Reinforcement learning

From Wikipedia, the free encyclopedia

Reinforcement learning is an area of machine learning inspired by behaviorist psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The problem, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics, and genetic algorithms. In the operations research and control literature, the field where reinforcement learning methods are studied is called approximate dynamic programming. The problem has been studied in the theory of optimal control, though most studies are concerned with the existence of optimal solutions and their characterization, and not with the learning or approximation aspects. In economics and game theory, reinforcement learning may be used to explain how equilibrium may arise under bounded rationality.

In machine learning, the environment is typically formulated as a Markov decision process (MDP) as many reinforcement learning algorithms for this context utilize dynamic programming techniques. The main difference between the classical techniques and reinforcement learning algorithms is that the latter do not need knowledge about the MDP and they target large MDPs where exact methods become infeasible.

Reinforcement learning differs from standard supervised learning in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. Further, there is a focus on on-line performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). The exploration vs. exploitation trade-off in reinforcement learning has been most thoroughly studied through the multi-armed bandit problem and in finite MDPs.

Contents

Introduction

The basic reinforcement learning model consists of:

1. a set of environment states S;
2. a set of actions A;
3. rules of transitioning between states;
4. rules that determine the scalar immediate reward of a transition; and
5. rules that describe what the agent observes.

The rules are often stochastic. The observation typically involves the scalar immediate reward associated with the last transition. In many works, the agent is also assumed to observe the current environmental state, in which case we talk about full observability, whereas in the opposing case we talk about partial observability. Sometimes the set of actions available to the agent is restricted (e.g., you cannot spend more money than what you possess).

A reinforcement learning agent interacts with its environment in discrete time steps. At each time t, the agent receives an observation o_t, which typically includes the reward r_t. It then chooses an action a_t from the set of actions available, which is subsequently sent to the environment. The environment moves to a new state s_{t+1} and the reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}) is determined. The goal of a reinforcement learning agent is to collect as much reward as possible. The agent can choose any action as a function of the history and it can even randomize its action selection.

When the agent's performance is compared to that of an agent which acts optimally from the beginning, the difference in performance gives rise to the notion of regret. Note that in order to act near optimally, the agent must reason about the

long term consequences of its actions: In order to maximize my future income I had better go to school now, although the immediate monetary reward associated with this might be negative.

Thus, reinforcement learning is particularly well suited to problems which include a long-term versus short-term reward trade-off. It has been applied successfully to various problems, including robot control, elevator scheduling, telecommunications, backgammon, checkers (Sutton and Barto 1998, Chapter 11) and go (AlphaGo).

Two components make reinforcement learning powerful: The use of samples to optimize performance and the use of function approximation to deal with large environments. Thanks to these two key components, reinforcement learning can be used in large environments in any of the following situations:

- A model of the environment is known, but an analytic solution is not available;
- Only a simulation model of the environment is given (the subject of simulation-based optimization);[1]
- The only way to collect information about the environment is by interacting with it.

The first two of these problems could be considered planning problems (since some form of the model is available), while the last one could be considered as a genuine learning problem. However, under a reinforcement learning methodology both planning problems would be converted to machine learning problems.

Exploration

The reinforcement learning problem as described requires clever exploration mechanisms. Randomly selecting actions, without reference to an estimated probability distribution, is known to give rise to very poor performance. The case of (small) finite MDPs is relatively well understood by now. However, due to the lack of algorithms that would provably scale well with the number of states (or scale to problems with infinite state spaces), in practice people resort to simple exploration methods. One such method is ϵ-greedy, when the agent chooses the action that it believes has the best long-term effect with probability $1-\epsilon$, and it chooses an action uniformly at random, otherwise. Here, $0 < \epsilon < 1$ is a tuning parameter, which is sometimes changed, either according to a fixed schedule (making the agent explore less as time goes by), or adaptively based on some heuristics.

Algorithms for control learning

Even if the issue of exploration is disregarded and even if the state was observable (which we assume from now on), the problem remains to find out which actions are good based on past experience.

Criterion of optimality

For simplicity, assume for a moment that the problem studied is episodic, an episode ending when some terminal state is reached. Assume further that no matter what course of actions the agent takes, termination is inevitable. Under some additional mild regularity conditions the expectation of the total reward is then well-defined,

for any policy and any initial distribution over the states. Here, a policy refers to a mapping that assigns some probability distribution over the actions to all possible histories.

Given a fixed initial distribution μ, we can thus assign the expected return ρ^π to policy π:

$$\rho^\pi = E[R|\pi],$$

where the random variable R denotes the return and is defined by

$$R = \sum_{t=0}^{N-1} r_{t+1},$$

where r_{t+1} is the reward received after the t-th transition, the initial state is sampled at random from μ and actions are selected by policy π. Here, N denotes the (random) time when a terminal state is reached, i.e., the time when the episode terminates.

In the case of non-episodic problems the return is often discounted,

$$R = \sum_{t=0}^{\infty} \gamma^t r_{t+1},$$

giving rise to the total expected discounted reward criterion. Here $0 \leq \gamma \leq 1$ is the so-called discount-factor. Since the undiscounted return is a special case of the discounted return, from now on we will assume discounting. Although this looks innocent enough, discounting is in fact problematic if one cares about online performance. This is because discounting makes the initial time steps more important. Since a learning agent is likely to make mistakes during the first few steps after its "life" starts, no uninformed learning algorithm can achieve near-optimal performance under discounting even if the class of environments is restricted to that of finite MDPs. (This does not mean though that, given enough time, a learning agent cannot figure how to act near-optimally, if time was restarted.)

The problem then is to specify an algorithm that can be used to find a policy with maximum expected return. From the theory of MDPs it is known that, without loss of generality, the search can be restricted to the set of the so-called stationary policies. A policy is called stationary if the action-distribution returned by it depends only on the last state visited (which is part of the observation history of the agent, by our simplifying assumption). In fact, the search can be further restricted to deterministic stationary policies. A deterministic stationary policy is one which deterministically selects actions based on the current state. Since any such policy can be identified with a mapping from the set of states to the set of actions, these policies can be identified with such mappings with no loss of generality.

Brute force

The brute force approach entails the following two steps:

1. For each possible policy, sample returns while following it
2. Choose the policy with the largest expected return

One problem with this is that the number of policies can be extremely large, or even infinite. Another is that variance of the returns might be large, in which case a large number of samples will be required to accurately estimate the return of each policy.

These problems can be ameliorated if we assume some structure and perhaps allow samples generated from one policy to influence the estimates made for another. The two main approaches for achieving this are value function estimation and direct policy search.

Value function approaches

Value function approaches attempt to find a policy that maximizes the return by maintaining a set of estimates of expected returns for some policy (usually either the "current" or the optimal one).

These methods rely on the theory of MDPs, where optimality is defined in a sense which is stronger than the above one: A policy is called optimal if it achieves the best expected return from any initial state (i.e., initial distributions play no role in this definition). Again, one can always find an optimal policy amongst stationary policies.

To define optimality in a formal manner, define the value of a policy π by

$$V^{\pi}(s) = E[R|s,\pi],$$

where R stands for the random return associated with following π from the initial state s. Define $V^*(s)$ as the maximum possible value of $V^{\pi}(s)$, where π is allowed to change:

$$V^*(s) = \sup_{\pi} V^{\pi}(s).$$

A policy which achieves these optimal values in each state is called optimal. Clearly, a policy that is optimal in this strong sense is also optimal in the sense that it maximizes the expected return ρ^{π}, since $\rho^{\pi} = E[V^{\pi}(S)]$, where S is a state randomly sampled from the distribution μ.

Although state-values suffice to define optimality, it will prove to be useful to define action-values. Given a state s, an action a and a policy π, the action-value of the pair (s,a) under π is defined by

$$Q^{\pi}(s,a) = E[R|s,a,\pi],$$

where, now, R stands for the random return associated with first taking action a in state s and following π, thereafter.

It is well-known from the theory of MDPs that if someone gives us Q for an optimal policy, we can always choose optimal actions (and thus act optimally) by simply choosing the action with the highest value at each state. The action-value function

of such an optimal policy is called the optimal action-value function and is denoted by Q^*. In summary, the knowledge of the optimal action-value function alone suffices to know how to act optimally.

Assuming full knowledge of the MDP, there are two basic approaches to compute the optimal action-value function, value iteration and policy iteration. Both algorithms compute a sequence of functions Q_k ($k = 0, 1, 2, \ldots,$) which converge to Q^*. Computing these functions involves computing expectations over the whole state-space, which is impractical for all, but the smallest (finite) MDPs, never mind the case when the MDP is unknown. In reinforcement learning methods the expectations are approximated by averaging over samples and one uses function approximation techniques to cope with the need to represent value functions over large state-action spaces.

Monte Carlo methods

The simplest Monte Carlo methods can be used in an algorithm that mimics policy iteration. Policy iteration consists of two steps: policy evaluation and policy improvement. The Monte Carlo methods are used in the policy evaluation step. In this step, given a stationary, deterministic policy π, the goal is to compute the function values $Q^\pi(s,a)$ (or a good approximation to them) for all state-action pairs (s,a). Assume (for simplicity) that the MDP is finite and in fact a table representing the action-values fits into the memory. Further, assume that the problem is episodic and after each episode a new one starts from some random initial state. Then, the estimate of the value of a given state-action pair (s,a)can be computed by simply averaging the sampled returns which originated from (s,a) over time. Given enough time, this procedure can thus construct a precise estimate Q of the action-value function Q^π. This finishes the description of the policy evaluation step. In the policy improvement step, as it is done in the standard policy iteration algorithm, the next policy is obtained by computing a greedy policy with respect to Q: Given a state s, this new policy returns an action that maximizes $Q(s,\cdot)$. In practice one often avoids computing and storing the new policy, but uses lazy evaluation to defer the computation of the maximizing actions to when they are actually needed.

A few problems with this procedure are as follows:

- The procedure may waste too much time on evaluating a suboptimal policy;
- It uses samples inefficiently in that a long trajectory is used to improve the estimate only of the single state-action pair that started the trajectory;
- When the returns along the trajectories have high variance, convergence will be slow;
- It works in episodic problems only;
- It works in small, finite MDPs only.

Temporal difference methods

The first issue is easily corrected by allowing the procedure to change the policy (at all, or at some states) before the values settle. However good this sounds, this may be problematic as this might prevent convergence. Still, most current algorithms implement this idea, giving rise to the class of generalized policy iteration algorithm. We note in passing that actor critic methods belong to this category.

The second issue can be corrected within the algorithm by allowing trajectories to contribute to any state-action pair in them. This may also help to some extent with the third problem, although a better solution when returns have high variance is to use Sutton's temporal difference (TD) methods which are based on the recursive Bellman equation. Note that the computation in TD methods can be incremental (when after each transition the memory is changed and the transition is thrown away), or batch (when the transitions are collected and then the estimates are computed once based on a large number of transitions). Batch methods, a prime example of which is the least-squares temporal difference method due to Bradtke and Barto (1996), may use the information in the samples better, whereas incremental methods are the only choice when batch methods become infeasible due to their high computational or memory complexity. In addition, there exist methods that try to unify the advantages of the two approaches. Methods based on temporal differences also overcome the second but last issue.

In order to address the last issue mentioned in the previous section, function approximation methods are used. In linear function approximation one starts with a mapping ϕ that assigns a finite-dimensional vector to each state-action pair. Then, the action values of a state-action pair (s,a) are obtained by linearly combining the components of $\phi(s,a)$ with some weights θ:

$$Q(s,a) = \sum_{i=1}^{d} \theta_i \phi_i(s,a).$$

The algorithms then adjust the weights, instead of adjusting the values associated with the individual state-action pairs. However, linear function approximation is not the only choice. More recently, methods based on ideas from nonparametric statistics (which can be seen to construct their own features) have been explored.

So far, the discussion was restricted to how policy iteration can be used as a basis of the designing reinforcement learning algorithms. Equally importantly, value iteration can also be used as a starting point, giving rise to the Q-Learning algorithm (Watkins 1989) and its many variants.

The problem with methods that use action-values is that they may need highly precise estimates of the competing action values, which can be hard to obtain when the returns are noisy. Though this problem is mitigated to some extent by temporal difference methods and if one uses the so-called compatible function approximation method, more work remains to be done to increase generality and efficiency. Another problem specific to temporal difference methods comes from their reliance on the recursive Bellman equation. Most temporal difference methods have a so-called λ parameter $(0 \leq \lambda \leq 1)$ that allows one to continuously interpolate between Monte-Carlo methods (which do not rely on the Bellman equations) and the basic temporal difference methods (which rely entirely on the Bellman equations), which can thus be effective in palliating this issue.

Direct policy search

An alternative method to find a good policy is to search directly in (some subset of) the policy space, in which case the problem becomes an instance of stochastic optimization. The two approaches available are gradient-based and gradient-free

methods.

Gradient-based methods (giving rise to the so-called policy gradient methods) start with a mapping from a finite-dimensional (parameter) space to the space of policies: given the parameter vector θ, let π_θ denote the policy associated to θ. Define the performance function by

$$\rho(\theta) = \rho^{\pi_\theta}.$$

Under mild conditions this function will be differentiable as a function of the parameter vector θ. If the gradient of ρ was known, one could use gradient ascent. Since an analytic expression for the gradient is not available, one must rely on a noisy estimate. Such an estimate can be constructed in many ways, giving rise to algorithms like Williams' REINFORCE method (which is also known as the likelihood ratio method in the simulation-based optimization literature). Policy gradient methods have received a lot of attention in the last couple of years (e.g., Peters et al. (2003)), but they remain an active field. An overview of policy search methods in the context of robotics has been given by Deisenroth, Neumann and Peters.[2] The issue with many of these methods is that they may get stuck in local optima (as they are based on local search).

A large class of methods avoids relying on gradient information. These include simulated annealing, cross-entropy search or methods of evolutionary computation. Many gradient-free methods can achieve (in theory and in the limit) a global optimum. In a number of cases they have indeed demonstrated remarkable performance.

The issue with policy search methods is that they may converge slowly if the information based on which they act is noisy. For example, this happens when in episodic problems the trajectories are long and the variance of the returns is large. As argued beforehand, value-function based methods that rely on temporal differences might help in this case. In recent years, several actor-critic algorithms have been proposed following this idea and were demonstrated to perform well in various problems.

Theory

The theory for small, finite MDPs is quite mature. Both the asymptotic and finite-sample behavior of most algorithms is well-understood. As mentioned beforehand, algorithms with provably good online performance (addressing the exploration issue) are known. The theory of large MDPs needs more work. Efficient exploration is largely untouched (except for the case of bandit problems). Although finite-time performance bounds appeared for many algorithms in the recent years, these bounds are expected to be rather loose and thus more work is needed to better understand the relative advantages, as well as the limitations of these algorithms. For incremental algorithm asymptotic convergence issues have been settled. Recently, new incremental, temporal-difference-based algorithms have appeared which converge under a much wider set of conditions than was previously possible (for example, when used with arbitrary, smooth function approximation).

Current research

Current research topics include: adaptive methods which work with fewer (or no) parameters under a large number of conditions, addressing the exploration problem in large MDPs, large-scale empirical evaluations, learning and acting under partial information (e.g., using Predictive State Representation), modular and hierarchical reinforcement learning, improving existing value-function and policy search methods, algorithms that work well with large (or continuous) action spaces, transfer learning, lifelong learning, efficient sample-based planning (e.g., based on Monte-Carlo tree search). Multiagent or Distributed Reinforcement Learning is also a topic of interest in current research. There is also a growing interest in real life applications of reinforcement learning. Successes of reinforcement learning are collected on here (http://umichrl.pbworks.com/Successes-of-Reinforcement-Learning/) and here (http://rl-community.org/wiki/Successes_Of_RL).

Reinforcement learning algorithms such as TD learning are also being investigated as a model for Dopamine-based learning in the brain. In this model, the dopaminergic projections from the substantia nigra to the basal ganglia function as the prediction error. Reinforcement learning has also been used as a part of the model for human skill learning, especially in relation to the interaction between implicit and explicit learning in skill acquisition (the first publication on this application was in 1995-1996, and there have been many follow-up studies). See http://webdocs.cs.ualberta.ca/~sutton/RL-FAQ.html#behaviorism for further details of these research areas above.

Implementations

- RL-Glue (http://glue.rl-community.org/) provides a standard interface that allows you to connect agents, environments, and experiment programs together, even if they are written in different languages.
- Maja Machine Learning Framework (http://mmlf.sourceforge.net/) The Maja Machine Learning Framework (MMLF) is a general framework for problems in the domain of Reinforcement Learning (RL) written in python.
- Software Tools for Reinforcement Learning (Matlab and Python) (http://jamh-web.appspot.com/download.htm)
- PyBrain(Python) (http://www.pybrain.org/)
- TeachingBox (http://servicerobotik.hs-weingarten.de/en/teachingbox.php) is a Java reinforcement learning framework supporting many features like RBF networks, gradient descent learning methods, ...
- C++ and Python implementations (http://webdocs.cs.ualberta.ca/~vanhasse/code.html) for some well known reinforcement learning algorithms with source.
- Orange, a free data mining software suite, module orngReinforcement (http://www.ailab.si/orange/doc/modules/orngReinforcement.htm)
- Policy Gradient Toolbox (http://www.ias.informatik.tu-darmstadt.de/Research/PolicyGradientToolbox) provides a package for learning about policy gradient approaches.
- BURLAP (http://burlap.cs.brown.edu) is an open source Java library that provides a wide range of single and multi-agent learning and planning methods.

Inverse reinforcement learning

In inverse reinforcement learning (IRL), no reward function is given. Instead, one tries to extract a policy given an observed behavior, in order to mimic the observed behavior which is often optimal or close to optimal. Since an agent learning by inverse reinforcement learning, once it has deviated from the track followed by the observed behavior, often needs some way to get back on the track in order for its own behavior to be stable, it is sometimes necessary to demonstrate the behavior multiple times with small perturbations each time.

In apprenticeship learning, one assumes that an expert demonstrating a behavior is trying to maximize a reward function, and tries to recover the unknown reward function of the expert.

See also

- Temporal difference learning
- Q-learning
- SARSA
- Fictitious play
- Learning classifier system
- Optimal control
- Dynamic treatment regimes
- Error-driven learning
- Multi-agent system
- Distributed artificial intelligence

References

- Sutton, Richard S. (1984). Temporal Credit Assignment in Reinforcement Learning (PhD thesis). University of Massachusetts, Amherst, MA.
- Williams, Ronald J. (1987). "A class of gradient-estimating algorithms for reinforcement learning in neural networks". Proceedings of the IEEE First International Conference on Neural Networks.
- Sutton, Richard S. (1988). "Learning to predict by the method of temporal differences". Machine Learning (Springer) 3: 9–44. doi:10.1007/BF00115009.
- Watkins, Christopher J.C.H. (1989). Learning from Delayed Rewards (PDF) (PhD thesis). King's College, Cambridge, UK.
- Bradtke, Steven J.; Andrew G. Barto (1996). "Learning to predict by the method of temporal differences". Machine Learning (Springer) 22: 33–57. doi:10.1023/A:1018056104778.
- Bertsekas, Dimitri P.; John Tsitsiklis (1996). Neuro-Dynamic Programming. Nashua, NH: Athena Scientific. ISBN 1-886529-10-8.
- Kaelbling, Leslie P.; Michael L. Littman; Andrew W. Moore (1996). "Reinforcement Learning: A Survey". Journal of Artificial Intelligence Research 4: 237–285.
- Sutton, Richard S.; Barto, Andrew G. (1998). Reinforcement Learning: An Introduction. MIT Press. ISBN 0-262-19398-1.
- Peters, Jan; Sethu Vijayakumar; Stefan Schaal (2003). "Reinforcement Learning for Humanoid Robotics" (PDF). IEEE-RAS International Conference on Humanoid Robots.
- Powell, Warren (2007). Approximate dynamic programming: solving the curses of dimensionality. Wiley-Interscience. ISBN 0-470-17155-3.
- Auer, Peter; Thomas Jaksch; Ronald Ortner (2010). "Near-optimal regret bounds

for reinforcement learning". Journal of Machine Learning Research 11: 1563–1600. Missing |last2= in Authors list (help)
- Szita, Istvan; Csaba Szepesvari (2010). "Model-based Reinforcement Learning with Nearly Tight Exploration Complexity Bounds" (PDF). ICML 2010. Omnipress. pp. 1031–1038.
- Bertsekas, Dimitri P. (August 2010). "Chapter 6 (online): Approximate Dynamic Programming". Dynamic Programming and Optimal Control (PDF) II (3 ed.).
- Busoniu, Lucian; Robert Babuska; Bart De Schutter; Damien Ernst (2010). Reinforcement Learning and Dynamic Programming using Function Approximators. Taylor & Francis CRC Press. ISBN 978-1-4398-2108-4.
- Deisenroth, Marc Peter; Gerhard Neumann; Jan Peters (2013). A Survey on Policy Search for Robotics. Foundations and Trends in Robotics 2. NOW Publishers. pp. 1–142.

1. Gosavi, Abhijit (2003). Simulation-based Optimization: Parametric Optimization Techniques and Reinforcement. Springer. ISBN 1-4020-7454-9.
2. Deisenroth, Marc Peter; Neumann, Gerhard; Peters, Jan (2013). A Survey on Policy Search for Robotics. NOW Publishers. pp. 1–142. ISBN 978-1-60198-702-0.

Literature

Conferences, journals

Most reinforcement learning papers are published at the major machine learning and AI conferences (ICML, NIPS, AAAI, IJCAI, UAI, AI and Statistics) and journals (JAIR (http://www.jair.org), JMLR (http://www.jmlr.org), Machine learning journal (http://www.springer.com/computer/ai/journal/10994), IEEE T-CIAIG (http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=4804728)). Some theory papers are published at COLT and ALT. However, many papers appear in robotics conferences (IROS, ICRA) and the "agent" conference AAMAS. Operations researchers publish their papers at the INFORMS conference and, for example, in the Operation Research (http://or.pubs.informs.org), and the Mathematics of Operations Research (http://mor.pubs.informs.org) journals. Control researchers publish their papers at the CDC and ACC conferences, or, e.g., in the journals IEEE Transactions on Automatic Control (http://www.nd.edu/~ieeetac/), or Automatica (http://www.elsevier.com/locate/automatica), although applied works tend to be published in more specialized journals. The Winter Simulation Conference (http://www.wintersim.org/) also publishes many relevant papers. Other than this, papers also published in the major conferences of the neural networks, fuzzy, and evolutionary computation communities. The annual IEEE symposium titled Approximate Dynamic Programming and Reinforcement Learning (ADPRL) and the biannual European Workshop on Reinforcement Learning (EWRL (http://ewrl.wordpress.com/)) are two regularly held meetings where RL researchers meet.

External links

- Website for Reinforcement Learning: An Introduction (http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html) (1998), by Rich Sutton and Andrew Barto, MIT Press, including a link to an html version of the book.
- Reinforcement Learning Repository (http://www-anw.cs.umass.edu/rlr/)
- Reinforcement Learning and Artificial Intelligence (http://spaces.facsci.ualberta.ca/rlai/) (RLAI, Rich Sutton's lab at the University of Alberta)
- Autonomous Learning Laboratory (http://www-all.cs.umass.edu/) (ALL, Andrew

Barto's lab at the University of Massachusetts Amherst)
- RL-Glue (http://glue.rl-community.org)
- Software Tools for Reinforcement Learning (Matlab and Python) (http://jamh-web.appspot.com/download.htm)
- The Reinforcement Learning Toolbox from the (Graz University of Technology) (http://www.igi.tugraz.at/ril-toolbox)
- Hybrid reinforcement learning (http://www.cogsci.rpi.edu/~rsun/hybrid-rl.html)
- Piqle: a Generic Java Platform for Reinforcement Learning (http://sourceforge.net/projects/piqle/)
- A Short Introduction To Some Reinforcement Learning Algorithms (http://webdocs.cs.ualberta.ca/~vanhasse/rl_algs/rl_algs.html)
- Reinforcement Learning applied to Tic-Tac-Toe Game (http://www.lwebzem.com/cgi-bin/ttt/ttt.html)
- Scholarpedia Reinforcement Learning (http://www.scholarpedia.org/article/Reinforcement_Learning)
- Scholarpedia Temporal Difference Learning (http://www.scholarpedia.org/article/Temporal_difference_learning)
- Stanford Reinforcement Learning Course (http://www.troovoo.com/vid.php?a=Stanford&c=Machine+Learning&l=Applications+of+Reinforcement+Learning)
- Real-world reinforcement learning experiments (http://www.dcsc.tudelft.nl/~robotics/media.html) at Delft University of Technology
- Reinforcement Learning Tools for Matlab (http://busoniu.net/repository.php)
- Stanford University Andrew Ng Lecture on Reinforcement Learning (https://www.youtube.com/watch?v=RtxI449ZjSc&feature=relmfu)

Categories: Markov models | Machine learning algorithms | Belief revision

---