## Temporal-difference algorithms for reinforcement learning and artificial intelligence

Research in my laboratory pursues an approach to artificial intelligence (AI) and engineering problems in which they are formulated as large optimal-control problems and approximately solved using *reinforcement learning*, a new body of theory and techniques for optimal control that has been developed in the last thirty years primarily within the machine learning and operations research communities, and which have separately become important in psychology and neuroscience. Reinforcement learning researchers have developed novel methods to approximate solutions to optimal control problems that are too large or too ill-defined for classical solution methods. My PhD supervisor, Andy Barto, and I played a leading role in re-awakening interest in this area in the 1980s, culminating in the publication of our popular introductory textbook (Sutton & Barto 1998). Some of the recent accomplishments of reinforcement learning include strategic decision-making in Watson (the computer player of *Jeopardy!* that defeated the best human players (Tesauro et al. 2012, Thompson 2010)), achieving autonomous acrobatic flight of computer controlled helicopters (Abbeel et al. 2007), and widespread application to commercial advertisement placement on the internet (e.g., see Baccot 2010). Reinforcement learning and its problem formulation—Markov decision processes—currently form the basis for one of the most active and prominent modern approaches to AI.

The primary focus of the proposed NSERC-funded research is the development of new learning algorithms and corresponding theory based on *temporal-difference (TD) learning*. TD algorithms, which I introduced (Sutton 1988), are a core technology at the heart of much of the excitement and many of the successes of modern reinforcement learning. TD methods are learning algorithms specially suited to learning to predict long-term aspects of a dense, high-dimensional time series. Their initial use in reinforcement learning is to predict the total future reward for each of a set of possible choices and thereby discover the optimal decision-making policy (other uses will be discussed shortly below). A key feature of TD algorithms is that learning is driven by the difference in temporally successive predictions (thus "temporal difference"). This *TD error* is used to update the process by which the earlier prediction is made such that in the future it produces predictions closer to the later prediction, which is presumed more accurate. For example, if features of a chess position are being used to predict the probability of ultimately winning the game, then the change in predicted probability from one position to the next would be the TD error; if it increased, then the weighting of the first position's features would be incremented so that its predicted probability is increased, and vice versa.

The TD error should be contrasted with the conventional error obtained by comparing the earlier prediction to the actual long-term outcome, which we here call the *Monte Carlo* error. The actual outcome may arrive much later, maybe seconds, minutes, even years, depending on the application. This makes using the Monte Carlo error computationally inconvenient; one has to remember all the predictions made and the basis for making them for the entire time from making the predictions until the arrival of their corresponding actual outcomes. In TD methods, in contrast, the immediately available TD error is used to update each prediction immediately after it is made. Vastly less memory and less detailed bookkeeping in required by TD methods, and this computational advantage is perhaps the most important reason to use them.

In addition to the computational advantages of TD methods, in certain circumstances they can be proven to give statistically better answers with less data. In other situations (principally when the problem deviates significantly from the Markov property) Monte Carlo methods are superior. This motivates methods such as TD($\lambda$) that can slide smoothly from TD methods ($\lambda$=0) to Monte Carlo methods ($\lambda$=1) while retaining in all cases the computational congeniality of TD methods (all computation is temporally local). Best performance is typically achieved at an intermediate value of $\lambda$. All core reinforcement learning methods have been generalized in this way, including Q-learning (Watkins 1989) and Sarsa (Rummery 1995), to produce Q($\lambda$) and Sarsa($\lambda$), although not in a completely successful way. One of the specific goals of the research proposed here is to do this more successfully, as I discuss further below.

In the late 1990s it was recognized that the most prominent signal and neurotransmitter emitted from the reward processing areas in the brains of many animals, including rats, honeybees, primates, and humans, was well understood as the TD error of the brain's reward-prediction system (Schultz, Dayan & Montague 1997). This perspective has proven extremely useful, explaining many experimental details, and is now the standard view of reward processing in the brain against which all alternative proposals must compete (e.g., see O'Doherty 2012). The story of TD errors in the brain is the most important positive example of the productive interplay of engineering and neuroscience since at least the Hebb (1949) rule. The proposed NSERC-funded work remains within the realm of AI rather than biology, but I have separately developed the TD(λ) algorithm as a model of animal learning behaviour (Sutton & Barto 1981, 1990, Ludvig, Sutton & Kehoe 2012). This confluence of engineering and biological constraints provides further motivation for a thorough study of TD algorithms.

**Recent Progress**

My group has made major progress in the last six years by introducing the gradient-TD family of algorithms, the first TD learning algorithms that are sound under off-policy training with linear function approximation, and which also have superior convergence properties under nonlinear function approximation.

*Linear function approximation* refers to cases, like the chess example mentioned earlier, in which states are represented by features, such as the number of pieces of each type and color, each of which has a corresponding scalar weight. If the prediction for a state is a simple sum of the state's features weighted by the corresponding weights, then we have linear function approximation. Once the weights have been learned the approximator can estimate the value of any chess position, whereas a table-lookup approach can only offer informed estimates for positions it has seen before. If the weighted sum is transformed into a probability estimate by applying a "squashing" function, such as the logistic function, then the overall map is nonlinear and we have a case of *nonlinear* function approximation. The nonlinear case also includes more complicated cases such neural networks (e.g., TD-Gammon, Tesauro's (1995) world-champion backgammon player based on TD(λ)). Some form of function approximation is widely viewed as essential to scaling reinforcement learning to large problems.

*Off-policy learning* refers to learning about one decision-making policy from data obtained while following a different policy. The policy learned about is called the *target policy*, and the policy used to generate the data is called the *behaviour policy*. If the two policies are the same, then we have the *on-policy* case. Typically the thing being learned is the value function for the target policy—the expected total upcoming reward given that we start in each state (or each state–action pair) and that actions in subsequent states are selected according to the target policy. For example, suppose you are observing a chemical plant while it is controlled by a person (the behaviour policy) but hoping to learn the value function for a different policy (the target policy) that you would use if you were allowed to take control of the plant. This is a case of off-policy learning. In general, off-policy learning is important whenever you want to take full advantage of the available data without completely interfering with an existing controller—a very common case indeed.

Off-policy learning arises even in *Q-learning* (Watkins 1989), perhaps the most popular of all reinforcement learning methods, and is in fact one of its most appealing features. Q-learning accepts data from any behaviour policy and attempts to estimate the state–action value function for the optimal policy (the target policy). Of course, convergence is only assured if the behaviour policy repeatedly tries every action in every state, but the optimal (target) policy will necessarily be more selective (it is typically deterministic). Thus the two policies are different and we have a case of off-policy learning. Part of the appeal of Q-learning is that the behaviour policy can be exploratory, even completely random, and still the optimal policy is guaranteed to be found (in the table-lookup case).

Now we can state the problem that we have solved with gradient-TD methods. If a TD method such as TD(λ) or Q-learning is used without function approximation (that is, in a table-lookup form, with one learned estimate for each state) then all is fine; convergence is guaranteed for both on-policy and off-

policy learning (Watkins & Dayan 1992; Jaakkolla, Jordan & Singh 1994, Tsitsiklis 1994). If linear function approximation is used together with on-policy training, then the theoretical properties are still good; convergence to a point is guaranteed for fixed target and behaviour policies, and to a region for greedy target policies (Gordon 1995). For off-policy learning, however, there is a subtle change, and convergence can no longer be assured with linear function approximation even for fixed policies. Importance sampling techniques are required to correct for the difference in the two policies, but if they are used then a certain matrix is no longer guaranteed to be positive definite, and as a result the weights may diverge to infinity on some problems. In particular, examples were found in which divergence occurs even for Q-learning (Baird 1995).

After the early negative results, many new ideas and algorithms were proposed to remedy the problem, too many to identify all here, but none (prior to our recent work) were fully satisfactory. Good algorithms were found whose complexity scaled with the square of the number of weights, such as LSTD (Bradtke & Barto 1996, Boyan 2002). This was not a satisfying solution both because the original on-policy algorithms required only linear complexity, and because many of the large applications tended to involve very large numbers of weights for which quadratic complexity was prohibitive (e.g., Computer Go, with millions of features, Silver et al. 2007). Baird (1999) proposed an initially promising linear-complexity solution based on gradient descent, but it converged slowly and required double sampling, which meant a simulation model was required. Averagers (Gordon 1995) and other interpolation methods were stable but have scaling issues and enabled only a weaker form of function approximation.

We believe that we have largely solved this problem with the introduction of gradient-TD algorithms (Sutton, Szepesvari, & Maei 2009, Sutton et al. 2009) and their elaboration into a complete family of TD methods, including GQ(λ), a multi-dimensional generalization of Q-learning (Maei & Sutton 2010, Maei et al. 2010a). Like Baird's methods, gradient-TD methods are based on true gradient-descent and inherit its robust convergence properties, but learning is faster and no double sampling is needed. Arbitrary target and (exciting) behaviour policies are permitted, and convergence guarantees are available in all cases, including cases of nonlinear function approximation (Maei et al. 2010b).

**Objectives**

Gradient-TD methods were an important advance, making linear complexity function approximation compatible with off-policy TD learning. They are more than a first step, but they are not the final step. It is better to say that they open the door to a host of new possibilities in off-policy learning. Some of these are concrete problems that are immediately plain to me and others are more long term. Some directly build upon the solution provided by gradient-TD and others concern other issues that can be addressed now that off-policy learning is possible.

In the short term, we have begun work on yet another new family of TD methods that can be viewed as a hybrid of gradient-TD methods and conventional on-policy TD methods. These *hybrid-TD* methods are just enough like gradient-TD methods to ensure convergence but are otherwise as close as possible to on-policy TD methods. On-policy updates are preferred, when they can be made safely, because they seem to result in faster convergence (Hackman 2012). Hybrid-TD methods gradually and automatically shift toward or away from on-policy methods as the target policy comes closer to or further from the behaviour policy. This happens even on a state-by-state basis, so that an on-policy TD update can be made in some states while more gradient-TD updates are made in others. Hybrid-TD methods are not true gradient-descent methods, just as on-policy methods are not, but we can show that their key matrix is positive definite and thus convergence can probably be assured. Establishing it absolutely is one objective of the proposed research.

Other near-term objectives concern completing the extension of gradient-TD methods to cover all important cases. One of these which remains incomplete is the extension to *policy-gradient actor-critic methods* (Sutton et al. 2000, Konda & Tsitsiklis 2003) for off-policy learning. We have some initial positive results (Degris, White & Sutton 2012) but the central theoretical problem remains unsolved. Work is ongoing here in collaboration with Hamid Maei at McGill and Susan Murphy at Michigan on some new

algorithmic ideas that we hope will provide an elegant solution. Related to this work are some ideas for extending all of the gradient-TD work to the *average-reward-per-step* setting that is most appropriate when working on control (as opposed to prediction) when using function approximation.

Once off-policy learning is efficient, stable, and reliable, many new issues can be investigated in the longer term. One of these is the issue of *computational analogs of curiosity*. The major benefit of off-policy learning is that the target and behaviour policies are decoupled. That is, the behaviour policy no longer has to slavishly follow whatever policy is being learned about at the time; it can be selected to achieve other purposes. One natural idea is to choose and shape the behaviour policy to maximize the total amount that is learned per unit time (Schmidhuber 1991, see Baldassarre & Mirolli 2013). Notice that, with reliable off-policy learning available, it is possible for the first time to learn about many different target policies in parallel. Previously a target policy could only be learned about only if it was followed exactly. Any deviation risked divergence. Of course, it is inevitable that one will learn more about a target policy if its actions overlap with those of the behaviour policy. But any behaviour policy will overlap with many possible target policies, and those will in general be in different stages of being learned and thus have greater and lesser needs for additional data. Shaping the behaviour to maximize the total learning per time step should greatly speed the overall learning of a large set of predictions. In my lab we have just completed several extensive studies of parallel learning of large numbers of predictions about the interaction between a robot and its environment (Modayil, White & Sutton, in press). These predictions are intended to eventually form a rich and detailed model of the robot's world, such as would be useful in planning. This setting is ripe for a sophisticated demonstration of the potentially powerful effect of computational curiosity.

We noted earlier that all of the core TD methods have been generalized to include $\lambda$, known as the *bootstrapping* parameter, but that these generalizations are not entirely satisfactory in off-policy cases such as Q($\lambda$). In particular, when $\lambda$=1, on-policy methods such as TD($\lambda$) become equivalent in their off-line forms to a Monte Carlo method (and approximately equivalent in their online forms). However, the same cannot be said for any off-policy method, including Watkins's Q($\lambda$), Peng's Q($\lambda$), GQ($\lambda$), or GTD($\lambda$) (Watkins 1989, Peng & Williams 1996, Maei & Sutton 2010, Maei 2011). All these methods choose to bootstrap completely, in effect setting $\lambda$ to 0 (and cutting off the eligibility traces), whenever the behaviour policy deviates from the target policy (i.e., whenever a non-max, exploratory action is taken). Because they bootstrap, these methods cannot be equivalent to a Monte Carlo method (which would only include complete actual outcomes). Up until last year this had seemed unavoidable to me; I could imagine no way in which an online algorithm could update its weights while the target policy was being followed and then, if there was a deviation from the target policy, go back and selectively "unmake" just the right updates now that the current outcome was no longer valid. And, even if it was somehow possible, I really could not imagine it could be done with a complexity and simplicity similar to existing TD methods. To my surprise, I now see how I believe this can be done. If this is successful, it would mean that the off-policy learning and the $\lambda$ parameter can be completely decoupled, that one could separately choose the degree of bootstrapping and the target and behaviour policies.

The equivalences mentioned above, between what are called the forward and backward views of the TD algorithms, are only exact for off-line methods, in which updates are computed "on the side" and the weights are only actually changed at the end of an episode. They hold only approximately if the weights are updated online at each step as the updates are computed. This has also seemed unavoidable to me, but recent work by my postdoc, Harm van Seijen, suggests that an exact forward-backward equivalence may be possible for the online case, again without significantly increasing computational complexity.

In closing, let me mention a few more areas in which core TD methods can be generalized and improved. One is simple generalization. The $\lambda$ parameter could not only be set to any value, it could be an entire function, with $\lambda$ taking on a different value for each state (this idea is mentioned by Sutton & Barto (1998) but it has yet to be developed). Another potentially powerful generalization is to let the "discount rate" parameter $\boldsymbol{\gamma}$ also be a function of state. Here $\boldsymbol{\gamma}$ should be thought of not as a discount, but

as a degree of termination, or better, as a horizon on the prediction—rewards are predicted up until they are sunset-ed by $\gamma$ falling or multiplying to zero. With these and other generalizations (see Modayil, White & Sutton, in press, Section 5) we have been able to turn computationally simple and uniform TD algorithms into a surprisingly expressive representation language for capturing information about the dynamics of the world that can be used, like "options" (Sutton, Precup & Singh 1999) in dynamic-programming style planning methods.

**Impact**

Computationally efficient model-free TD learning methods form the core of modern reinforcement learning, which is itself a core technology for modern artificial intelligence. Many might believe that there is little more to be done with model-free TD methods, that they have already reached the limit of their usefulness. I could not disagree more. As we seek fundamental principles of learning and intelligence, we should seek above all to strengthen the core algorithms. The better we understand these components, and the more we will be able to do reliably and scalably with them, and the more powerful systems we will be able to build with them.

In the field of reinforcement learning I think there is a widespread realization of the importance of core TD methods. TD(λ), Sarsa(λ), and Q(λ) are very widely used today, and I believe this is due most of all to their reliability, computational simplicity, and relatively clear theory. If replacement algorithms are proposed that do not compromise these positive features, they are likely to become widely accepted and used. Neuroscientists also appreciate the simplicity of TD algorithms and may look for biological analogs of the new methods if they are stated and explained simply.

The greatest long-term impact of the proposed work would be if we succeeded in putting forth general, uniform TD algorithms that are capable of learning and expressing general world knowledge, compatible with planning methods. This has been a long-standing goal of my research group (Sutton 2012, 2009) which we have not yet achieved. As we strive toward it, working with robots or with computational worlds such as gridworlds, we keep being pushed back toward refining the basic toolkit of core TD algorithms. Each time we make them fundamentally more general, powerful, or robust, each time we identify a flaw and correct it, we come closer to the point at which the learning of world knowledge can be demonstrated with enough generality to capture the imagination of the research community. I don't know when this will happen, but it is coming closer.

**Methodology**

Much of the research and algorithm development in my lab is conceptual and done on paper and whiteboard. Simple worlds are imagined as well as the corresponding behaviour of the algorithm. At a certain point we switch to formal analysis and computations, leading ultimately to formal proofs of convergence or equivalences. We also make extensive use of computational microworlds, small imaginary worlds that are completely understood as Markov decision processes and that can be used to test learning algorithms and to compare their performance. The members of my group get extensive training in the appropriate way to vary step-size and other algorithm parameters to permit fair comparisons.

In the last five years, we have added a substantial robotic component to our methodology. We still focus on relatively small and simple environments—*robot microworlds* we call them. Most of our robots simply roll around low to the ground and sense their local environment. One of our robots is custom build and has a few dozen sensors, but no cameras or auditory processing. Most of the others are iRobot creates, the hobbyist version of the iRobot Roomba without the vacuum cleaner. These have even fewer sensors, providing a minimalist interface with lots of hidden state. Robot microworlds bring us all the worthy complexities of real-time behaviour, including rapid response and the need for computational simplicity, yet are simple and reliable enough to bring in a minimum of additional distractions.

We also sometimes challenge our TD learning algorithms with a real application of importance to individuals and society: assisting amputees with interfaces to their prosthetic arms (e.g., Pilarski et al. 2013).