly. They are the generalization to the stochastic case of those older ideas.

The core equations of DHP are a stochastic generalization of the Pontryagin equation – discussed, again, in HLADP and HIC.

This leads to an interesting design possibility. A global critic, based on DHP, can be used to output the value signals or price signals used to manage lower-level controllers or agents in a truly optimal distributed multi-agent system. This is far superior to some of the totally ad hoc multiagent designs one finds in current literature. But to construct such a global critic requires the use of designs very high up on our tree; a simple MLP would not be powerful enough, let alone linear basis function nets!!!

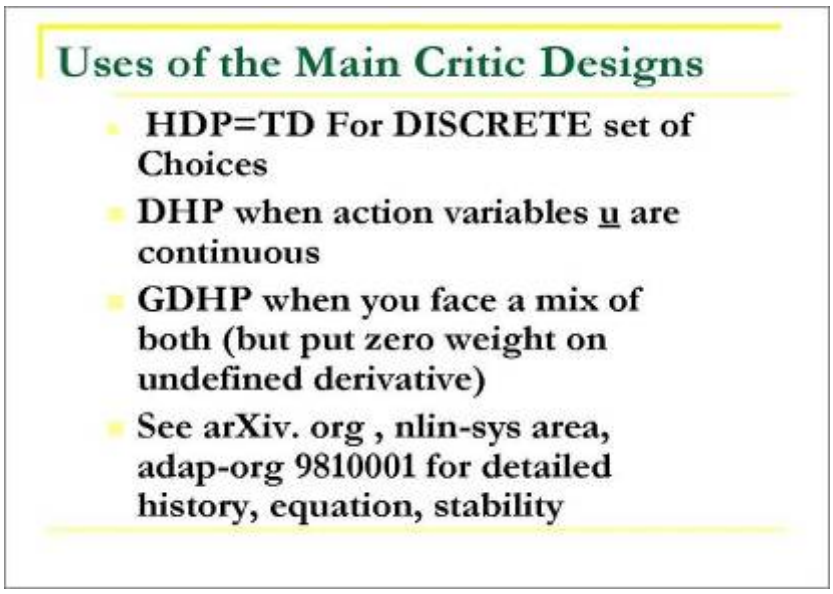
Don Wunsch, Texas Tech ADP Turbogenerator Control
CAREER 9702251, 9704734, etc.

- Stabilized voltage & reactance under intense disturbance where neuroadaptive & usual methods failed
- Being implemented in full-scale experimental grid in South Africa
- Best paper award IJCNN99

This is one of applications of DHP discussed in HLADP, and also in IJCNN03(Portland). It has continued to work on larger physical test systems, involving multiple generators and digital power switching devices ("FACTS").

Because it reduces "down times" on real generators by a substantial fraction, the team which has developed this technology is discussing possible commercial testbeds, while extending their research to address ever larger systems. The implications for the electric power system are discussed in HLADP.

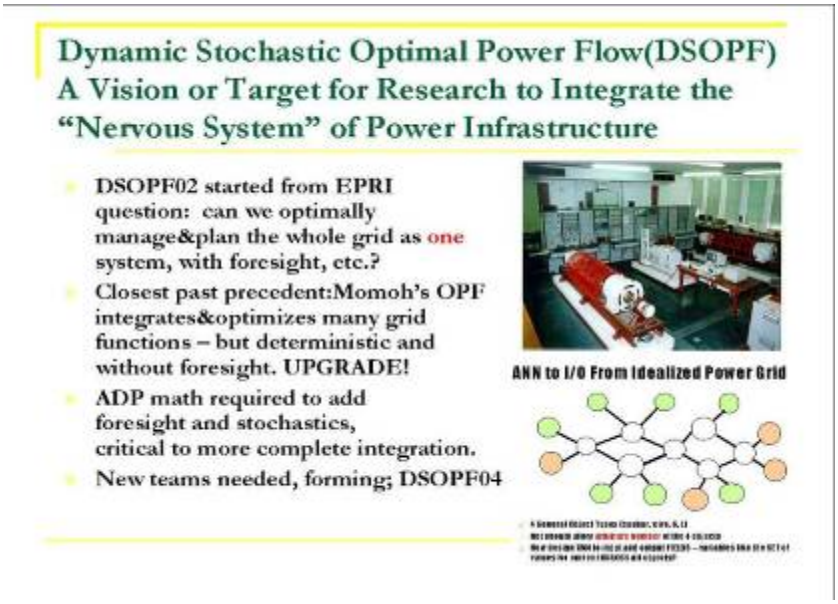
This work, centered on Missouri-Rolla (but also Georgia Tech and South Africa), is one of the important nuclei from which we could develop "dynamic stochastic optimal power flow," something badly needed in the electric power sector.



This slide was discussed at very great length in Mexico, because it provides an early introduction to these 5 streams of work. It was intended to help orient people in each of the 5 disciplines to appreciate how the work in related disciplines complements their own, and fits together. There was even some time to mention the important other work depicted by "..." on the slide. Speakers in each area were more complete, of course, in reviewing their area.

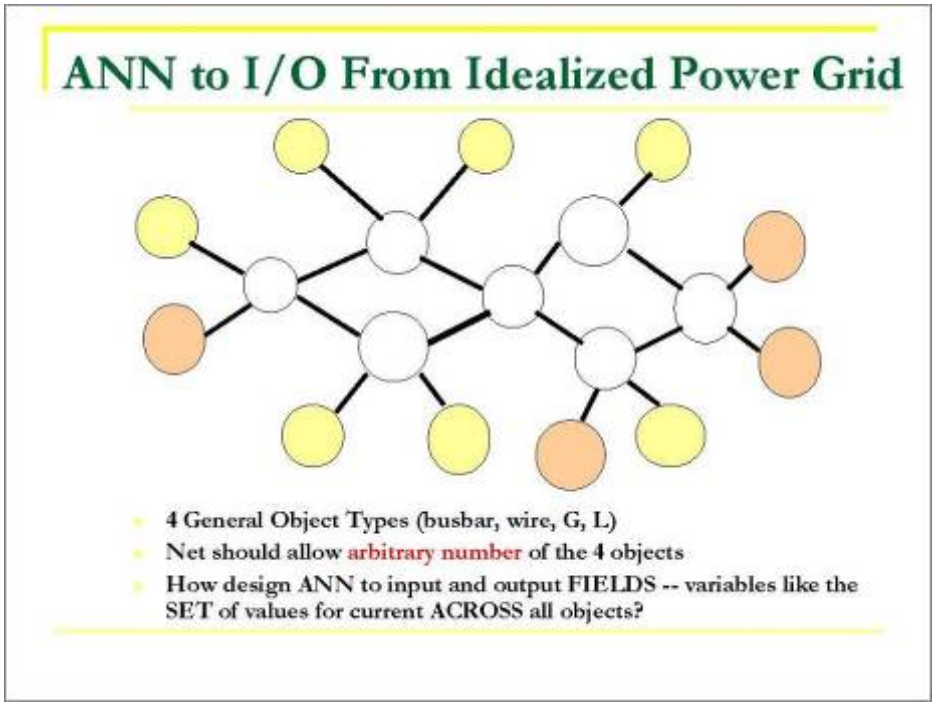
Later slides in this talk (which I did not reach) reviewed the neural net area and parts of control. In control theory --Mike Athans of MIT, a leader of hard-core nonlinear robust control, has sometimes said "what we need most of all are better general-purpose numerical solvers of the HJB equation" --and that is another way of stating the goals of ADP. Adaptive control has long been interested in learning and neural networks; there is reason to believe ADP could solve important unsolved problems in that area --including tot system stability without the usual restrictive assumptions. Optimal control might even claim ADP as one of its children. Cao at this conference discussed discrete event aspects of ADP, in work which has parallels in computer science and in operations research (e.g. Melamed); it will be important to bring together these strands, to better cope with discrete variables, which are part of what the brain can deal with. Hybrid systems control al Sastry provides another window into the discrete/continuous issues, and also provides effective multiple-time-scale methods for solving and factoring the Bellman equation, important to handling complexity in the time dimension.

Computer science has done a magnificent job of educating large numbers of people about basic ADP designs, and making software available. It has also developed demonstrations of ADP on strategic games which make it clear that the word "intelligent" is not being misused when it is applied to this class of methods. It has addressed many core long-term issues. Yet many people who only know the AI part of the story believe that ADP only works on small, discrete problems, and that there are no complex real-world applications; collaboration with engineering would be crucial to dispelling such widespread misconceptions, which are part of what has hurt funding for AI in recent years.



ADP02 was coordinated with DSOPF02, a conference joint-sponsored by NSF and the Electric Power Research
Institute (EPRI), held by EPRI's request. EPRI asked, in brief: "How can we get past the old paradigm of designing and controlling little pieces of the grid separately? How can we get to a new paradigm, where we optimize EVERYTHING in a unified way, as one big optimization problem, from operation to planning, so as to maximize global performance, measured by something like value added?" (I can still remember one conference where one engineer said "we don't care about value-added," and a major industry guy responded,"If you don't, we won't care about your work.")

Perhaps the closest thing to this vision NOW AVAILABLE is the Optimal Power Flow (OPF) package developed by Jim Momoh at Howard, which does integrate many functions together, and is very popular in rea industry. Perhaps the most practical way to implement the new vision proposed by EPRI is to try to UPGRADE OPF systems to fully incorporate foresight and handle stochastic effects. This leads us back to the same general mathematical challenge described in the last slide. We do have algorithmic tools now in operation which can handle a small number of generators or FACTS devices on an ADP basis (see top picture); the recent special issue of Neural Networks from IJCNN03 included two papers showing how robustness to disturbance can be tripled in such systems by moving to new technology. But to handle entire grids, we face the research challenge mentioned in the previous slide. But we do have ideas, at least on paper, for how to rise to this challenge. There do exist the kernels of new teams willing and able to explore these radical new directions, of importance BOTH to general-purpose cyberinfrastructure science AND to the electric power infrastructure itself. It is a long-term high-risk area, and it will not easy to build up these new islands of creative thinking, surrounded by a more conservative narrowly-disciplinary environment. But it certainly is an excellent opportunity for NSF.



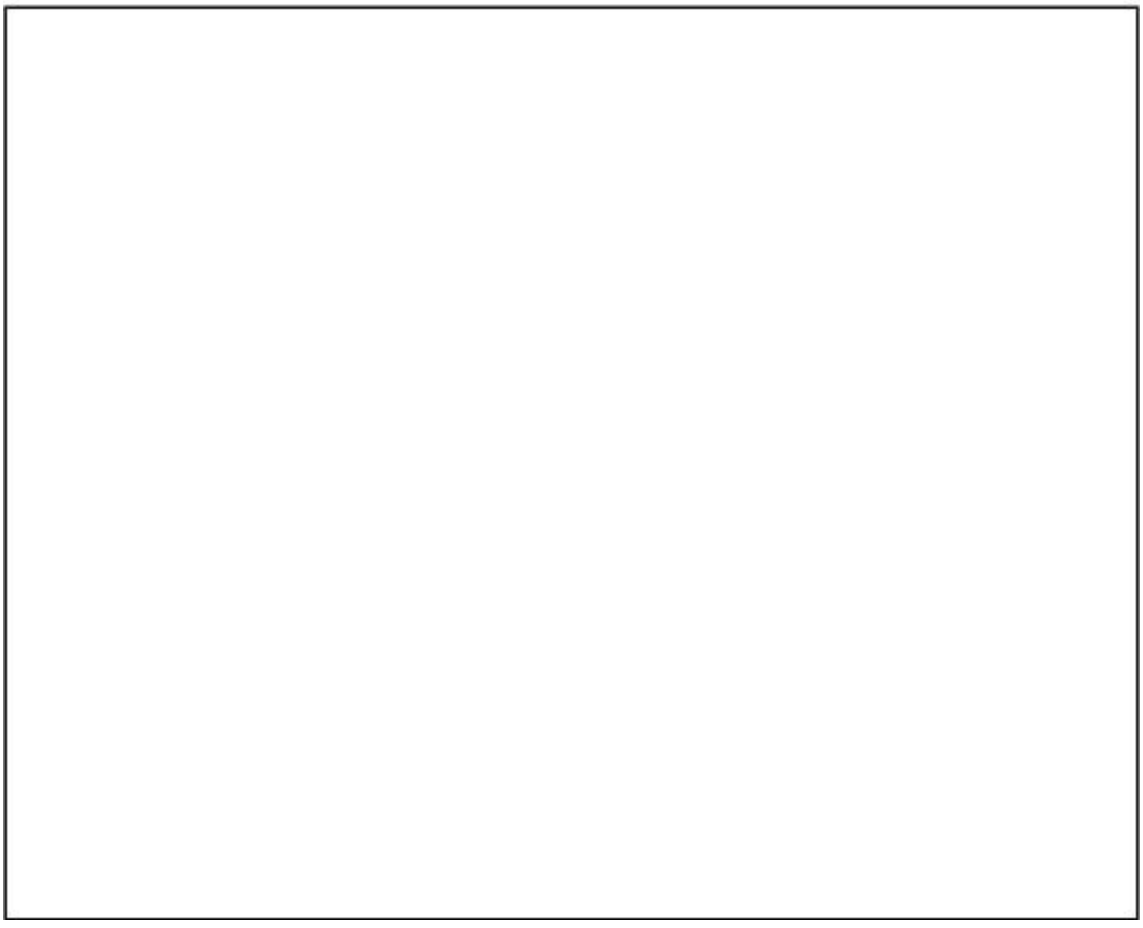
In order to apply methods like DHP to the entire global electric power grid, we would need to develop neural networks able to input and output something new – not vectors, as in ordinary neural nets – but complete relational networks.

This slide illustrates the problem.

To solve the problem, I have developed a concept called “Object Nets,” for which I recently received a patent. The core idea is illustrated in the next two slides. ObjectNets are basically a concrete piece pulled out of the more complex design concept outlined in my chapter in Karny et al eds, Dealing With Complexity: A Neural Network Approach, Springer, 1997.

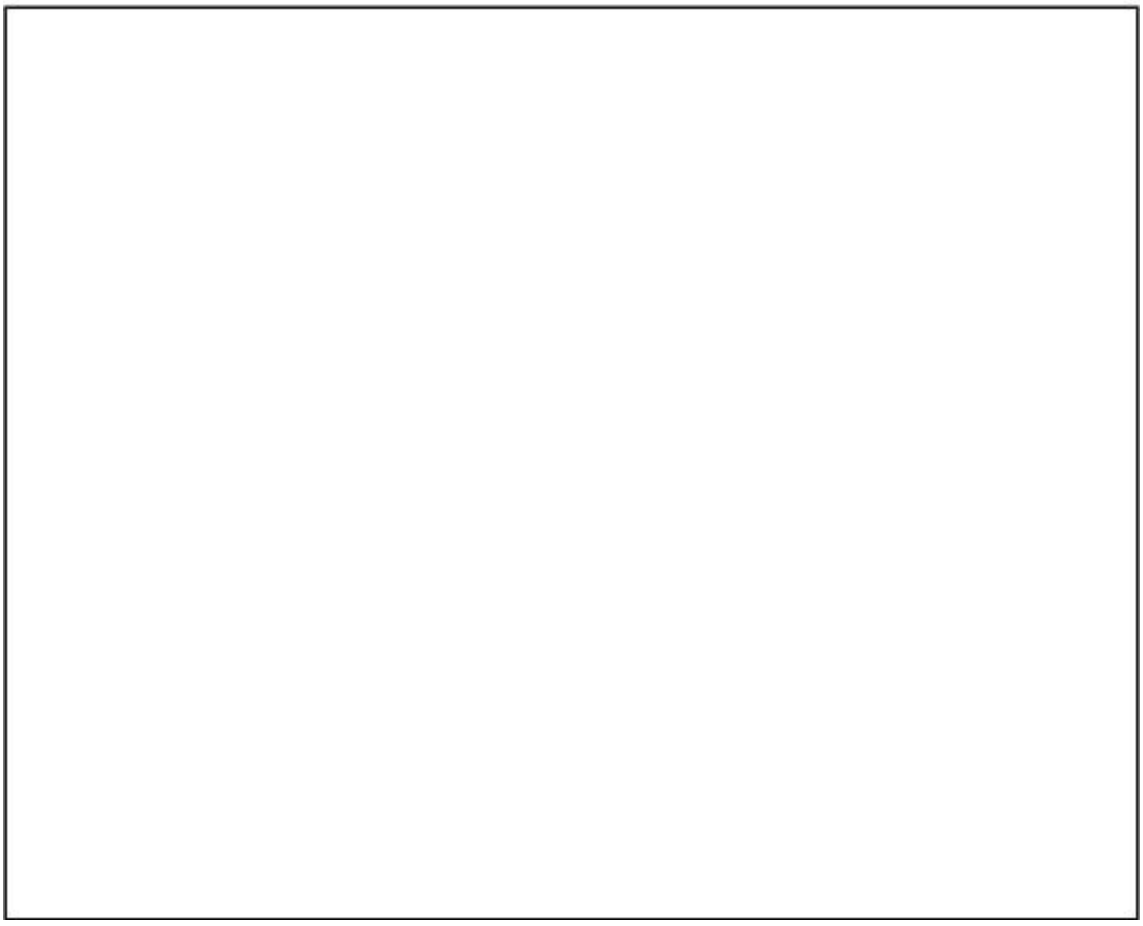
Simple Approach to Grid-Grid Prediction in Feedforward (FF) Case

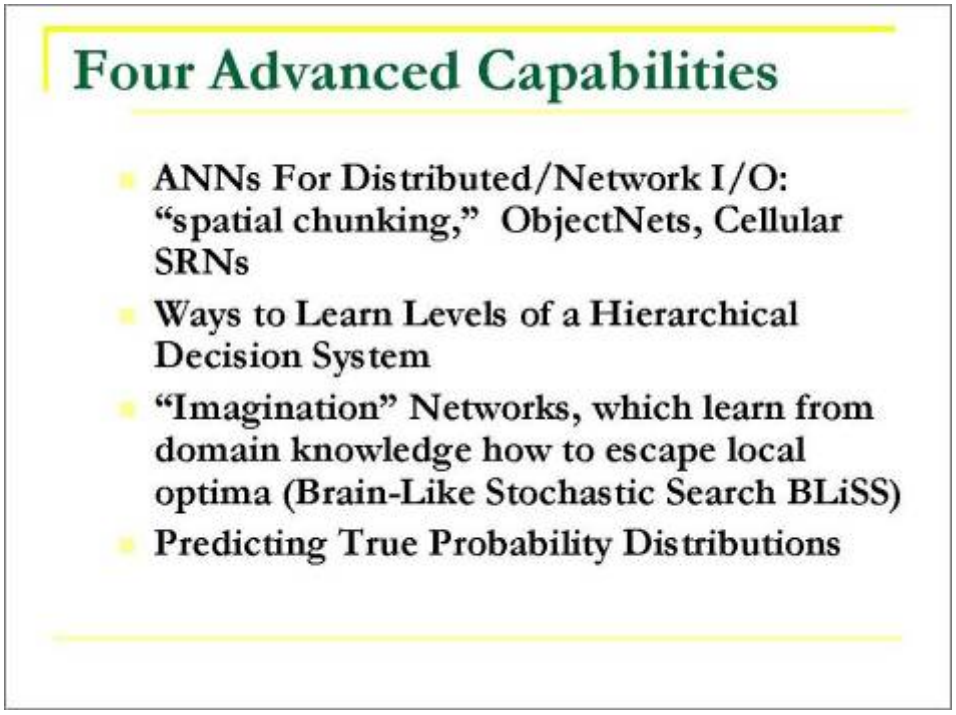
- Train 4 FF Nets, one for each TYPE of object, over all data on that object.
- E.g.: Predict Busbar(t+1) as function of Busbar(t) and Wire(t) for all 4 wires linked to that busbar (imposing symmetry).
- Dortmund diagnostic system uses this idea
- This IMPLICITLY defines a global FF net which inputs $\underline{X}(t)$ and outputs grid prediction



ObjectNets: A Recurrent Generalization (patent pending)

- Define a global FF Net, FF, as the combination of local object model networks, as before
- Add an auxiliary vector, $\underline{y}$, defined as a field over the grid (just like $\underline{X}$ itself)
- The structure of the object net is an SRN:
  - $\underline{y}^{[k+1]} = FF(\underline{X}(t), \underline{y}^{[k]}, W)$
  - prediction (e.g. $\underline{X}(t+1)$) = $g(\underline{y}^{[\infty]})$
- Train SRNs as in xxx.lanl.gov, adap-org 9806001
- General I/O Mapping -- Key to Value Functions

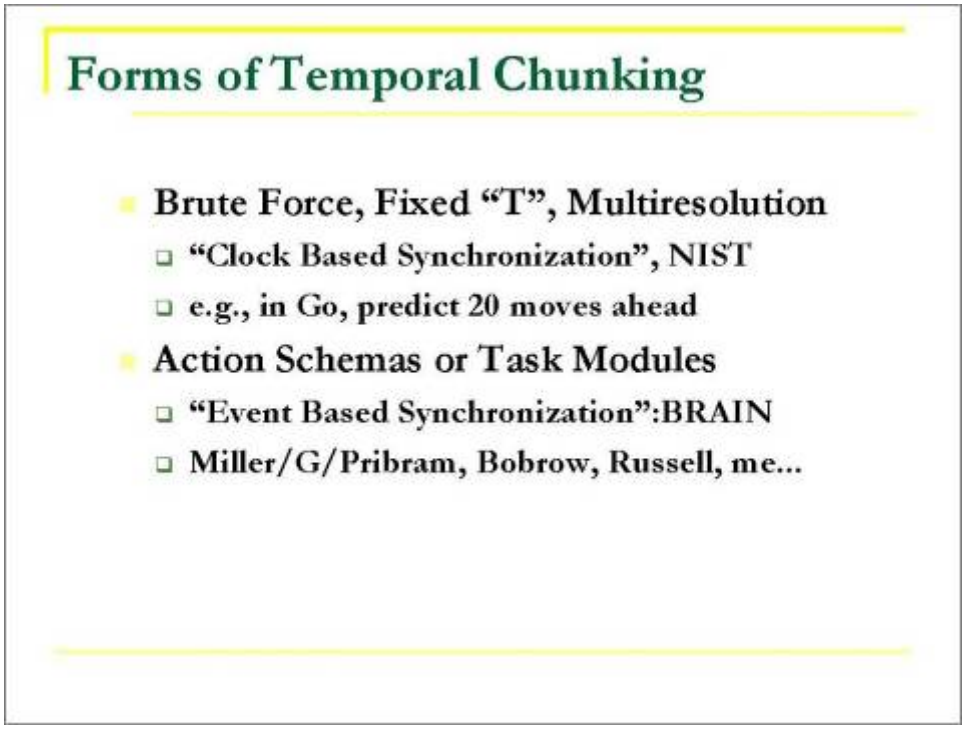




ADP has already made significant progress. As a rough rule of thumb, we can now handle highly nonlinear problems involving 10-50 continuous variables --a vast improvement over the early days of ADP. For some types of challenging real-world nonlinear problems, Powell has shown that ADP can even handle many thousands of variables. So, again, we face two major challenges: (1) how to consolidate, upgrade, teach and apply what we have already accomplished; (2) how to bridge the remaining gap to the basic mammal level of capability here.

This slide depicts one view of what we need to do to bridge the gap. In essence, we need to build systems which handle greater complexity across SPACE (the first point) and across TIME (the second point). As we do so, we will need address a third need, the need for "imagination" subsystems. In today's ADP systems, people choose actions u(t) at any time mainly by exhaustively searching every possible action, or else by using the output of a trained "Action net" or "Controller." As life becomes more complex, we will need to replace "Action nets" with "Option nets," and learn to search through a varied but finite set of possible options. We will also need to develop and use subsystems and testbeds which live up more completely to the stochastic capabilities of DP.

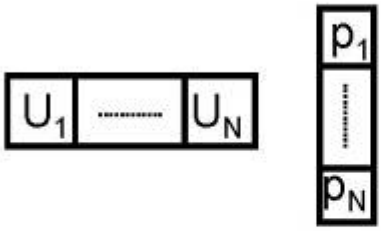
I will soon discuss the first three issues, more or less in order. The list here comes from my early (regrettably complicated) paper in M. Karny et al, Dealing With Complexity, Springer, 1997, which I will cite further. Of course, we have all learned a lot since 1997, and have a lot more to learn, but there may be useful ideas buried in that paper.



The next six slides summarize some ideas I have explored myself for coping with temporal structure, initially published in enormous detail in Karny et al (1997), and later summarized in a paper in Narendra's Proceedings on Learning and Adaptation (Narendra, Yale, 2000). The key to this design was a new set of factored Bellman's equations, which are similar in gross appearance to Sutton's earlier "Bellman option equation" but radically different in properties and context. (They actually originated from the 1978 final report to DARPA of a grant I then had from them in conflict prediction and management. The main stimulus to my revisiting them was discussion about the mammal brain in workshops organized by Karl Pribram, reported in his edited books from Erlbaum. It is clear that the "loop" structure between the cerebral cortex and the basal ganglia and the thalamus really do implement something similar in character to all of this.)

In order to demonstrate the value of such approaches in engineering, I have suggested the use of partitioned state spaces (as discussed in more detail in Narendra's workshop) as a starting point for better performance in control. This could be applied for example to fuzzy or gain-scheduled kinds of control structures.Of course, it would be interesting to see what synergies exist between this approach to temporal structure and the approaches discussed by Dietterich and Sastry at this workshop.

Lookup Table Adaptive Critics 1



$$\langle U(\underline{x}) \rangle =$$

$$\text{SUM (over i) } U_i\, p_i$$

$$= U^T \underline{p} \text{ or } U^T \underline{x}$$

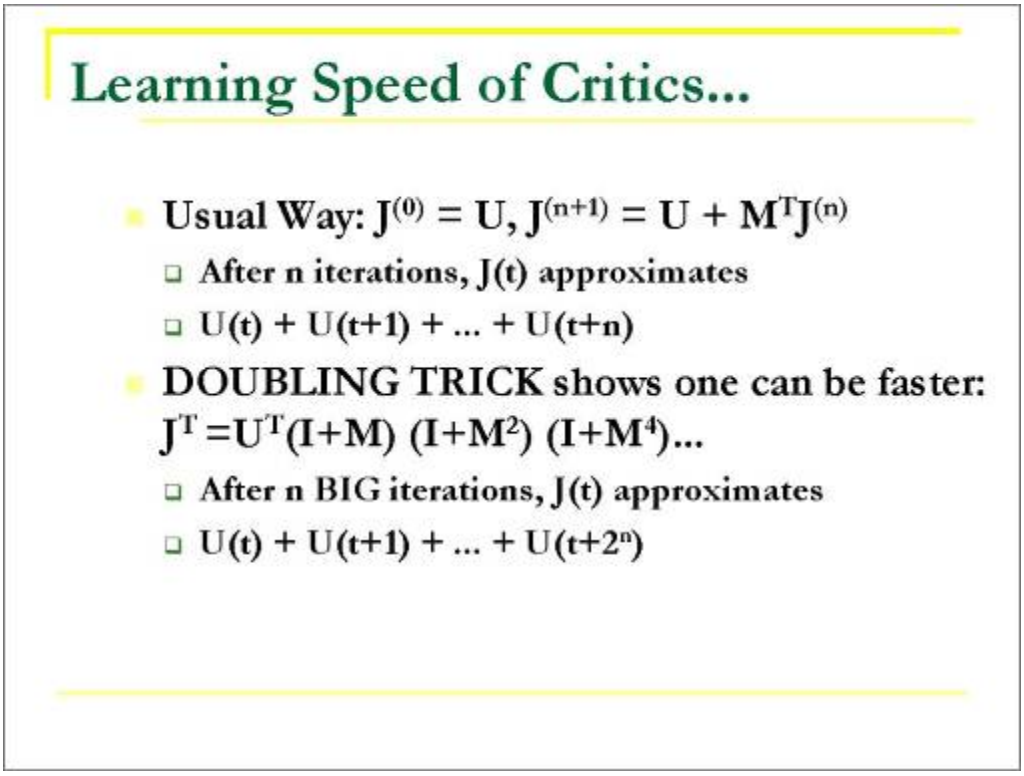
Where $p_i = Pr(x_i)$ AND $M_{ij} = Pr(x_i(t+1) \mid x_i(t))$

Review of Lookup Table Critics 2

Bellman: $J(\underline{x}(t)) = \langle U(\underline{x}(t)) + J(x(t+1)) \rangle$

$$J^T\underline{x} = U^T\underline{x} + J^TM\underline{x}$$

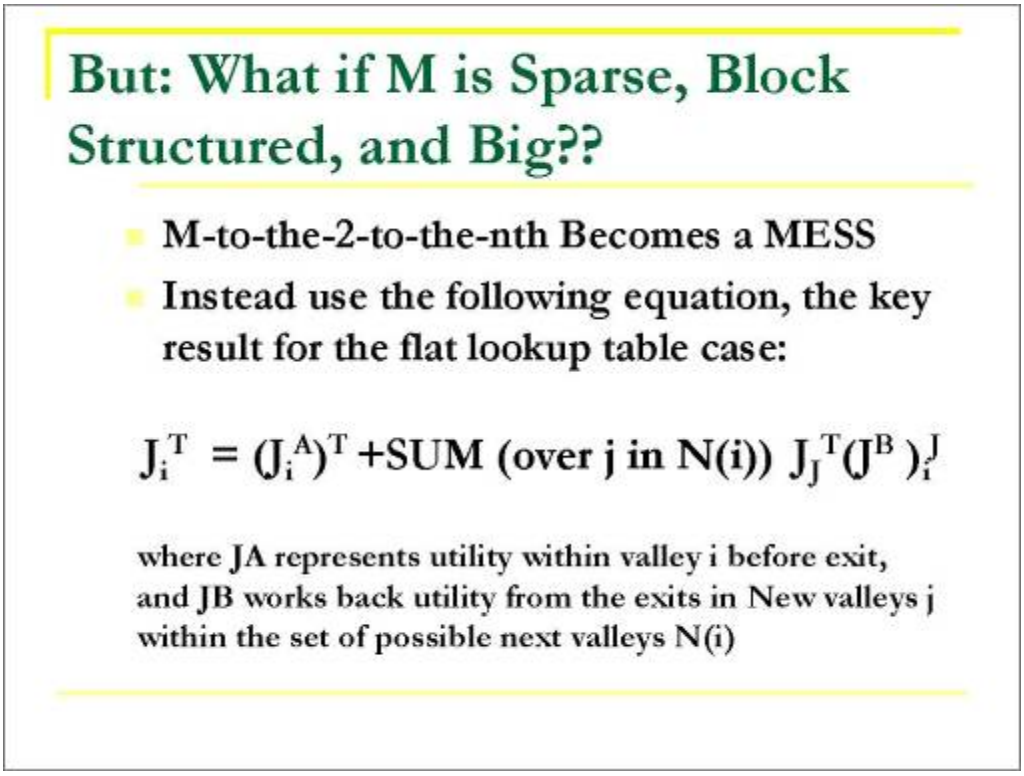
$$J^T = U^T(I-M)^{-1}$$



In theory, the ordinary type of adaptive critic can eventually converge to the best control strategy. What more do we need? Wouldn't that be enough for complex concepts like multiple time scales to be LEARNED by the system, without any need to hard wire that kind of structure? If naïve versions of GDHP learn slowly, couldn't we solve by using better learning procedures instead of apriori structure? These are very important questions; I now think the answer is "no," but certainly it would be reasonable for someone to try to revisit the questions, and see if hey could come up with something new.

For now, I think we can do better on learning speed, by exploiting a neural net cleaned-up version of an old AI trick which was called "temporal chunking" This slide gives an example, going back to the underlying mathematics of how we can do better, in a clean mathematical way – for the case of a plant with a finite number of possible states.

(By the way, many researchers get confused about the difference between a "state" and a "state variable." A system with just one continuous state variable, x1, actually has an infinite number of possible states that it could be in.)



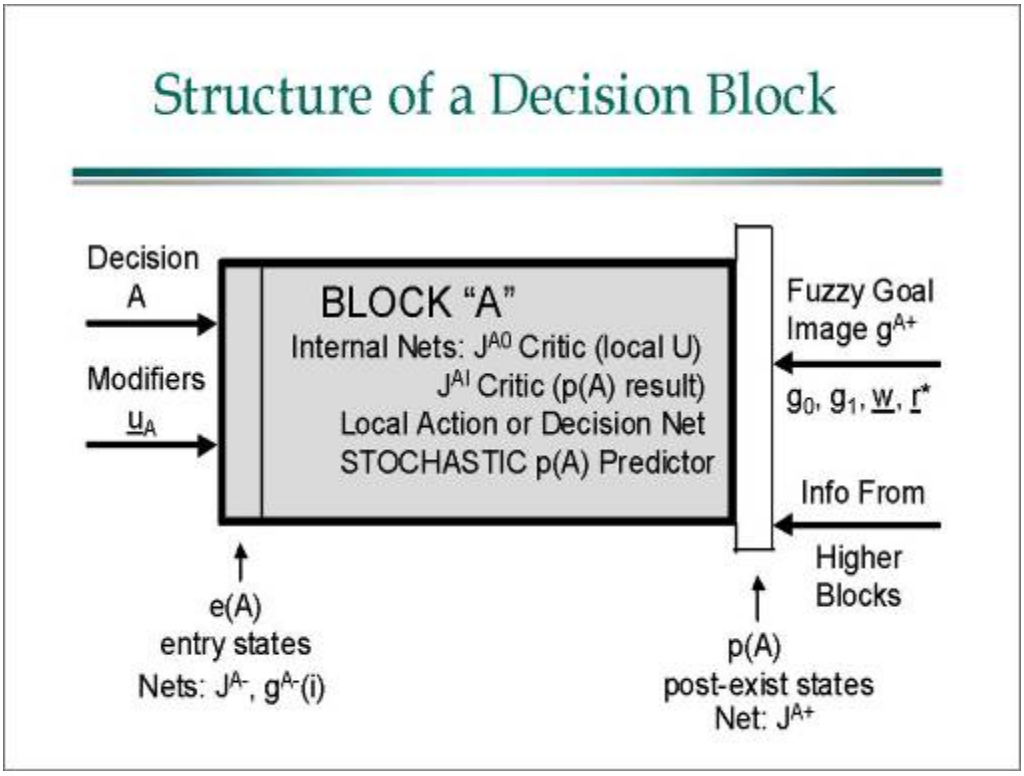
In building a neural net system to learn faster over time, we cannot just use the trick on the previous slide. This, in Karny et al, I developed a new pair of equations, which I might like to call the Werbos-Bellman equations. There are actually several versions, depending on the specification of the task structure. (The same is true of the Bellman equation!)

The Werbos-Bellman equations are similar in flavor to an earlier attempted extension by Sutton, in his first paper on "options;" however, the extended version includes a few extra dimensions, which are essential to making the idea viable for general-purpose neural network design.

Many AI researchers, inspired by Barto and by Sutton, have begun to ask about multiple time intervals – but there is clearly a need for more continuous-variable engineering and neural network thinking to complement those efforts, in order to develop working systems that exploit the principles here.

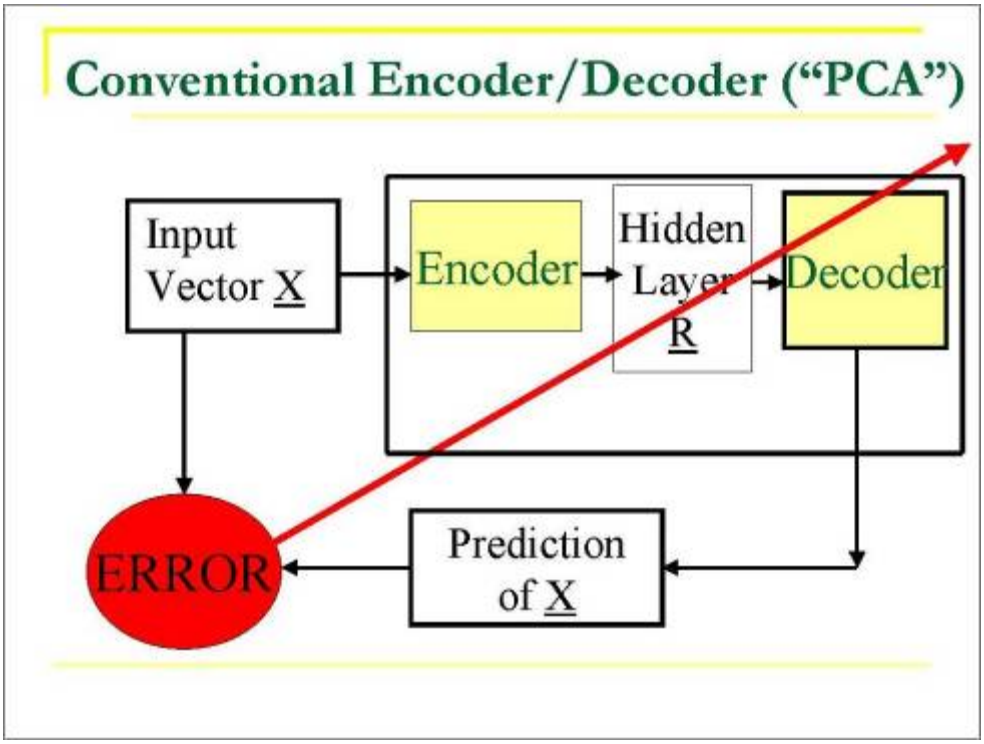
For now, this is an area of research, not a set of existing implementations.

Since the paper in Karny et al was very complex, I did try to summarize and extract and summarize some ideas for how to use the Werbos-Bellman equations in some engineering applications (e.g. when we need to partition the state space for optimal learning), in my paper on multiple models and ADP for the Yale Workshop on Learning and Adaptive Systems, Proceedings available from Narendra at Yale.



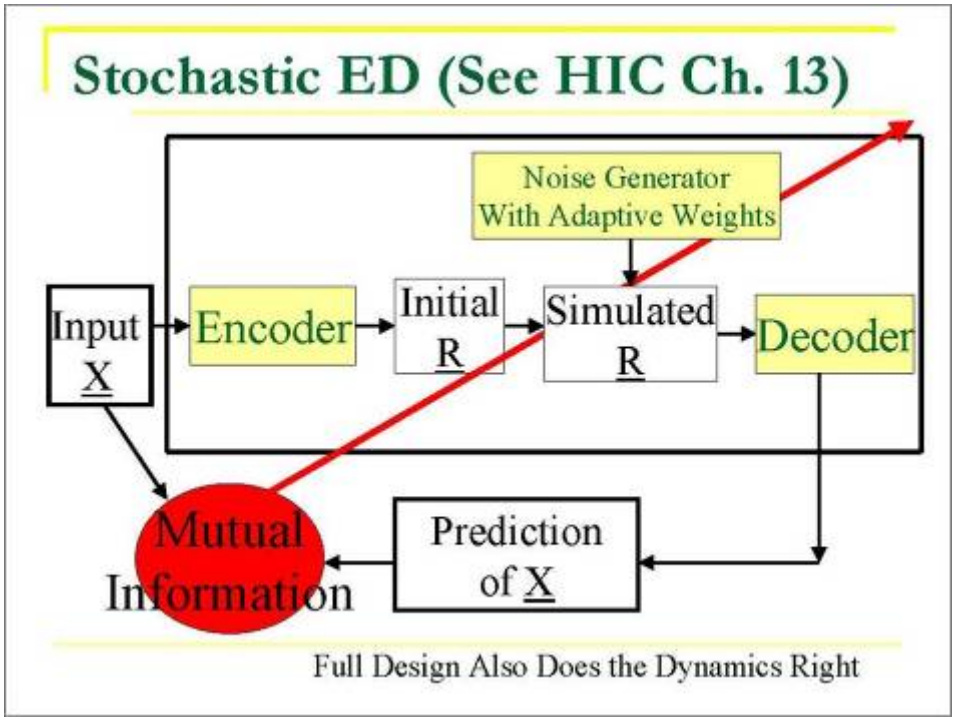
This came from Karny et al. It is part of my view of the path up to ANN designs which would capture the full power of high-level intelligence in the mammal brain. I do not recommend it for near-term engineering applications.. but it is nice to have a glimpse of the top of the ladder, even through the clouds.

Some would say that this is analogous to Sutton's idea of "options" (discussed in HLADP) – but it is PARAMETRIZED, and is much closer to the theory or vision of Miller, Gallanter and Pribram in their classic book on plans and the structure of behavior. I would claim it is the core of the "deep structure" of language – that a "decision block" is a verb, the goal an object (to some extent), and modifiers adverbs. To explain the subjects of sentences… one must go above the mouse level, and beyond the scope of this tutorial.

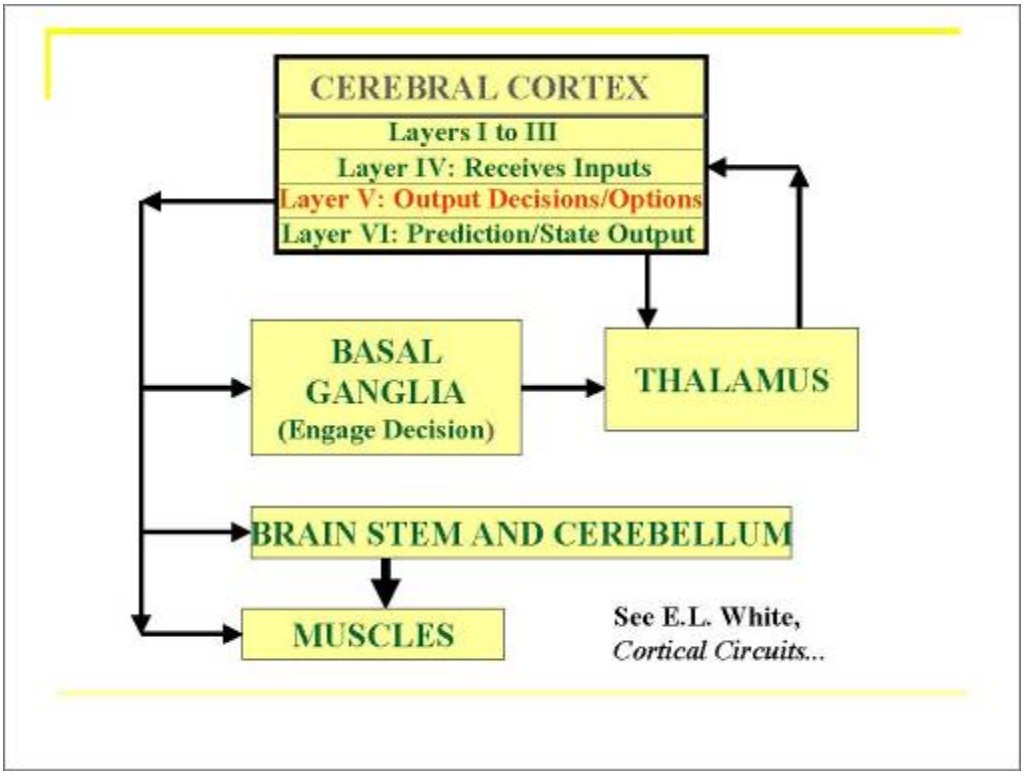


This kind of encoder-decoder design, or "bottleneck design" or "autoassociator" is very familiar in the ANN literature. Cottrell, for example, wrote some seminal papers on this. In ICNN88, I proposed some extensions of this old idea – but I later found out that the design has some severe problems.

The design on the next slide is one way to fix them.



This design, in chapter 13 of HIC, provides a kind of nonlinear generation of maximum likelihood factor design. Published in 1992, and also patented years ago, it provides a general way to use "mutual information" concepts for the nonlinear case. If the encoder, decoder and predictor are all chosen to be "universal deterministic predictors," this provides a kind of universal way of representing (and learning) partially observed STOCHASTIC dynamical systems. In stripped down form (nulling out the predictor), it provides a kind of nonlinear principal components capability, more principled than some of the others suggested in recent years in fields like chemical engineering and statistics.


CEREBRAL CORTEX
Layers I to III
Layer IV: Receives Inputs
Layer V: Output Decisions/Options
Layer VI: Prediction/State Output
BASAL GANGLIA
(Engage Decision)
THALAMUS
BRAIN STEM AND CEREBELLUM
MUSCLES
See E.L. White,
Cortical Circuits...