



Reinforcement Learning: A Review from a Machine Learning Perspective

Samiksha Mahajan

Dept. of Information Technology and Computer Sc.

VVES's Vikas College,

Mumbai, India

Abstract: *Machine Learning is the study of methods for programming computers to learn. Reinforcement Learning is a type of Machine Learning and refers to learning problem which allows machines and software agents to automatically determine the ideal behaviour within a specific context, in order to maximize its performance. RL is inspired by behaviourist psychology based on the mechanism of learning from rewards and 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. This paper provides the overview of Reinforcement Learning from Machine learning perspective. It discusses the fundamental principles and techniques used to solve RL problems. It presents nature of RL problems, with focus on some influential model free RL algorithms, challenges and recent trends in theory and practice of RL. It concludes with the future scope of RL.*

Keywords: *Reinforcement Learning, Machine Learning, TD algorithms, Q learning, SARSA, RL Methods, Actor-critic.*

I. INTRODUCTION

The research field of Machine Learning constitutes a subset of the broader field of Artificial Intelligence. Machine learning is the science of getting computers to act without being explicitly programmed. Machine learning deals with building systems that automatically improve their performance through experience [2]. In Machine Learning the model of learning technology can be of 3 different types:

- Supervised Learning (Learn the model and learning from labeled data)
- Unsupervised Learning (Learn the representation and learning from unlabeled data)
- Reinforcement Learning (Learn to control)

Supervised learning is the machine learning task of inferring a function from labelled training data.[4] The supervised learning process involves learning a classification model using the training data and testing the model using unseen test data to assess the model accuracy. In other words, the agent is provided with a goal for every input and then compares its actual response to the goal and adjusts its internal memory in such a way that it is more likely to produce the appropriate response the next time it receives the same input.

In unsupervised learning the agent receives no feedback from the world at all. Unsupervised learning agent learns by finding hidden structure in unlabelled training data (where output is not known) and is based on the similarities and differences among the input patterns. Its task is to re-represent the inputs in a more efficient way, as clusters or categories or using a reduced set of dimensions. As there are no explicit target output associated with each input, these input representations can then be used by other parts of the system in a way that affects behaviour. Unsupervised learning plays a role in perceptual systems.

Reinforcement Learning is in between these two approaches. Without external supervisor, the agent gets a feedback about the quality of its actions. So RL is based on the interaction of an agent who executes an action and its environment which gives positive or negative feedback i.e. reward. In other words, RL agent learns how to achieve the given goal by trial-and-error interactions with its environment along with maximizing a numerical reward signal. Reinforcement learning is defined not by characterizing learning methods, but by characterizing a learning problem. [1]

Reinforcement learning combines the fields of dynamic programming and supervised learning to yield powerful machine-learning systems. The term reinforcement comes from studies of animal learning in experimental psychology, where it refers to the occurrence of an event, in the proper relation to a response that tends to increase the probability that the response will occur again in the same situation.

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.

II. CORE ELEMENTS OF REINFORCEMENT LEARNING

- *Agent and Environment:* The learner and decision-maker is called the agent. What is outside the agent is considered as environment. With a dynamic environment RL system learns a mapping from situations to actions by trial-and-error interactions.
- *State and Action Spaces:* The agent interacts with an environment with the help of sensor data, changing the environment and obtaining a reward for his action. The states are parameters or features that describe the environment. The sensor data determine states. The state space is simply all the possible states that any specific world can be in. The action space is all the possible choices that the agent can make at any given time. The combination of the state and the action space is called the state-action space, and this is reinforcement learning will use to learn a good strategy for solving the problem. E.g. If agent don't know anything about the environment but agent need some information in order to make sure he can learn from its previous experiences so as to remember the path to the goal once it have found. This information is given to agent via a state space representation.
- *Policy:* The learning agent's way of behaving at a given time i.e. $\pi : S \rightarrow A$. It is a mapping from perceived states of the environment to actions to be taken when in those states. $\pi(s_t) = a_t$ decides upon action in state s_t . The policy may be a simple function, lookup table or it may involve extensive computation such as a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior.
- *Reward Function:* A reward function defines the goal in a reinforcement learning problem i.e. $r : S \rightarrow R$ is often identified with time steps: $r_0, \dots, r_n \in R$. It maps perceived states (or state-action pairs) of the environment to a single number, a reward, indicating the intrinsic desirability of the state. A reinforcement-learning agent's sole objective is to maximize the total reward it receives in the long run. The reward function defines appropriate events for the agent. They are the immediate and defining features of the problem faced by the agent. The reward function must necessarily be fixed and be used as a basis for changing the policy. For example, if an action selected by the policy is followed by low reward then the policy may be changed to select some other action in that situation in the future.
- *Value Function:* The total amount of reward an agent can expect to accumulate over the future given as $V : S \rightarrow R$. the value of a state is the total amount of reward an agent can expect to accumulate over the future starting from that state. Values indicate the long-term desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward. We are most concerned with values when making and evaluating decisions. Action choices are made on the basis of value judgments.
- *Model of environment:* The model might predict the resultant next state and next reward for a given state and action. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. The incorporation of models and planning into reinforcement learning systems is a relatively new development. Early reinforcement learning systems were explicitly trial-and-error learners; what they did was viewed as almost the opposite of planning. Reinforcement learning methods are closely related to dynamic programming methods, which do use models, and that they in turn are closely related to state-space planning methods. Modern reinforcement learning spans the spectrum from low-level, trial-and-error learning to high-level, deliberative planning.

III. STRUCTURE OF RL SYSTEM

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. 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 behaviour and learning algorithm to adapt this policy. 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 behaviour. 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.

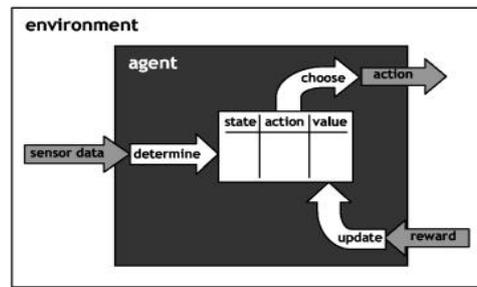


Figure.1: Structure of RL System (source: <http://www.ensta-paristech.fr/>)

Markov decision process (MDPs):

In reinforcement learning environments, states are often assumed to fulfil 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. An RL problem can be modelled as a Markov Decision Process (MDP) defined by a quadruple (S, A, P, R) where S is the set of states, $A(s)$ is the set of actions available in state s , $P : S \times A \times S \rightarrow [0, 1]$ is a transition distribution that specifies the probability of observing a certain state after taking a given action in a given state, $R : S \times A \rightarrow r$ is a reward function that specifies the instantaneous reward when taking a given action in a given state. There is a goal of learning to choose actions that maximize the accumulated sum of rewards over time $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$ where $\gamma \in (0, 1)$ is a discount factor and determines how strongly immediate rewards are weighted compared to rewards in the future. A discount factor $\gamma < 1$ guarantees that the future discounted return R_t is always a finite number if the immediate reward is bounded. If $\gamma < 1$ then rewards far in the future worth exponentially less than the reward received at the first stage then MDP is called a discounted reward MDP. When $\gamma = 1$, the MDP is called undiscounted.

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 observe 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.

IV. RL PROBLEMS AND APPROACHES

There are three main ways how to solve the general reinforcement learning problem. Each of these ways 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 [1].

RL methods are employed to address two related problems: the Prediction Problem and the Control Problem

1. **Prediction Problem:** RL is used to learn the value function for the policy followed. At the end of learning this value function describes for every visited state how much future reward we can expect when performing actions starting at this state.
2. **Control Problem:** By interacting with the environment, a policy is found which maximizes the reward when traveling through state space. At the end an optimal policy is obtained which allows for action planning and optimal. Since this is really a predictive type of control, solving the control problem would seem to require a solution to the prediction problem as well.

The strategies used to tackle with the Prediction and Control problems are Value Iteration, Policy Iteration and Policy Search which are described briefly as follows.

A. Finding Optimal Policy (Value and Policy iteration)

A (state) value function V is any function that maps states to real number i.e. $V: S \rightarrow \text{Real}$. The value of a state $V^\pi(s)$ is the expected return starting from that state. It depends on the agent's policy. The value functions can be estimated from experience. The value of taking an action in a state $Q^\pi(s,a)$ under policy π is the expected return starting from that state, taking that action. If an agent follows policy and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state's value, $V^\pi(s)$, as the number of times that state is encountered approaches infinity. If separate averages are kept for each action taken in a state, then these averages will similarly converge to the action values, $Q^\pi(s,a)$.

The agent that interacts with the MDP is modelled in terms of a policy. A deterministic policy $\pi: S \rightarrow A$ is a mapping from the set of states to the set of actions. Applying π means always selecting action $\pi(s)$ in state s . This is a special case of a stochastic policy, which specifies a probability distribution over A for each state s , where $\pi(s, a)$ denotes the probability to choose action a in state s .

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. π^* is the optimal policy function for all states S . All optimal policies of an MDP share the same value function V^* and include at least one deterministic policy. The objective is to find a policy π^* such that $V^{\pi^*}(s) \geq V^{\pi}(s)$ for any policy and any state s . Two standard methods for finding optimal policy are:

1. *Value Iteration:*

Zero-initialize and iteratively refine $V(s)$ as it will converge towards $V^*(s)$.

- Choose any initial state value function V_0

- Repeat for all $n \geq 0$

For all s ,

$$V_{n+1}(s) \leftarrow \max_a \{R(s, a) + \gamma V_n(T(s, a))\}$$

Until convergence

This converges to V^* and any greedy policy with respect to it will be an optimal policy.

2. *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^* .

- Choose any initial policy π_0

- Repeat for all $n \geq 0$

Compute V^{π_n} (policy evaluation step)

Choose π_{n+1} greedy with respect to V^{π_n} (policy improvement step)

- Until $V^{\pi_{n+1}} = V^{\pi_n}$

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.

B. Policy search

Policy search is the solution to RL problem focusing on behaviour of agent. It provides a powerful tool for solving many problems in reinforcement and control. These methods represent and work with policies directly without the intermediate step of representing a value function i.e. it provides family of methods which prefer direct search in policy space. Policy search typically requires explicitly searching in π for good policy.

The form of policy search can be summarize as

$$\pi(s) = \max_a Q(s, a)$$

Here policy needs to continuously adjust to improve the performance and then stop. The problem can be formalized to be solved by using standard local function optimization. This may be computationally expensive and more prone to local optima than certain dynamic programming based methods

V. RL METHODS FOR ITS ALGORITHMS

Reinforcement learning algorithms can be broadly classified into critic-only, actor-only, and actor-critic methods. [4]

A. Critic-only methods:

These are based on the idea to first find the optimal value function and then to derive an optimal policy from this value function. Critic Only methods rely exclusively on value function approximation and aim at learning the approximate solution to the Bellman equation, and then trying to find a near-optimal policy. Such methods are indirect in the sense that they do not try to optimize directly over a policy space. A method of this type may succeed in constructing a “good” approximation of the value function yet lack reliable guarantees in terms of near-optimality of the resulting policy.

E.g. Dynamic Programming, Temporal Difference Learning, Use of eligibility traces for speed up learning with algorithms such as $Q(\lambda)$, SARSA (λ), and TD (λ) [1], Use of function approximators for large state problems, Use of model-based RL techniques without knowing P and R a priori if the agent possesses an internal model of the environment building i.e. the world model for agent to create simulated experience and then learn from this second source of experience

B. Actor Only methods:

Actor-only methods search directly in policy space. These methods work with a parameterized family of policies by a real-valued parameter vector θ which allows integration of prior knowledge about the task and thus reduce the search complexity. The gradient of the performance, with respect to the actor parameters, is directly estimated by simulation, and the parameters are updated in a direction of improvement. A possible drawback of such methods is that the gradient estimators may have a large variance. When the policy further changes, a new gradient is estimated independently of past estimates. So there is no “learning” in the sense of accumulation and consolidation of older information. E.g. William’s REINFORCE algorithm [5] for immediate rewards, which is a special case of a policy gradient method, Evolutionary Algorithms which are randomize direct search algorithms. [6-8]

C. Actor-critic methods:

Actor-Critic methods [9] combine the strong points of actor-only and critic-only methods. The critic uses an approximation architecture and simulation to learn a value function, which is then used to update the actor's policy parameters in a direction of performance improvement. This architecture offers the possibility to combine a value function approach (as the critic) with a policy gradient approach (as the actor).

The actor-critic concept was originally introduced by Witten [10] and then by Barto, Sutton and Anderson [11], who coined the terms actor and critic. It has been particularly successful in neuronal RL. A detailed study of actor-critic algorithms is given in [12] E.g. Policy gradient strategies (which assume a differentiable structure on a predefined class of stochastic policies and ascent the gradient of a performance measure)

Different policy gradient methods [13-15] vary in the way the performance gradient is estimated and the value function is approximated. An unified view of these methods based on perturbation analysis [16][17] builds an algorithm directly for the metric, while [18] integrate the natural gradient in an efficient actor-critic architecture. [19]Policy search approaches to reinforcement learning for quadruped locomotion.

VI. MODEL BASED AND MODEL FREE METHODS

Depending on whether the RL algorithm needs or learns explicitly transition probabilities and expected rewards for state action pairs, Critic-only, actor-only and actor-critic methods are further divided into model-based and model-free algorithms. [3][4]

A. 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.

B. 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[23], Real Time Dynamic Programming, Queue Dyna, Plexus Planning.

VII. MODEL FREE ALGORITHMS

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. The TD method is called a "bootstrapping" method, because the value is updated partly using an existing estimate and not a final reward.

TD(λ) [1] 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]

Q-learning [21] 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. Q-Learning does not pay attention to what policy is being followed. Instead, it just uses the best Q-Value. Thus, it is an **off-Policy** learning algorithm. It is called an off-policy because the policy being learned can be different than the policy being executed.

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. SARSA is an **on-policy** learning Algorithm. It updates value functions strictly on the basis of the experience gained from executing some (possibly non-stationary) policy.

VIII. CHALLENGES IN RL

Implementing RL algorithms in complex real world domains 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.

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. [22] 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.

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.

The reward the agent receives is not necessarily correlated to the last action the agent has performed, but can be delayed in time. The problem of mapping the reward only to those choices that led to it is called the temporal credit assignment problem [1]

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. Generalization can be achieved using function approximators, where gluing together neighboring states can be viewed as the simplest example.

In order to learn structure and features of an unknown environment, the agent needs to explore the state-action space. In the RL context, exploring means choosing actions, which are regarded as suboptimal given the current state of knowledge. While exploration is necessary in the initial learning phase, it may become less relevant with increasing knowledge the agent has gathered by trial-and-error and that can be exploited for decision making. Finding the balance between obtaining as much reward as possible and sufficiently exploring the solution space is known as the Exploration versus Exploitation problem.

IX. RECENT TRENDS IN THEORY AND PRACTICE OF RL

Strength of RL is that it does not require explicit examples of executing correct action. [24] It can be applied to system for which such examples are not readily available. E.g. Dynamic channel assignment in communications networks [25], to construct fuzzy logic rule bases for fuzzy control systems [26]. A well-known example for the successful application of RL technique is the back-gammon computer developed by Tesauro. [27]

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. Recently new class of algorithms, which include the E^3 [28], R-MAX [29], MBIE [30] and GCB [31] algorithms, were presented and proved to reach a nearly optimal policy (with high probability) within a time that is polynomial in the problem size.

A. Multiagent RL:

Some key ideas in multi-agent RL are drawn from game theory. An important generalization of MDPs is stochastic games (or multi-agent MDPs) in which a team of agents interacts. Every agent may have its individual reward function, and optimality of a policy can only be interpreted as an optimal response to the behaviour of the other agents. Problems which occur in multi-agent environments are how to define the expected accumulated reward of an agent and how to coordinate all agents.

B. Hierarchical RL:

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. [32]

Policies learned in sub problems can be shared (reused) for multiple parent tasks. The value functions learned in sub problems can be shared, so when the sub problem 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.

C. Relational RL:

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.[33] RRL is the use of a relational (first-order) representation to represent states, actions and (learned) policies. The use of relational representations in reinforcement learning offers many advantages such as generalization across objects and possibly transfers of learned knowledge to different tasks in similar environments.

D. Bayesian RL and Kernel Based Methods:

A Bayesian approach to RL introduces a prior on a value function or a model. Experience is then used to compute a posterior over the value function or model, which captures all the knowledge the agent can have about the environment.

On this basis, decisions that optimize future expected return yield theoretically an optimal trade-off between exploration and exploitation. Several kernel based methods have recently been developed in the context of RL [34]

These approaches use kernel methods as approximators for value functions or the unknown MDP and combine them with classical approaches such as dynamic programming.

X. CONCLUSIONS

Reinforcement learning as a type of machine learning extends the application and research to a broad area of control and decision problems which are not generally tackled by supervised or unsupervised learning techniques. 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.

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 with the help of recent practices in reinforcement learning and new approaches. 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.

REFERENCES

- [1] Richard Sutton and Andrew Barto (1998). Reinforcement Learning. MIT Press. ISBN 0-585-02445-6.
- [2] Anderson, John Robert. Machine learning: An artificial intelligence approach. Eds. Ryszard S. Michalski, et al. Vol. 2. Morgan Kaufmann, 1986.
- [3] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." arXiv preprint cs/9605103 (1996).
- [4] Heidrich-Meisner, Verena, Martin Lauer, Christian Igel, and Martin A. Riedmiller. "Reinforcement learning in a nutshell." In ESANN, pp. 277-288. 2007.
- [5] [R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3):229–256, 1992. 15]
- [6] D.E. Moriarty, A.C. Schultz, and J.J. Grefenstette. Evolutionary Algorithms for Reinforcement Learning. Journal of Artificial Intelligence Research, 11:199–229, 1999.
- [7] K. Chellapilla and D.B. Fogel. Evolution, neural networks, games, and intelligence. Pro-ceedings of the IEEE, 87(9):1471–1496, 1999.
- [8] S.M. Lucas and G. Kendall. Evolutionary computation and games. IEEE Computational Intelligence Magazine, 1(1):10–18, 2006.
- [9] [A. Barto, R. Sutton, and C. Anderson, Neuron-like elements that can solve difficult learning control problems, IEEE Transactions on Systems, Man and Cybernetics, 13 (1983), pp. 835–846.]
- [10] I.H. Witten. An adaptive optimal controller for discrete-time markov environments. Information and Control, 34(4):286–295, 1977.
- [11] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man and Cybernetics, 13:834–846, 1983.]
- [12] V.R. Konda and J.N. Tsitsiklis. On Actor-Critic Algorithms. SIAM Journal on Control and Optimization, 42(4):1143–1166, 2006.
- [13] V.R. Konda and J.N. Tsitsiklis. On Actor-Critic Algorithms. SIAM Journal on Control and Optimization, 42(4):1143–1166, 2006.
- [14] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In Advances in Neural Information Processing Systems, volume 12, pages 1057–1063, 2000.
- [15] J. Baxter and P.L. Bartlett. Infinite-horizon policy-gradient estimation. Journal of Artificial Intelligence Research, 15(4):319–350, 2001.]
- [16] X.R. Cao. A basic formula for online policy gradient algorithms. IEEE Transactions on Automatic Control, 50(5):696–699, 2005.
- [17] J. Bagnell and J. Schneider. Covariant policy search. In Proc. 18th Int'l Joint Conf. on Artificial Intelligence, pages 1019–1024, 2003.
- [18] Peters et al. [26 J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In Proc. 3rd IEEE-RAS Int'l Conf. on Humanoid Robots, pages 29–30, 2003.]
- [19] Jimenez, Antonio R. Diss. Massachusetts Institute of Technology, 2006. Deisenroth, Marc Peter, Gerhard Neumann, and Jan Peters. "A Survey on Policy Search for Robotics." Foundations and Trends in Robotics 2.1-2 (2013): 1-142.
- [20] Tesauro, Gerald (March 1995). "Temporal Difference Learning and TD-Gammon". Communications of the ACM 38 (3). Retrieved 2010-02-08.
- [21] Watkins, C.J.C.H. (1989). Learning from Delayed Rewards. PhD thesis, Cambridge University, Cambridge, England.
- [22] Bellman, Richard. A Markovian decision process. No. P-1066. RAND CORP SANTA MONICA CA, 1957.

- [23] 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.
- [24] 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.
- [25] Nie, J. and Haykin, S., A dynamic channel assignment policy through Q-learning. IEEE Trans. Neural Netw.10, 1443–1455 (1999).
- [26] 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).
- [27] Tesauro, Gerald. "Practical issues in temporal difference learning." Reinforcement Learning. Springer US, 1992.
- [28] M. Kearns and S. P. Singh. Near-optimal reinforcement learning in polynomial time. Machine Learning, 49:209{232, 2002.
- [29] R. I. Brafman and M. Tennenholtz. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research, 3:213{231, 2002.
- [30] A. L. Strehl and M. L. Littman. A theoretical analysis of model-based interval estimation. In Proceedings of the 22nd International Conference on Machine Learning (ICML 2005), pages 857{864, 2005.
- [31] H. Chapman. Global confidence bound algorithms for the exploration-exploitation tradeoff in reinforcement learning. Master's thesis, Technion Israel Institute of Technology, 2007.
- [32] Dietterich, Thomas G. "Hierarchical reinforcement learning with the MAXQ value function decomposition." arXiv preprint cs/9905014 (1999).
- [33] Džeroski, Sašo, Luc De Raedt, and Kurt Driessens. "Relational reinforcement learning." Machine learning 43.1-2 (2001): 7-52.
- [34] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In Proc. 22nd Int'l Conf. on Machine learning, pages 201–208. ACM Press New York, NY, USA, 2005.).