Chapter 1
Introduction

1.1 The Need for Unified Computational Intelligence

Humanity's newfound ability to compute has granted us unprecedented acceleration of the rate of technological innovation, altered dramatically the structure of social engagement, and presented us with a chance to create a reality to match our imagination. From optimal control to bioinformatics to economic systems, the field of computational intelligence has proven itself a leader in maximizing what society stands to gain from its increasing supply of computing resources. Riding the bleeding edge of what computers are able to do, computational intelligence researchers find ways to increase the energy output from renewable sources, secure large-scale networks, stabilize power grids, control aircraft, detect skin cancer, find land mines, and even teach humans something new about games they have played for millennia. This book introduces ways to unify some key elements of the computational intelligence field in order to guide humanity's quest for a more perfect alignment between her most ancient dreams and her everyday life.

Computational intelligence encompasses a wide variety of techniques that allow computation to *learn*, to *adapt*, and to *seek*. That is, they may be designed to *learn* information without explicit programming regarding the nature of the content to be retained, they may be imbued with the functionality to *adapt* to maintain their course within a complex and unpredictably changing environment, and they may help us *seek* out truths about our own dynamics and lives through their inclusion in complex system modeling. These capabilities place our ability to compute in a category apart from our ability to erect suspension bridges, although both are products of technological advancement and reflect an increased understanding of our world. In this book, we show how to unify aspects of *learning* and *adaptation* within the computational intelligence framework. While a number of algorithms exist that fall under the umbrella of computational intelligence, with new ones added every year, all of them focus on the capabilities of *learning*, *adapting*, and helping us *seek*. So, the term *unified computational intelligence* relates not to the individual algorithms but to the underlying goals driving them. This book focuses on the computational intelligence areas of neural networks and dynamic programming, showing how to unify aspects of these areas

J. Seiffertt & D.C. Wunsch: Unified Computational Intell. for Complex Sys., ALO 6, pp. 1–17.
springerlink.com

to create new, more powerful, computational intelligence architectures to apply to new problem domains.

The first part of this book, Chapters 1 through 3, introduces an approach to unifying the ability of computational intelligence methods to *learn*. The second part, Chapters 4 through 6, discuss unification aimed at increasing the capability to *adapt.* Finally, the third part, Chapter 7, speaks to the computational spark allowing us to *seek*.

Before discussing an application that requires the use of a unified computational intelligence approach, a clarification of the usage of the word *unified* in comparison to the oft-used term *hybrid* is in order. As used in the literature, a hybrid computational intelligence technique is one that combines multiple algorithms into a single implementation. For example, a neural network trained with a particle swarm optimizer or an evolutionary algorithm that incorporates fuzzy logic in its fitness function may be classified as hybrid. The term *unified* refers to combinations along a different axis. For unified learning, a single architecture incorporates all three canonical learning modes into one unit so that signals from each learning mode can update and access the same content. This paradigm says, "You know these learning methods that are out there, segregated, each with their own algorithm? Well, here's the thing—they are actually the same algorithm." This is very different from what is meant by a hybrid algorithm, as a hybrid algorithm does not require the sharing of memory space. Also, the highly mathematical section of this book discusses unification in terms of input domains. There, the appropriate words are, "You know how there is one set of equations for continuous domains and another set for discrete domains? Well, here's the thing—these equations are actually the same thing." In this book, a framework is provided to deal cogently with multiple learning methods as subsets of unified learning in much the same sense that differential and difference equations are treated as special cases of the unified dynamic equations.

Further, it must be noted that *unified computational intelligence* means that a given algorithm seamlessly incorporates multiple learning modes that share weights and that may influence the already learned associations of the other modes, or that the algorithm will work on both discrete and continuous input signals using a single set of equations. In this way, these algorithms unify within the *learn* and *adapt* characteristics of computational intelligence approaches. *Unified computational intelligence* is not a term used to discuss an algorithm that can act as a neural network, a swarm intelligence system, and a fuzzy logic system all at once. The *unification* occurs at the "what is the algorithm doing" level, not at the "how is the algorithm classified" level. While this latter form of unification may be discussed within the purview of mathematical logic where all algorithms may be related, it is not within the scope of the current book.

What follows now is an example to help motivate the desire to develop such unified architectures. The example described is only one of many areas in which combining multiple learning methods can be fruitful. For example, Mohagheghi, Venayagamoorthy, and Harley (2007b) report on using multiple learning modes for a wide area controller for power systems, although their design is not "unified" in the sense described in this book. What it does, however, is give a powerful

incentive to further develop algorithms that incorporate multiple learning modes, as the control of the power grid is one of the key control applications currently under investigation. Additionally, applications from finance are well suited for this approach.

We cover one such application in this field in the Future Work section. This example is a smart sensor application involving developing situation awareness for a force protection scenario. We provide here an overview and discussion of why unified learning is advantageous; full details may be found in Chapter 3. Figure 1.1 presents a graphical depiction of the problem. The practical considerations of this work include the need to develop field-deployable hardware capable of performing intelligent sensor fusion quickly, efficiently, and with minimum overhead.

An intelligent sensor fusion algorithm, like an intelligent creature, can make informed use of all three types of learning in this environment on the data set given. Certain paths may be pretrained prior to deployment, thus granting the human operators license to verify that the most obvious sensor patterns will be classified successfully. During operation, a reinforcement signal provided either by the environment or by the human operator acting off of the fusion algorithm's recommendations can adjust the current adaptive weight profile to curtail faulty clustering. Finally, in the absence of any external signal, the algorithm will learn in an unsupervised manner, comparing current inputs to what it already knows. In this way, all three learning methods are incorporated into a single application, providing a need for unified learning rather than a conglomeration of techniques pieced together in a computational sprawl.

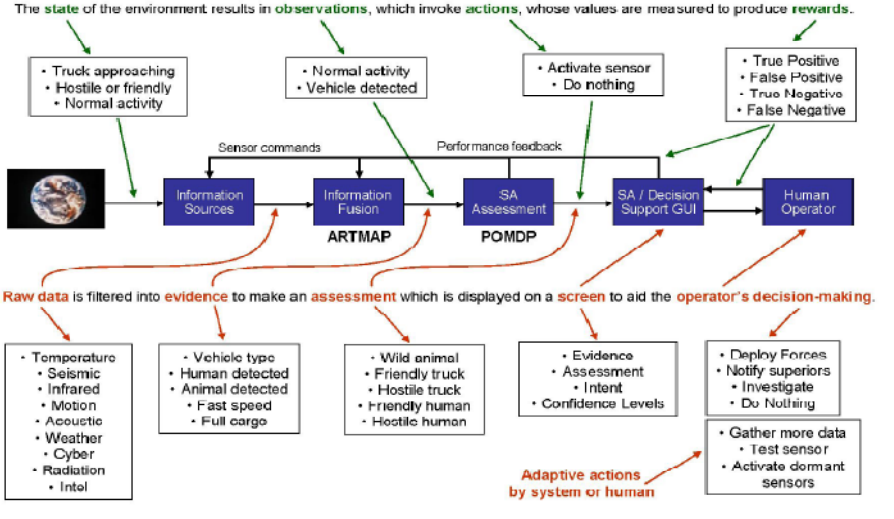

Fig. 1.1 An Application Using Unified Computational Intelligence

1.2 Contributions of This Work

The contributions of this book fall in the area of unified computational intelligence and encompass algorithm design, applications, theoretical developments, and the identification of new frontiers for multidisciplinary research.

Chapter 1 presents a new way to look at unified learning systems using a Markov Decision Processes framework. It also introduces new problem domains to which these unified approaches may be applied and defines the notion of unified computational intelligence.

Chapter 2 outlines an entirely novel Adaptive Resonance Theory-based unified learning architecture. From the motivation behind the algorithm to the design details to the extensions for future research, everything in this chapter is novel.

Chapter 3 contains an article that has appeared in the journal *Neural Networks* detailing an application of the architecture presented in Chapter 2.

Chapter 4 begins the theoretical component of the book. A new theorem in the time scales calculus is proven, and mathematics from scattered sources is brought together and organized for the first time.

Chapter 5 develops the theory of dynamic programming on time scales, one of the components of unified computational intelligence. This chapter contains new theorems regarding the nature of the dynamic programming algorithm and the Hamilton-Jacobi-Bellman equation in the time scales calculus. Also presented are new results from the area of quantum calculus. These results mark the first occurrence of the fields of time scales mathematics and dynamic programming being brought together.

Chapter 6 extends the time scales analysis to the domain of neural network learning, where the backpropagation algorithm is proven to hold in this new calculus as well as in its quantum calculus rendition. Additionally, the idea of an ordered derivative on time scales, a concept fundamental to the backpropagation algorithm, is introduced. The results in this chapter represent the first work to be published uniting times scales with neural network learning.

Chapter 7 discusses applications of computational intelligence in the emerging field of agent-based computational social science. It is increasingly important for researchers trying to make sense of complex economic, financial, and social systems to have as part of their technical vocabulary the language of computational intelligence. This chapter details how the approaches described in the book may be used in a setting outside the mainstream of engineering.

Altogether, this book introduces a new approach within the increasingly relevant field of computational intelligence and maps out new paths this research can take.

1.3 The Three Types of Machine Learning

1.3.1 Unsupervised Learning

Also called *clustering*, unsupervised learning refers to a situation in which an algorithm has no external guidance to focus its attention. When learning the

mapping of inputs to clusters, it must rely solely on its own internal structure. For a full treatment of clustering, the reader is directed to Xu & Wunsch, 2008.

The type of unsupervised learning algorithm considered most thoroughly in this book is a neural network approach called Adaptive Resonance Theory (ART). Developed by Carpenter and Grossberg as a solution to the plasticity and stability dilemma, i.e., how adaptable (plastic) should a learning system be so that it does not suffer from catastrophic forgetting of previously-learned rules, ART can learn arbitrary input patterns in a stable, fast, and self-organizing way, thus overcoming the effect of learning instability that plagues many other competitive networks. ART is a learning theory hypothesizing that resonance in neural circuits can trigger fast learning.

ART exhibits theoretically rigorous properties desired by neuroscientists, which solved some of the major difficulties faced by modelers in the field. Chief among these properties is stability under incremental learning. In fact, it is this property that translates well to the computational domain and gives the ART1 clustering algorithm, the flavor of ART most faithful to the underlying differential equation model, its high status among unsupervised learning algorithm researchers. At its heart, the ART1 algorithm relies on calculating a fitness level between an input and available categories. In this way, it appears very much like the well-known *k-means algorithm*, although the number of categories is variable and grows dynamically as needed by the given data set.

What fundamentally differentiates ART1 from similar distance-based clustering algorithms is a second fitness calculation during which a given category can reject the inclusion of an input if the input does not meet the category's standards as governed by a single global parameter. Cognitively, this is modeling the brain's generation and storage of expectations in response to neuronal stimulation. The initial fitness, measuring the degree to which each input fits each of the established categories, is considered a short-term memory trace which excites a top-down expectation from long-term memory. Computationally, this second fitness calculation acts to tune the number of categories, and it may force the creation of new categories where a *k-means* styled algorithm would not, thus exhibiting stronger, more nuanced, classification potential. The ART1 algorithm has enjoyed great popularity in a number of practical application areas of engineering interest. Its chief drawback is the requirement that input vectors be binary. The ART2 algorithm was first proposed to get around this restriction, but the Fuzzy ART modification of ART1 now powers most of the new ART research and applications.

Fuzzy ART admits input vectors with elements in the range [0,1]. Typically, a sort of preprocessing called *complement coding* is applied to the input vectors, as well as any normalization required to map the data to the specified range. Fuzzy ART's core fitness equations take a different form than those of ART1, leveraging the mechanics of fuzzy logic to accommodate analogue data vectors. Fuzzy ART incorporates fuzzy set theory into ART and extends the ART family by being

capable of learning stable recognition clusters in response to both binary and real-valued input patterns with either fast or slow learning.

The basic Fuzzy ART architecture consists of two-layer nodes or neurons, the feature representation field F_1, and the category representation field F_2, as shown in Figure 1. The neurons in layer F_1 are activated by the input pattern, while the prototypes of the formed clusters, represented by hyper-rectangles, are stored in layer F_2. The neurons in layer F_2 that are already being used as representations of input patterns are said to be committed. Correspondingly, the uncommitted neuron encodes no input patterns. The two layers are connected via adaptive weights, $\mathbf{W}_j$, emanating from node j in layer F_2. After layer F_2 is activated according to the winner-take-all competition between a certain number of committed neurons and one uncommitted neuron, an expectation is reflected in layer F_1 and compared with the input pattern. The orienting subsystem with the pre-specified vigilance parameter ρ ($0 \leq \rho \leq 1$) determines whether the expectation and the input pattern are closely matched. If the match meets the vigilance criterion, learning occurs and the weights are updated. This state is called resonance, which suggests the name of ART. On the other hand, if the vigilance criterion is not met, a reset signal is sent back to layer F_2 to shut off the current winning neuron for the entire duration of the presentation of this input pattern, and a new competition is performed among the remaining neurons. This new expectation is then projected into layer F_1, and this process repeats until the vigilance criterion is met. In the case in which an uncommitted neuron is selected for coding, a new uncommitted neuron is created to represent a potential new cluster. Researchers have concocted a wide variety of ART-based architectures by modifying the fitness equations to specialize them for a given problem domain.

For example, Gaussian ARTMAP uses the normal distribution to partition categories, with the relevant fitness equations incorporating the Gaussian kernel. This parametric statistical approach to ART was the first in what has become a rich field of study. Other parametric methods incorporate different probability distributions or allow for alternative preprocessing schemes based on statistics.

Other specializations of ART include ARTMAP-IC, which allows for input data to be inconsistently labeled and is shown to work well on medical databases; Ellipsoidal ARTMAP, which calculates elliptical category regions and produces superior results to methods based on hyper-rectangles in a number of problem domains; and a version of ART that uses category theory to better model the storage and organization of internal knowledge. Overall, Adaptive Resonance Theory enjoys much attention by those studying computational learning for both scientific and engineering purposes.

ART incorporates two steps: category choice and vigilance test. Let x be the input, w^j the weights associated with category j (this is really w_i^j where the weight is a vector of i elements, but this subscript is typically suppressed), and ρ be the vigilance.

In category choice, the degree of match is calculated:

$$\frac{|x \wedge w^j|}{|w^j|} \tag{1.1}$$

for each category j. In the ART calculations, $\wedge$ is the fuzzy AND operator, and $|x|$ represents the L_1-norm of x.

For the vigilance test, calculate

$$\frac{|x \wedge w^j|}{|x|} \tag{1.2}$$

and compare with ρ. The algorithm then cycles between category choice and the vigilance test until resonance occurs and weights update according to $w^j = w^j \wedge x$.

The relation given by Equation 2 calculates, in fuzzy logic terms, the percentage of w^j covered by x. See Figure 1.2 for a visual representation. The numerator tells us to what degree they overlap, and dividing then gives us a percentage. This is done so that the category choice reflects which category x is closest to. If one considers only the numerator, then the category choice value for the category that is identically equal to x (i.e., the perfect match) is the same as the category choice value for the uncommitted node of all **1**'s. The idea is to select the category to which the input fits just barely, and so the choice value is penalized for large w^j's. It makes no sense here to divide by $|x|$ because that quantity is the same for all categories; it would have no impact on which category is selected.

For the vigilance test, top-down expectations are checked. The category that won the competition in F2 is checked to see if it predicts that something like x ought to be the input pattern. In the fuzzy logic sense, this expectation checking is interpreted as follows:

$$\frac{|x \wedge w^j|}{|x|} \tag{1.3}$$

The algorithm now divides by $|x|$ to check what percentage of x is covered by w^j. Here, it makes no sense to divide by $|w^j|$ because it is not of concern how well w^j predicts itself. The check here, basically, is how many elements of the weights are less than elements of the inputs because w^j is everywhere larger than x this quantity is **1** and will not pass vigilance.

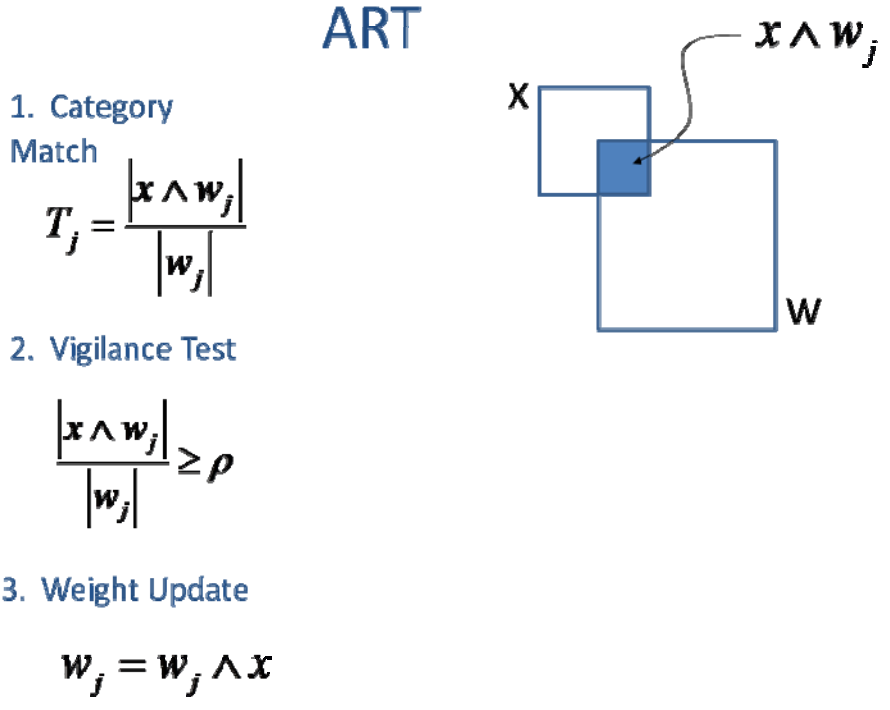

Fig. 1.2 An Adaptive Resonance Theory (ART) Classifier

This interplay between categories and inputs is the defining characteristic of ART systems.

1.3.2 Supervised Learning

Whereas in an unsupervised learning algorithm the system has no idea into what category an input *should* be placed, in a supervised learning problem this information is provided by the environment. It is the job of the system to match up the inputs with the given outputs, much as in a statistical regression problem. Typically, the supervised learning algorithm seeks to modify a list of adaptive weights in such a way as to minimize an error measure. In regression, these weights are the β_i coefficients. In a multi-layer perceptron neural network, these weights govern the movement of signals through the transfer functions at each layer. The neural network example is of principle importance and is studied in Chapter 6 in detail. In the development of the unified computational intelligence architecture of Chapters 2 and 3, however, the ARTMAP neural network is used as the supervised learning model.

1.3.3 Reinforcement Learning

Many fields, from animal learning theory to educational psychology, make use of the term *reinforcement learning* to mean a great variety of things. This book refers to a very specific mathematical definition of a problem type presented in Figure 1.3.

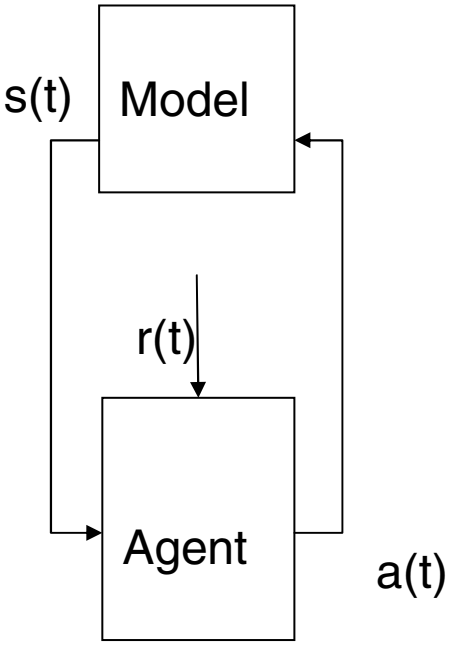

Fig. 1.3 Basic Reinforcement Learning Model Framework. Actions a(t), rewards r(t), and states s(t) are generated by the environment model and the agent controller

Some form of the Bellman equation is applied here to represent the agent's optimality criterion. It is important to understand that this literature hinges vitally on the notion of the agent as a maximizer of some utility function. In that way, there is much in the fields of economics and operations research that can usefully inform ADP theory (Werbos 2004.)

Barto and Sutton (1998) discuss a wide variety of solution methods for these problems. In particular, this chapter will focus on one solution method, a member of the TD-λ family of optimization algorithms (Sutton 1995), called Q-learning (Watkins 1989).

Note that the Q-learning algorithm, depicted in Figure 1.4, iteratively updates the value of each state-action pair. The appropriate modification is calculated based on the difference between the current and realized valuations, when

maximized over all possible next actions. This is a key fact that sets up the more advanced techniques discussed in the next chapter.

Q-Learning Algorithm

1. Initialize Q(s,a)
2. Set t = 1
3. Initialize s
4. Set a = π(s), calculate s'
5. Update $Q(s,a) = Q(s,a) + \gamma[r(s') + \delta \max_{a'} Q(s',a') - Q(s,a)]$
6. Set s = s'
7. If s is not terminal, goto 4.
8. Increment t
9. If t is not equal to the maximum number of iterations, goto 3.

Fig. 1.4 The Q-Learning Algorithm

This algorithm utilizes a lookup table to store the Q-values for each state-action pair. As the scale of the simulation grows, the amount of memory required to catalogue these values can grow at a staggering rate.

Next, the generalization of the Q-learning algorithm to the artificial Higher Order Neural Network technique of Adaptive Critics is covered.

1.3.3.1 Approximate Dynamic Programming

A Widely-Used and Increasingly effective approach to solving problems of adaptation and learning in applied problems in engineering, science, and operations research is that of Approximate Dynamic Programming (ADP) (Si 2004, Bertsekas 1996). ADP techniques have been used successfully in applications ranging from helicopter flight control (Enns 2003), to automotive engine resource management (Javeherian 2004), to linear discrete-time game theory, a topic near and dear to the heart of many an economist (Al-Tamimi, 2007). As ADP techniques continue to enjoy favor as the approach of choice for large-scale, nonlinear, dynamic control problems under uncertainty, it becomes important for the computational economist to be aware of them. Approximate Dynamic Programming is a field grounded in mathematical rigor and full of social and biological inspiration that is being used as a unification tool among researchers in many fields.

This section overviews the structure of ADP. Markov Decision Processes are discussed first to introduce the core structural terminology of the field. Next, the Bellman Equation of Dynamic Programming, the true heart of ADP, is explained.

1.3.3.2 Markov Decision Processes

First, some terminology. The state of a system records all the salient details needed by the model. For an agent deciding how much of an asset to buy or sell, the modeler may set the state space to be a count of the current number of shares the agent is holding along with the current, stochastically generated dividend payment for the next time period. In the computational modeling of games such as Chess and Go, the relevant state would be the position of all the pieces currently on the board, and possibly the number of captured stones (for Go.) At each state, the agent has a choice of actions. (In a control application, where the agent is a power plant or some other complex operation to be optimally managed, the actions are called controls.) Our economic trading agent may buy or sell a certain number of shares, the totality of which entirely enumerates its possible actions. Returning to the game example, the entire range of legal moves constitute the action set for a given board configuration, or state. Each state nets the agent a level of reward. States that lead to desirable outcomes, as measured by some reasonable criteria, are assigned positive reward, while states that should be avoided are given negative reward. For example, the state arrived at after choosing the action that moves a Chess piece such that the opponent can place one's king in checkmate would generate a highly negative reward, while a winning Tic-tac-toe move would evolve the system to a state with high reward. The manner in which the agent proceeds from state to state through the choice of action is called the evolution of the system; it is governed stochastically through transition probabilities. The agent, upon buying a number of shares of a risky asset, finds itself in a new state. Part of the state's structure, the size of the agent's holdings, is under deterministic control. The stochastic dividend payment, however, evolves according to a statistical rule unknown to the agent. Therefore, the agent cannot know for certain to which state it will advance upon taking a certain action. Instead, the next states constitute a probability distribution described by the transition probability matrix. To contrast, the evolution is completely deterministic in Chess and Go, as no randomness is involved.

The way the state has been defined, as embodying all necessary information to calculate the future system evolution, allows the use of a mathematical Markov chain to model the system dynamics. Any such system, said to satisfy the Markov Property, can be analyzed with the following techniques. In practice, systems of interest often have a degree of error in the state representation, or some other influx of imperfect information, and therefore do not technically fulfill the Markov Property. However, approximation techniques for these situations abound, and the careful researcher still can make appropriate use of Markov chain modeling in many cases. For a more thorough analysis of such cases, see Sutton and Barto (1998).

A Markov Decision Process (MDP) model is one in which Markov chains are used to analyze an agent's sequential decision-making ability. In MDP terminology, the agent calculates a policy, an assignment of an action to every possible state. The goal is to find an optimal policy, given some reasonable

criterion for optimality. An MDP consists of the components previously defined: states, actions, rewards, and transition probabilities. The time scale under consideration is also important. Discrete MDPs typically evolve along the positive integers, while continuous MDPs are defined on the non-negative real numbers. Other time scales are feasible for constructing MDPs. See the book by Bohner and Peterson (2001) for a more rigorous mathematical presentation of time scales.

MDPs have been studied and applied extensively in such areas as inventory management (Arrow 1958), behavioral biology (Kelly 1993), and medical diagnostic testing (Fakih 2006). Standard solution techniques are available and well understood (Puterman 1994). Solutions consist of an optimal policy for the agent to follow in order to maximize some measure of utility, typically infinite horizon expected reward.

It is not always the case that a system can be adequately expressed as a standard MDP. When the state information is not fully available to the agent, then the model must be supplemented with a probabilistic description of the current state, called a belief space. An MDP under this addition becomes a Partially Observable Markov Decision Process (POMDP). A classic POMDP example involves an agent deciding which of two doors to open. Behind one is a tiger, and behind the other is a lovely prince or princess ready to sweep the agent off its feet. In a straight MDP, the agent would have access to the transition probabilities for the two states and would be able to calculate which door is most likely to contain the desired result. In the POMDP formulation, however, the agent does not have access to such information. Instead, the agent receives observations, such as hearing the tiger growl, that combine to form a Bayesian approach to solving the optimal policy. POMDPs have demonstrated an ability to model a richer set of systems than the pure MDP formulation. For example, POMDPs have been used in dynamic price modeling when the exact demand faced by the vendor is unknown (Aviv 2005). When the demand at each period is known, an MDP can be used to calculate the best policy under expected reward criteria. But, when faced with an unknown state element, the agent must refer to observations such as historical marketing data to help make its decision.

Standard solution methods for POMDPs work only on specific frameworks and require significant computational capability to implement. To avoid these problems, it is common to use a technique such as a Bayesian Filter to transform a POMDP into an MDP once the observations key the agent's belief space to a sufficient degree. The solution techniques for MDPs then can be applied to the POMDP and the optimal policy calculated.

The next section provides the mathematical formulation of the task of the economic agent—the maximization of a particular optimality criterion.

1.3.3.3 The Bellman Equation

Consider an economic agent modeled with a finite set of states s, actions a, rewards $r(s)$, and transition probabilities $P(s, a)$ in a discrete time scale defined to

be the positive integers. In order to calculate the agent's optimal policy, some utility function must be maximized. In the core Approximate Dynamic Programming paradigm, the function to be maximized is the Bellman equation:

$$J(s) = r(s) + \gamma \sum_{s'} P(s', a) J(s', a) \tag{1.4}$$

This is the discounted expected reward optimality criterion. In this equation, $J(s)$ represents the current value of a given state, s' signifies the next-states, and a discount factor $\gamma$is applied to the future rewards. This equation is to be maximized over all actions. Note that this is a special case of the full Hamilton-Jacobi-Bellman equation studied in Chapter 5.

The Bellman equation states that the current value of a state is equal to the immediate reward of taking an action plus the discounted future reward that accrues from that state. Other optimality criteria are possible to account for infinite horizon or nondiscounted models. The task of ADP is to solve this equation.

One standard solution algorithm is that of backwards induction. Other approaches include value and policy iteration. The interested reader is directed to Puterman (1994) and similar texts for further details on these and other optimization techniques. The solution method to be discussed in this chapter is found in the next section.

1.3.3.4 Heuristic Dynamic Programming

Q-learning is robust and has been shown to work quite well in a large number of problem domains, including being the base of the temporal difference approach at the center of a computational agent which, without any exogenously provided understanding of the rules of Backgammon, learned to perform at the master level and which was able to teach new strategies to arguably the world's oldest game to champion-level players (Tesauro 1994). However, its reliance on a lookup table to store values is a severe limitation. Generalizations of Q-learning, falling under the heading of Heuristic Dynamic Programming (HDP), replace the Q-table with a multi-layer neural network function approximator. Another generalization of Q-learning, dubbed Z-learning, involving a variable transformation to linearize the underlying MDP formulation, has been introduced (Todorov, 2007).

The diagram for HDP, the simplest of the class of architectures broadly known as Adaptive Critic Designs (Werbos, 1992, Prokhorov, 1997), is presented in Figure 1.5.

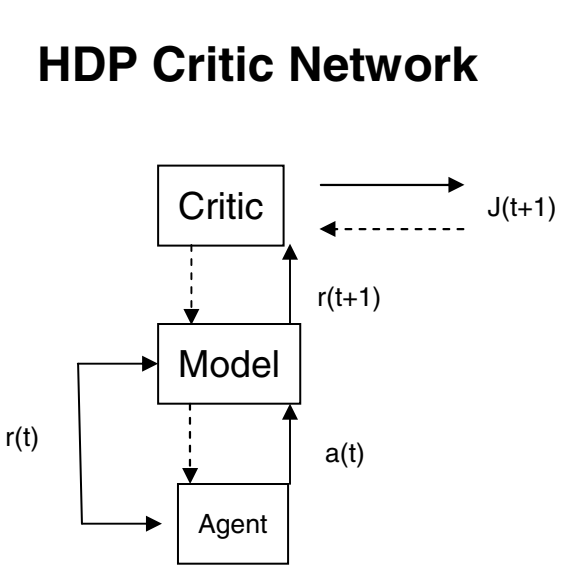

Fig. 1.5 Basic Adaptive Critic Design. J(t) is the value function being approximated, r(t) is the reward, and a(t) is the action control signal. The critic evaluates the agent's choice, modifying its adaptive weights in response to the chosen actions

The Adaptive Critic architecture, in essence, translates a reinforcement learning problem into a supervised learning problem. This is beneficial because much is known about solving supervised learning problems. The critic network learns the value function, and error between the current J-function value and the J-function value in the next time step is backpropagated through the network (Werbos, 1990).

Adaptive Critic architectures have found many application areas, including missile control (Han, 2002 and Chuan-Kai, 2005), fed-batch biochemical process optimization (Iyer, 2001), intelligent engine control (Kulkarni, 2003), multimachine power system neurocontrol (Mohagheghi, 2007), and even the management of a beaver population to prevent nuisance to humans (Padhi, 2006). The promise of finding rewarding application of these techniques in the fields of computational economics and finance is too alluring to ignore.

1.4 A Unified Approach

Based on the Markov Decision Process framework, a unified approach to the three learning modes can be established. In this unified view, unsupervised and supervised learning modes are seen as the extreme ends on a continuum of reinforcement learning. Unsupervised learning corresponds to the absence of any reinforcement, and supervised learning corresponds to the presence of perfect reinforcement.

First, consider the MDP model of a reinforcement learning agent. In this case, there are states representing salient features of the environment, and there is a

mechanism to transition from one state to another in the presence of a control or action. The value of the reinforcement will help to determine if the chosen action evolved the state of the environment into a more or less valuable position. In the context of an autonomous robot navigating a maze or an intelligent controller minimizing a cost function, the structure of the reinforcement learning model is intuitive. Note that the *critic* is the unit that processes reinforcement, and the *actor* is the unit that calculates the action.

For a supervised learning problem, however, such intuition takes more effort to establish. In this case, the environment consists solely of the inputs to be learned. Regardless of the selected control, the environment will always transition to the next input on the list. However, the reinforcement signal is not to be interpreted as a measure of value; rather, it is communicating what the correct action ought to be, which requires the supervised learning critic to behave differently than the reinforcement learning critic. Where the RL critic may hold a value function that chronicles the appropriateness (as determined by the environment) of each state-action pair, the SL critic tracks no such information. Rather, it is a function that will update adaptive weights to better coordinate the input with the actual signal received. Many RL critics do this as well when they backpropagate an error through the neural-network based actor, but in the SL case this is more explicit—the error signal to be backpropagated is generated not from the critic's value table but from an error function, typically a least squared error measure between the input and the desired output (found in the signal from the environment).

In this way, the supervised learning mode emerges as a subset of reinforcement learning. The state is now simply an input vector, the transition probability matrix gives the next input vector with probability 1, the reinforcement signal from the environment contains the desired action value (target), and the action itself represents the target vector.

For the UL case, there is no reinforcement signal at all. The environment will generate a next state, which, as in SL, is interpreted to be the next input vector on the list, but the reinforcement signal is not present.

Notice that in the UL operation, the critic is dormant; without a signal from the environment for it to process, it may as well not exist. In order to see this as a subset of more general learning, however, it is useful to view the critic as existing but simply watching the action around it take place rather than taking an active role. In this case, all the internal machinery determining to which category an input is to be assigned is located within the actor. And, as in the SL case, the transition matrix for the input is degenerate, and the system always transitions to the next input vector as the "state."

These ideas are illustrated using the following equation model based on the Q-learning algorithm. Consider the following basic update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + V - Q(s_t, a_t)] \tag{1.5}$$

This is the basic Q-learning formula, with the V component equal to the internal operation of the critic. If the idea of the critic is expanded to refer to whatever internal calculation the algorithm performs, then we may use this equation to perform both supervised and unsupervised learning in addition to reinforcement

learning. Of course, for both supervised and unsupervised learning methods, the reinforcement term r_{t+1} would be zero, but the value of V would depend on the dynamics of the particular approach being utilized. So, for a basic supervised algorithm, the V would represent least-squared error, and for basic unsupervised learning, the V would equal the fitness calculation between the input and any given category. More advanced algorithms could have V representing the ARTMAP or ART dynamics or a fuzzy c-means fitness, etc. A single update equation can be interpreted, through the activity of the critic, to perform any of the three learning modes. This is not to say that Q-learning leads to the most effective approach to supervised or unsupervised learning, only that this is the beginning of a single framework within which to discuss the three types. This framework will form the basis of the algorithms presented in Chapter 2, which show integrated learning in action.

1.5 Future Work

Extensions of the work presented in this book will cover new applications, more robust synthesis of the learning modes, and further convergence of discrete and continuous signals within a single theoretical framework.

In the application domain, there are many problems in social science in which an agent equipped with multiple learning modes may prosper. For example, a commonly studied computational economics situation is one in which a collection of heterogeneous agents must analyze an input signal that gives information about the future value of a risky asset. These agents must decide how to allocate resources between the risky asset and the risk-free asset so as to maximize payoff. This problem requires the agents to learn a match between the information signal and the future performance of the risky asset. Simple multi-layer perceptron neural network architectures have been employed successfully in approximating the functional relationship between these two signals of interest. However, in a more realistic application environment, the information signal would be expected to be intermittent, requiring the agent to process in an unsupervised mode while access to the information stream was limited. In this way, the use of a unified learning method, where the content learned in supervised and unsupervised modes would be mixed together within a single memory, may be the correct model of actual economic agents. It helps that the ART-based neural networks are themselves designed specifically to describe how biological neural and cognitive systems operate.

Furthermore, in a more robust application requiring portfolio balancing, an agent would be faced not with a single information source but with multiples. Advances in the theory of unified computational intelligence applied to the learning modes can generate algorithms that rate different supervisory signals via the reinforcement signals. When a supervisor indicates a match that subsequently generates poor reinforcement results from the environment, then the sagacity of that supervisor will be in doubt. In this way, an extension of the unified computational intelligence learning neural network presented in this book may be

able to process multiple supervisory sources. This would be an entirely new problem domain for machine learning research.

In general, the development of new learning algorithms that are capable of synthesizing different sorts of feedback signals is a much needed extension of this book. The idea that multiple signals may impact the same stored memory is one not yet represented in the literature. Further theorems and applications regarding the interplay among competing learning signals are needed.

Theoretical extension of the simple MDP model presented in this book is needed. More theorems concerning the view of supervised and unsupervised modes as the extreme versions of the reinforcement mode would help guide the development of further algorithms for applications.

On the dynamic programming front, further analysis of the unification of discrete and continuous signals is needed. The time scales calculus is still an emerging area with relatively few active researchers worldwide. As more and more of the fundamental applied mathematics gets formulated in this calculus (multiple valued Taylor series, n-variable chain rules, variational calculus, functional analysis, and nonlinear systems theory, to name a few), new results on the control front will become available. Single algorithms capable of operating in discrete or continuous time will be available and may prove worthwhile in applications. In 20 years, aspiring control theorists may have to know a thing or two about the time scales calculus and dynamic equations.

Extensions of the quantum calculus and dynamic programming are of value as well. A quantum dynamic programming can be devised using the state space representations of quantum mechanics instead of the ones in classical mechanics. This approach may allow decision theory under uncertainty to be formulated in a new and useful way.

Finally, the area of agent-based modeling of social systems in general is still in its infancy. As more computational results contribute to an understanding of complex systems to the same degree as has traditional analytic mathematics, more scientists will understand the power behind these methods and begin using them with wider acceptance. Computational intelligence techniques, themselves inspired in a way by biological systems, extend the possibilities of computation and may fuse with these agent-based models in a vital way to help advance both basic and applied research into the governing dynamics of social and economic systems.