REINFORCEMENT LEARNING IN COMPLEX REAL WORLD DOMAINS: A REVIEW

Samiksha Mahajan
Ex Assistant Professor
Dept of Information Technology
VVES's Vikas College,
Vikhroli, Mumbai 400 083, India
mahajan.samiksha@gmail.com

Abstract

Reinforcement Learning is an area of Machine Learning inspired by behaviorist psychology based on the mechanism of learning from rewards. RL does not require prior knowledge and automatically get optimal policy with the help of knowledge obtained by trial-and-error and continuous interaction with the dynamic environment. In complex real world domains implementing RL algorithms is the major practical problem due to the large and continuous space. It can give rise to problems like Curse of Dimensionality, Partial Observability Problem, Credit Structuring Problem, Generalization and Exploration-Exploitation Dilemma. This paper gives an introduction to Reinforcement Learning, discusses its basic model and system structure, and discusses the problems faced while implementing RL algorithms in complex real world domains. At last but not the least this paper briefly describes the techniques which can make the working of RL process easier in the complex domains. It concludes with research scope of RL in complex real world.

Keywords**:** *Reinforcement Learning, Machine Learning, Large States, Continuous Action Spaces.*

1. Introduction

In Artificial Intelligence the aim is to build intelligent entities [1]. The research field of Machine Learning constitutes a subset of the broader field of Artificial Intelligence. ML researchers investigate how to construct algorithms that allows a computer to learn from observations [2][3]. In Machine Learning the model of learning technology can be of 3 different types:

- Supervised Learning (Learn the model)
- Unsupervised Learning (Learn the representation)
- Reinforcement Learning (Learn to control)

RL is in between these two approaches. There is no omniscient teacher, but the agent gets a feedback about the quality of its actions. Thus, RL is based on the interaction of an agent who executes an action and its environment who gives positive or negative feedback. RL is successfully studied in many other disciplines such as Game theory (e.g. Backgammon, Chess, Solitaire, Checkers), Control theory (e.g. helicopter control), Operation research (e.g. Vehicle routing, Pricing, Targeted marketing), Robotics (e.g. Robot Soccer, Air Hockey, Quadruped Gait Control, Quadruped ball acquisition), Economics (e.g. Trading) , Information theory, Simulation based optimization, Statistics and Genetic algorithms

The basic methods in reinforcement learning are based on the classical dynamic programming algorithms that were developed in the late 1950s [28, 29]. However, reinforcement learning methods have over two important advantages over classical dynamic programming. First, the methods are online. This permits them to focus their attention on the parts of the state space that are important and to ignore the rest of the space. Second, the methods can employ function approximation algorithms (e.g., neural networks) to represent their knowledge. This allows them to generalize across the state space so that the learning time scales much better.

Current line of research is to establish highly data efficient and robust RL algorithms that are able to solve real world problems. Complex real world domains generally have very large, continuous action spaces, have hidden states and delayed rewards problem which presents a major obstacle in successfully applying RL to these domains. There come the problems like Curse of Dimensionality, Partial Observability, Temporal Credit assignment, Generalization problem, Exploration-exploitation trade off. These problems can be are avoided and efficient implementation can be done with the help of techniques like Function Approximation, Hierarchical RL, Relational RL, State Aggregation Methods, Finite Strategy Space Searching Methods , Kernel Based Methods, Multi grid Algorithms, Factored Representation, Shaping etc.

2. Reinforcement learning model & basic structure

The RL model consists of:

i. A discrete set of environment states S
ii. A discrete set of agent action A
iii. A set of scalar reinforcement signals {0, 1} or the real numbers.

The basic structure of RL is as shown in Figure.1 where, the agent interacts with an environment with the help of sensor data, changing the environment and obtaining a reward for his action. What is outside the agent is considered as environment. The states are parameters or features that describe the environment. The sensor data determine states. Being in a given state S, the value function assesses the different actions that can be taken.

It is therefore some kind of prediction of the reward the environment will probably give. An RL agent senses the environment and learns the optimal policy or near optimal policy by taking actions in each state of the environment. Agent in RL is quite simple consisting of policy of behavior and learning algorithm to adapt this policy. Policy can be considered as a function that maps situation to actions. The agent chooses an action in each state, which may take agent to a new state and receives a reward. The agent will simply learn how to optimize the rewards he gets. It depends on the reward function whether his solution complies with the desired behavior.

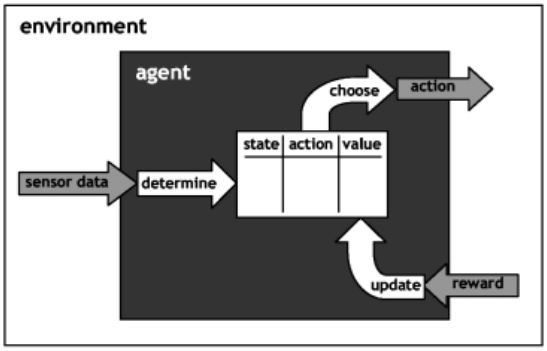

Figure.1: Structure of RL System (source: http://www.ensta-paristech.fr/)

The agent eventually learns the best action to perform for obtaining the maximum expected accumulated reward by repeating this process. In each iteration, the agent perceives its current state ($s \in S$), select an action ($a \in A$), possibly changing its state, and receives a reward signal($r \in R$). In this process, the agent needs to obtain useful experiences regarding states, actions, state transitions and rewards to act optimally and the evaluation of the system occur concurrently with learning process. If the probabilities or rewards are unknown, the problem is one of RL. [1]

RL's main objective is to learn how to map states to actions while maximizing a reward signal. In RL, an autonomous agent follows trial and error process to learn the optimal action to perform in each state in order to reach its goals.

Strength of RL is that it does not require explicit examples of executing correct action. [16] It can be applied to system for which such examples are not readily available. E.g. Dynamic channel assignment in communications networks [22], to construct fuzzy logic rule bases for fuzzy control systems [23]. A well-known example for the successful application of RL technique is the back-gammon computer developed by Tesauro. [24]It consists of a feed forward network, which is trained by playing against itself. TD-backgammon outperformed all backgammon programs available at that time. There exist RL algorithms such as sparse-sampling, approximate value iteration, approximate policy iteration, policy gradient, conservative policy iteration, etc., whose complexity is independent of the size of the state space.

There are three main approaches how to solve the general reinforcement learning problem. Each of these methods has strengths and weaknesses:

Dynamic programming methods require a complete and accurate model of the environment while being well developed from a mathematical point of view. The conceptually simple Monte Carlo methods, in contrast, don't need a model, but cannot be used for incremental computation. The Temporal Difference (TD) methods don't have this disadvantage, but they are quite complex to analyze.

3. Concepts related to reinforcement learning

3.1 Markov decision process (MDPs)

In reinforcement learning environments, states are often assumed to fulfill the Markov property. If this condition is met, it's possible to predict the next state and the next reward using only the current state and action. MDP is the probabilistic model of a sequential decision problem, where states can be perceived exactly, and the current state and action selected determine a probability distribution on future states. The model is Markov if the state transition are independent of any previous environment states or agent actions.[9] Many previous works in RL are limited to MDP which are excellent models for delayed RL tasks with uncertainty and discrete state transition. [18] For large and continuous state or action space, RL agents have to incorporate some form of generalization e.g. using general function approximators to represent value function or control policies. Hidden state problem arises in that case that RL agents cannot observer the state of environment perfectly owing to noisy or insufficient sensors, partial information etc. Partially observable Markov Decision Process is an appropriate model for hidden state problems.

3.2 Model based and model free methods

There are 2 Reinforcement Learning strategies namely, Model Based and Model Free.

3.2.1 Model free RL

Model free RL methods learn how to act without explicitly learning the transition probabilities. These methods directly learn a value function without requiring knowledge of consequence of doing actions. E.g. Q learning, SARSA (λ) TD (λ).These are value based methods and learn Q function. They never learn transition probabilities and reward function. At the end of learning, agent knows how to act, but doesn't explicitly know anything about the environment.

Q-learning [19] is the most popular and most effective model-free algorithm for learning from delayed environment. Q-learning can be applied to discounted infinite-horizon MDPs. Q-learning can also be applied to undiscounted problems if the optimal policy is guaranteed to reach a reward-free state and the state is periodically reset.

TD algorithms are the class of learning methods based on idea of comparing temporally successive predictions, possibly the single most fundamental idea in all of RL. TD(λ) [4] This algorithm was famously applied to create TD-Gammon, a program that learned to play the game of backgammon at the level of expert human players.[20]

SARSA algorithms, as the name simply reflects the fact that the main function for updating the Q-value depends on the current state of the agent S_1, the action the agent chooses A_1, the reward R the agent gets for choosing this action, the state S_2 that the agent will now be in after taking that action, and finally the next action A_2 the agent will choose in its new state. Taking every letter in the quintuple (s_t, a_t, r_t, s_{t+1}, a_{t+1}) yields the word SARSA.

3.2.2 Model based RL

Model Based RL computes value functions using a model of system dynamics e.g. Adaptive Real time DP. Basic idea behind Model based RL is to learn the model of the MDP (transition probabilities and rewards) and learn how to act (solve the MDP) simultaneously and find optimal policy. Its working can be described as follows:

-While learning the model, it keeps track of how many times states s' follows state s when you take action a and update the transition probability P(s'|s, a) according to the relative frequencies. Also it keeps track of rewards R(s).

-While learning how to act, it estimates the utilities U(s) using Bellman's equations and choose the action that maximizes expected future utility π^*.

Model based methods explicitly estimate a model from experience and use dynamic-programming algorithms on the approximated model. They make effective use of experience and have high computational costs. E.g. Sutton's Dyna[51], Real Time Dynamic Programming, Queue Dyna, Plexus Planning.

3.3 Value andpolicy iteration

Solving an MDP is finding an optimal policy. For every state, there is no other action that gets a higher sum of discounted future rewards which is known as Optimal Policy. For every MDP there exists an optimal policy. A specific policy converts an MDP into a plain Markov system with rewards.
The optimal policy for state s gives the action A that maximizes the optimal value function for that state. $\pi^*$is the optimal policy function for all states S. Two standard methods for finding optimal policy are:

- Value Iteration: Zero-initialize and iteratively refine V(s) as it will converge towards V*(s). Finally use equation 1 to find the optimal policy π^*

- Policy Iteration: Random-initialize and iteratively refine $\pi(s)$ by alternating between computing V(s) and then $\pi(s)$. π eventually converges to the optimal policy π^*

In small state spaces, policy iteration is typically very fast and converges quickly but in case of large state spaces it may be slow because it needs to solve a large system of linear equations. Value iteration is preferred in such cases. For very large state spaces, value function can be approximated using some regression algorithm but the optimality guarantee is lost.

4. RL and problems in complex real world domains

Complex real world domains generally have very large states and continuous action spaces which presents a major obstacle in successfully applying RL to these domains. Also RL is expensive in data or learning steps turn around times for results are long. [25] This gives rise to following problems:

4.1 Curse of dimensionality

The curse of dimensionality is a term to describe the problem caused by the exponential increase in volume associated with adding extra dimensions to Euclidean space. [27] It causes explosions in state and action space and large number of possible next states of an action due to stochasticity [26] when trying to explore all possible actions from all possible states.[10]

4.2 Partially observable/hidden state problem

It often happens in real world scenario that RL agent will not know exactly in what state it will end up after performing an action. The agent does not have full and accurate perception of environment. RL algorithms require full observability; they cannot be used in this case. As states must be history independent, it must be irrelevant how one has to reach a certain state. So to address this problem of hidden states in real world domains, POMDP methods are used.

4.3 Temporal credit assignment problem

In fine grained state-action spaces, there is a problem of terribly temporally delayed rewards. Such reward signals will only very weakly affect all temporally distant states that have preceded it. The agent may normally perform many moves through its state-action space where immediate rewards are almost zero and where more relevant events are rather distant in the future. E.ga network designed to play chess would receive a reinforcement signal after a long sequence of moves. The question arises is: How do we assign credit/blame individually to each move in a sequence that leads to an eventual victory or loss? This is called the temporal credit assignment problem in contrast with the structural credit problem where we must attribute network error to different weight. It is often done by some form of RL [4]

4.4 Generalization problem

Generalization in RL is a hard problem that requires knowledge of great part of field. When the state-action space is small enough, the Q-function is stored in a table, with one entry for each state-action pair. In a continuous observation space it is impossible for agent to visit every state and store the value for this state in a table. As the number of possible states in an environment gets larger and larger, it becomes infeasible for an agent to visit all possible state enough times to find the optimal actions for those states.

4.5 Exploration vs. exploitation problem

The problem of Exploration-exploitation has been extensively studied in the case of k-armed bandits.[4] In some cases of real world domains, the agent faces a non-stationary environment, so it needs to choose the best moment to explore in order to adapt to the changes. However in some cases agent faces a relatively large state-action space and it therefore needs to choose the most promising subset of states/actions to explore. Finding the balance between obtaining as much reward as possible and sufficiently exploring the solution space is known as the Exploration versus Exploitation problem. MDPs with a large (or infinite) number of states are not solved within a reasonable computational time, or within a reasonable number of agent steps. Recently new class of algorithms, which include the E^3 [47], R-MAX [44], MBIE [45] and GCB [46] algorithms, were presented and proved to reach a nearly optimal policy (with high probability) within a time that is polynomial in the problem size.

5. Techniques for beating challenges in complex real world problems

RL still faces many practical issues related to the complexity of the environment in particular when dealing with large states or action sets. When the number of possible actions A is neither on the scale of ‘a few’ nor outright continuous, the situation becomes difficult. In order to solve highly complex problems in real world domains one should give up ‘tabula rasa’ learning techniques and begin to incorporate bias that will give leverage to learning process.[9]Biases like Shaping, Local Reinforcement Signals or Problem Decomposition can be used.[9] Naive table lookup methods are often employed to memorize the value for each individual state in small problems.

Many off shelf RL algorithms have poor scalability to some extent. They become increasingly expensive when there are large states or continuous action spaces. Current solutions include Function Approximation, Hierarchical RL, Relational RL, State Aggregation Methods, Transfer Learning, Finite Strategy Space Searching Methods, Kernel Based Methods, Multigrid Algorithms, Factored Representation, Shaping etc. Some of those are discussed out here.

5.1 Function approximation

Most real-world problems of practical interest have very large state spaces which render the tabular representation infeasible. When the number of states in an environment gets larger and larger, it becomes infeasible for an agent to visit all possible states. For example, the size of the state space of backgammon is estimated to be over 1020 [20]. So it becomes important to be able to generalize the learning experiences in a particular state to the other states in the environment. In order to avoid Curse of Dimensionality, for such large problems, compact representations (or approximation) have to be used, including decision trees, artificial neural networks, etc.

In function approximation the agent extracts the relevant feature vector to calculate the approximate value it gets from being in that state which is done by the dot product between the feature vector and parameter vector. The goal of RL with function approximation is to learn the best values for this parameter vector. [9]

Function approximation techniques allow the agent to generalize and handle large/infinite number of states. Many theoretical results are based on an assumption that the MDP is ergodic and every state is visited infinitely often. But in practice, “infinity” can never be achieved. Instead, the agent can first compute a good policy for a subspace of S, and then generalize the policy to other states that are visited less frequently. For problems with a continuous state space, a lookup table cannot be used because it is impossible to enumerate all states in a table and it is unlikely that a state will be visited more than once. In such a case, function approximation is even more critical to the generalization ability to unseen states. While function approximation has been used successfully in many cases, caution is required as convergence of the value function is not guaranteed for all generalizations. Defining the class of functions and conditions to ensure convergence is an active research topic.

There are number of function approximators for generalization purpose some of which are given as:

Fuzzy Function Approximators [11], Evolutionary Function Approximators [12] Value Function Approximators [14], TD based Neural Function Approximators [15], Fuzzy Neural Networks [13] etc.

5.2 Hierarchical RL

Hierarchical methods rely on decomposing a problem into smaller parts. Decomposing a huge problem of learning into collection of smaller ones and providing useful reinforcement signals for the sub problems [30-34]have experimented with different methods of hierarchical reinforcement learning and hierarchical probabilistic planning. This research has explored many different points in the design spaceof hierarchical methods, but several of these systems were designed for specific situations. The decomposition into sub problems has many advantages.

- Policies learned in subproblems can be shared (reused) for multiple parent tasks.
- The value functions learned in subproblems can be shared, so when the subproblem is reused in a new task, learning of the overall value function for the new task is accelerated.
- If state abstractions can be applied, then the overall value function can be represented compactly as the sum of separate terms that each depends on only a subset of the state variables. This more compact representation of the value function will require less data to learn, and hence, learning will be faster.[37]]

Hierarchical reinforcement learning is an approach to reinforcement learning which splits the global goals of a reinforcement learning agent up into smaller sub goals, and then attempts to tackle each sub goal separately. By doing this the state space is decreased and therefore the efficiency increased. Hierarchical reinforcement learning has been applied to some complex problems with great success. There are several important design decisions that must be made when constructing a hierarchical reinforcement learning system. [37]

5.2.1 Specify subtasks

Hierarchical reinforcement learning involves breaking the target Markov decision problem into a hierarchy of sub problems or subtasks. There are three general approaches to defining these subtasks. One approach is to defineeach subtask in terms of a fixed policy that is provided by the programmer (or that has been learned in some separate process). The option method of [35] takes this approach. The second approach is to define each subtask in terms of a nondeterministic finite-state controller. The Hierarchy of Abstract Machines (HAM) method of [36] takes this approach. This method permits the programmer to provide a “partial policy" that constrains the set of permitted actions at each point, but does not specify a complete policy for each subtask. The third approach is to define each subtask in terms of a termination predicate and a local reward function.

These define what it means for the subtask to be completed and what the final reward should be for completing the subtask. The MAXQ method described in this paper follows this approach. [37]

5.2.2 Employ state abstractions within subtasks

The second design issue is whether to employ state abstractions within subtasks. A subtask employs state abstraction if it ignores some aspects of the state of the environment. For example, in many robot navigation problems, choices about what route to take to reach a goal location are independent of what the robot is currently carrying. With few exceptions, state abstraction has not been explored previously.

5.2.3 Non-hierarchical execution

The third design issue concerns the non-hierarchical execution of a learned hierarchical policy. However, in order to support this non-hierarchical execution, extra learning is required. In hierarchical reinforcement learning, the only states where learning is required at the higher levels of the hierarchy are states where one or more of the subroutines could terminate (plus all possible initial states). But to support non-hierarchical execution, learning is required in all states (and at all levels of the hierarchy). In general, this requires additional exploration as well as additional computation and memory.

5.2.4 Learning algorithm

The final issue is what form of learning algorithm to employ. An important advantage of reinforcement learning algorithms is that they typically operate online. However, finding online algorithms that work for general hierarchical reinforcement learning has been difficult, particularly within the termination predicate family of methods. Some relied on each subtask having a unique terminal state[30]; Some employed a mix of online and batch algorithms to train her hierarchy[32]; and work within the options framework usually assumes that the policies for the sub problems are given and do not need to be learned at all.

Various algorithms have been formed for the application of hierarchical reinforcement learning:

Navigating Continuous Spaces using Hierarchical Reinforcement Learning [38] describes his algorithm for hierarchical reinforcement learning in which two different hierarchical levels are given by the designer. Feudal Reinforcement Learning [39] gives an approach to hierarchical reinforcement learning, in which the hierarchical structure is given by the designer. Reducing the State Space in Hierarchical Reinforcement Learning [40] give a method for dividing up the global state space of a problem into smaller state spaces. The HASSLE Algorithm [41] is described in more mathematical detail than the algorithm on Feudal reinforcement learning. It follows the same general idea as Feudal reinforcement learning, but is more advanced and has methods to automatically discover sub goals and extract high level observations from the given information, without the intervention of a designer. Automatic Discovery of Sub goals in Hierarchical Reinforcement Learning Using Diverse Density [42] propose another algorithm for the automatic discovery of subgoals. Automatic Discovery of Subgoals Using Learned Policies[43] offer another method of automatic subgoal discovery.

Bias named ‘Shaping’ can be used to train hierarchical RL systems from the bottom-up and to all eviate problems of delayed reinforcement by decreasing the delay until the problem is well understood. In shaping technique, teacher presents very simple problems to solve first, then gradually exposes the learner to more complex problems.[9]

5.3 State aggregation and abstraction methods

In state aggregation the state space is discretized into a (relatively small) finite collection of cells. [48]The most straightforward approach for approximation in large-scale MDPs is state-action aggregation (or discretization). Each cell is said to aggregate the states that fall in this cell. Once the aggregation is performed, the new problem is a planning problem in a reduced state space, which can be solved by regular techniques. The main question that arises here is how to perform the aggregation, such that, on one the hand to obtain a good approximation to an optimal policy, and on the other hand, to minimize the problem complexity. While designing a learning algorithm with state/action aggregation, a combination of an efficient exploration technique and an efficient aggregation scheme should be considered, in order to obtain formal guarantees.

Aggregation-Abstraction techniques allow the explicit or implicit grouping of states that are indistinguishable with respect to certain characteristics(e.g., value or optimal action choice). We refer to a set of states grouped in this manner as an aggregate or abstract state, or sometimes as a cluster, and assume that the set of abstract states constitutes a partition of the state space; that is to say, every state is in exactly one abstract state and the union of all abstract states comprises the entire state space.[48]

Abstraction can be given as:

- A uniform abstraction is one in which variables are deemed relevant or irrelevant uniformly across the state space, while a non uniform abstraction allows certain variables to be ignored under certain conditions and not under others.
- An approximate abstraction is non-exact. The exact abstraction groups together states that agree on the value assigned to them by a value function, while the approximate abstraction allows states to be grouped together that differ in value. The extent to which these states differ is often used as a measure of the quality of an approximate abstraction.
- An adaptive abstraction technique is one in which the abstraction can change during the course of computation, while a fixed abstraction scheme groups together states once and for all.
- Goal regression is an abstraction technique that avoids the problem of choosing a particular goal state to pursue.

5.4 Relational reinforcement learning

Relational reinforcement learning is a learning technique that combines reinforcement learning with relational learning or inductive logic programming. The relational representation of states, actions, and policies allows for the representation of objects and relations among them. Due to the use of a more expressive representation language to represent states, actions and Q-functions, relational reinforcement learning can be potentially applied to a new range of learning tasks.[49]

RRL is the use of a relational (first-order) representation to represent states, actions and (learned) policies. Relational learning methods, originating from the field of inductive logic programming are used as generalization engines.

The use of relational representations of states and actions combined with relational regression for Q function generalization allows the use of structural in formation such as the existence of objects with the right properties or relations between objects in the description of the Q-values and as a consequence in the description of the derived policy. This enables the reuse of experience on smaller but related problems when confronted with more elaborate or simply larger tasks. [50]The possible states would not be listed explicitly as input to the RRL algorithm (as they might be for ordinary reinforcement learning). A relational language for specifying states would rather be defined. Actions would also be specified in a relational language and not all actions would be applicable in all states; in fact the number of possible actions may vary considerably across different states. Background knowledge generally valid about the domain can be specified in RRL. This includes predicates that can derive new facts about a given state. Declarative bias for learning relational representations of policies can also be given. Together with the background knowledge, this specifies the language in which policies are represented. The use of relational representations in reinforcement learning offers many advantages out of these is generalization across objects and possibly transfers of learned knowledge to different tasks in similar environments.

6. Conclusions

RL has become one of the intelligent agent's core technologies due to its characteristics of self improving, online learning and very less programming effort. Complex real world domains generally have very large, continuous action spaces, have hidden states and delayed rewards problem which presents a major obstacle in successfully applying RL to these domains. RL still faces many practical issues related to the complexity of the environment in particular when dealing with large states or action sets. Many off shelf RL algorithms have poor scalability to some extent. They become increasingly expensive when there are large states or continuous action spaces. RL techniques that work effectively on a variety of small problems, but there exists very few techniques that can be scaled to larger problems. There is need to establish highly data efficient and robust RL algorithms that are able to solve real world problems. With the help of techniques like Function Approximation, Hierarchical decomposition, Relational RL, State Aggregation Methods and appropriate bias like shaping, local reinforcement signals etc.

Besides development of more robust and efficient algorithms, still much work is to be carried out. There is need solving issues regarding learning techniques regarding methods for approximation, decomposition and incorporation of bias into real life problems. If the right features are represented prominently, then learning is easy; otherwise it is hard. It is time to consider seriously how features and other structures can be constructed automatically by machines rather than by people.

7. References

[1] Rusell, Stuart, and Peter Norvig. "Artificial intelligent: A modern approach." (2003).
[2] Anderson, John Robert. Machine learning: An artificial intelligence approach. Eds. Ryszard S. Michalski, et al. Vol. 2. Morgan Kaufmann, 1986.
[3] Bishop, Christopher M. Pattern recognition and machine learning. Vol. 1. New York: springer, 2006.
[4] Richard Sutton and Andrew Barto (1998).Reinforcement Learning.MIT Press.ISBN 0-585-02445-6.
[5] Powell, W.B., Van Roy, B.: Approximate Dynamic Programming for High-Dimensional Dynamic Resource Allocation Problems. In Si, J., Barto, A.G., Powell,W.B., Wunsch, D., eds.: Handbook of Learning and Approximate Dynamic Programming.Wiley-IEEE Press, Hoboken, NJ (2004) 33-53.
[6] P. Zang, R. Tian, A. L. Thomaz, C. L. Isbell, Batch versus interactive learning by demonstration, in: Proceedings of the International Conference on Development and Learning (ICDL), IEEE, 2010, pp. 219 224.
[7] K. Conn, R. A. Peters, Reinforcement learning with a supervisor for a mobile robot in a real-world environment, in: International Symposium on Computational Intelligence in Robotics and Automation (CIRA), IEEE, Los Alamitos, 2007, pp. 73,78.
[8] K. Ito, Y. Fukumori, A. Takayama, Autonomous control of real snakelike robot using reinforcement learning; abstraction of state-action space using properties of real world, in: M. Palaniswami, S. Marusic, Y. W. Law (Eds.), Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information (ISSNIP), IEEE, Los Alamitos, 2007, pp. 389{394.
[9] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." *arXiv preprint cs/9605103* (1996).
[10] FlorentinWoergoetter and Bernd Porr (2008) Reinforcement learning.Scholarpedia, 3(3):1448.
[11] Abe, Shigeo. "Fuzzy function approximators with ellipsoidal regions." Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 29.5 (1999): 654-661.
[12] Whiteson, Shimon, and Peter Stone. "Evolutionary function approximation for reinforcement learning." The Journal of Machine Learning Research 7 (2006): 877-917.
[13] Lovassy, Rita, et al. "Fuzzy Neural Networks as “Good” Function Approximators."
[14] Barto, A., and R. H. Crites. "Improving elevator performance using reinforcement learning." Advances in neural information processing systems 8 (1996): 1017-1023.
[15] Tsitsiklis, John N., and Benjamin Van Roy. "An analysis of temporal-difference learning with function approximation." Automatic Control, IEEE Transactions on 42.5 (1997): 674-690.
[16] Rashmi Sharma, Manish Prateek and Ashok K Sinha. Article: Use of Reinforcement Learning as a Challenge: A Review. International Journal of Computer Applications 69(22):28-34, May 2013. Published by Foundation of Computer Science, New York, USA.
[17] Noel, Joseph. "Reinforcement Learning with Function Approximation." (2011).
[18] Kimura, Hajime, Kazuteru Miyazaki, and Shigenobu Kobayashi. "Reinforcement learning in POMDPs with function approximation." ICML. Vol. 97. 1997.
[19] Watkins, C.J.C.H. (1989). Learning from Delayed Rewards.PhD thesis, Cambridge University, Cambridge, England.
[20] Tesauro, Gerald (March 1995). "Temporal Difference Learning and TD-Gammon".Communications of the ACM 38 (3).Retrieved 2010-02-08.
[21] Rummery& Niranjan Online Q-Learning using Connectionist Systems"(1994)
[22] Nie, J. and Haykin, S., A dynamic channel assignment policy through Q-learning. IEEE Trans. Neural Netw.10, 1443–1455 (1999).
[23] Beom, H. R. and Cho, H. S., A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning. IEEE Trans. Syst. Man Cybern.25, 464–477 (1995).
[24] Tesauro, Gerald. "Practical issues in temporal difference learning."Reinforcement Learning.Springer US, 1992.
[25] Navarro-Guerrero, Nicolás, et al. "Real-world reinforcement learning for autonomous humanoid robot docking." Robotics and Autonomous Systems 60.11 (2012): 1400-1407.
[26] Proper, Scott, and Prasad Tadepalli. "Scaling model-based average-reward reinforcement learning for product delivery." Machine Learning: ECML 2006. Springer Berlin Heidelberg, 2006.735-742.
[27] Bellman, Richard. A Markovian decision process. No. P-1066.RAND CORP SANTA MONICA CA, 1957.
[28] Bellman, R. E. (1957). Dynamic Programming.Princeton University Press.
[29] Howard, R. A. (1960). Dynamic Programming and Markov Processes.MIT Press, Cambridge,MA.
[30] Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequentialtasks. Machine Learning, 8, 323-339.
[31] Lin, L.-J. (1993). Reinforcement learning for robots using neural networks.Ph.D. thesis,Carnegie Mellon University, Department of Computer Science, Pittsburgh, PA.
[32] Kaelbling, L. P. (1993). Hierarchical reinforcement learning: Preliminary results. In Proceedings of the Tenth International Conference on Machine Learning, pp. 167{173 SanFrancisco, CA. Morgan Kaufmann.
[33] Dayan, P., & Hinton, G. (1993). Feudal reinforcement learning.In Advances in Neural Information Processing Systems, 5, pp. 271{278. Morgan Kaufmann, San Francisco,CA.
[34] Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., &Boutilier, C. (1998). Hierarchicalsolution of Markov decision processes using macro-actions. In Proceedings of theFourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI{98), pp.220{229 San Francisco, CA. Morgan Kaufmann Publishers.
[35] Sutton, R. S., Singh, S., Precup, D., &Ravindran, B. (1999). Improved switching amongtemporally abstract actions. In Advances in Neural Information Processing Systems,Vol. 11, pp. 1066{1072. MIT Press.
[36] Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. In Advances in Neural Information Processing Systems, Vol. 10, pp. 1043{1049 Cambridge,MA. MIT Press.
[37] Dietterich, Thomas G. "Hierarchical reinforcement learning with the MAXQ value function decomposition." arXiv preprint cs/9905014 (1999).
[38] Borga, M (1993). Hierarchical Reinforcement Learning.S. Gielen and B. Kappen,ICANN’93, Springer-Verlag, Amsterdam, p. 513
[39] Dayan, P and Hinton, GE (1993). Feudal Reinforcement Learning.In Advances in NeuralInformation Processing Systems 5, 1993. Morgan Kaufmann, San Mateo, CA
[40] Asadi, M and Huber, M (2004). State Space Reduction for Hierarchical ReinforcementLearning. In Proceedings of the 17th International FLAIRS Conference, pp. 509 - 514,Miami Beach, FL. 2004 AAAI
[41] Bakker, B and Schmidhuber, J (2004). Hierarchical Reinforcement Learning Based onSubgoal Discovery and Subpolicy Specialization. In F. Groen, N. Amato, A. Bonarini, E.Yoshida, and B. Krse (Eds.), Proceedings of the 8-th Conference on IntelligentAutonomous Systems, IAS-8, Amsterdam, The Netherlands, p. 438-445.
[42] McGovern, A and Barto, A (2001). Automatic discovery of subgoals in reinforcementlearning using diverse density. Proc. 18th International Conf. on Machine Learning, 2001

[43] Goel, S and Huber, M (2003). Subgoal Discovery for Hierarchical Reinforcement LearningUsing Learned Policies, In Proceedings of the 16th International FLAIRS Conference, pp.346-350, St. Augustine, FL. 2003 AAAI
[44] R. I. Brafman and M. Tennenholtz. R-MAX - a general polynomial timealgorithm for near-optimal reinforcement learning.Journal of MachineLearning Research, 3:213{231, 2002.
[45] A. L. Strehl and M. L. Littman.A theoretical analysis of model-basedinterval estimation.In Proceedings of the 22nd International Conferenceon Machine Learning (ICML 2005), pages 857{864, 2005.
[46] H. Chapman. Global con¯dence bound algorithms for the exploration-exploitation tradeoff® in reinforcement learning. Master's thesis, TechnionIsrael Institute of Technology, 2007.
[47] M. Kearns and S. P. Singh.Near-optimal reinforcement learning in polynomial time.Machine Learning, 49:209{232, 2002.
[48] Boutilier, Craig, Thomas Dean, and Steve Hanks. "Decision-theoretic planning: Structural assumptions and computational leverage." arXiv preprint arXiv:1105.5460 (2011).
[49] Džeroski, Sašo, Luc De Raedt, and Kurt Driessens. "Relational reinforcement learning." Machine learning 43.1-2 (2001): 7-52.
[50] Tadepalli, Prasad, Robert Givan, and Kurt Driessens. "Relational reinforcement learning: An overview." Proceedings of the ICML-2004 Workshop on Relational Reinforcement Learning. 2004.
[51] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Proceedings of the Seventh International Conference on Machine Learning, Austin, TX, 1990. Morgan Kaufmann.