Journal of Autonomous Intelligence (2019) Volume 2 Issue 2.
doi:10.32629/jai.v2i2.45

Reinforcement Learning: A Technical Introduction – Part I

Elmar Diederichs*

Elements of Euclid Berlin, Germany; Email: elmar.diederichs@eoe.science

ABSTRACT

Reinforcement learning provides a cognitive science perspective to behavior and sequential decision making provided that reinforcement learning algorithms introduce a computational concept of agency to the learning problem. Hence it addresses an abstract class of problems that can be characterized as follows: An algorithm confronted with information from an unknown environment is supposed to find step wise an optimal way to behave based only on some sparse, delayed or noisy feedback from some environment, that changes according to the algorithm's behavior. Hence reinforcement learning offers an abstraction to the problem of goal-directed learning from interaction. The paper offers an opinionated introduction in the algorithmic advantages and drawbacks of several algorithmic approaches to provide algorithmic design options.

Keywords: *Classical Reinforcement Learning; Markov Decision Processes; Prediction and Adaptive Control in Unknown Environments; Algorithmic Design*

ARTICLE INFO
Received: July 10, 2019
Accepted: Aug 1, 2019
Available online: Aug 19, 2019

***CORRESPONDING AUTHOR**
Dr. Elmar Diederichs, Elements of Euclid Berlin, Germany;
elmar.diederichs@eoe.science;

CITATION
Elmar Diederichs. Reinforcement Learning A Technical Introduction. Journal of Autonomous Intelligence 2019; 2(2): 25-41. doi: 10.32629/jai.v2i2.45

1. Introduction

Most readers are familiar with two different types of machine learning: On the one hand we have supervised learning. In supervised learning the algorithm learns from a training set of labeled examples provided by a knowledgeable external supervisor. In a sense the known label y gives the right and unique answer for every input x of the machine learning algorithm. However, for many sequential decision making and other adaptive control problems, it is difficult to provide an explicit supervision to the algorithm[38]. On the other hand we have unsupervised learning, where the general task is to find hidden structures in unlabeled data. Surprisingly reinforcement learning (RL) is different from supervised and unsupervised learning, since it tries to maximize some utility function concurrent with learning a reward signal coming from an environment that is under the influence of the algorithm. Consequently every reinforcement learning algorithm has to exploit what it already knows about the environment in order to obtain more rewards, but it also has to explore a still incomplete known, i.e. stochastic or dynamic environment and its causal relations in order to choose better actions for future situations. Exploration can be done by Monte-Carlo-Methods, genetic algorithms or genetic programming. Exploitation uses statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the environment[54].

The obvious dilemma is that neither exploitation nor exploration can be done by any algorithm without systematically failing at that task. So the only algorithmic solution here is to try a variety of different actions and progressively favor those that appear to be locally the best. Since this task of balancing simultaneously local exploration and exploitation does not arise in supervised and unsupervised learning, reinforcement learning is a generic type of machine learning[22]. The basic mathematical framework for reinforcement learning is the stochastic Markov decision process (MDP)[17]. A variety of reinforcement methods come up if we consider different types of underlying MDPs, auxiliary assumption, different reward

functions as only implicit given learning models adapted to certain kinds of environments and different algorithms for exploration and identifying the strategy. Reinforcement learning can generate global near-optimal solutions to large and complex MDPs.

In sum, reinforcement learning is a general-purpose framework for weak, human-like artificial intelligence and dates back to the early days of cybernetics[68,33] and since then has been successful applied to a large number of different practical problems like e.g. inventory control[37], queuing systems[41], maintenance management[47], transportation problems[69], ecology[34], stochastic shortest path problems[56], Atari video games[29], the board game Go[18], in controlling dynamic robotic systems for manipulation[26], locomotion[46], autonomous driving[45] and many others. Reinforcement learning algorithms have also been extensively used as tools for constructing autonomous systems that improve themselves making new experiences.

During the decades reinforcement learning has been become a general framework of various techniques with different origins:

i) The general model of model-based[43] or model-free RL is given by Markov decision processes and the recursive Bellman equations for the state and action value functions. Markov decision processes have been extend to partial observable Markov decision processes in case of incomplete information from the responding environment.

ii) Algorithms for RL can be classified into critic-only, actor-only, and actorcritic methods. In every class there are model-based and model-free algorithms, depending on whether the algorithm needs or learns explicitly transition probabilities and expected rewards for state-action pairs.

iii) Iterative learning methods (value and policy iteration) based on models for the environment have been developed and extended to large state spaces of the Markov decision processes using approximations of the value functions.

iv) Model-free iterative methods (several variants of Temporal Difference Learning) for unknown environments have been developed and extended by new learning methodologies like the actor-critic-approach.

Contribution of this paper: Recently many articles that aim to provide some guidance while introducing to reinforcement learning have been published. They are either focused on deep reinforcement learning e.g.[30] on applications of reinforcement learning e.g.[39] or on special topics e.g.[24,51,36]. However from an algorithmic point of view finding the right design for a given problem requires the description of sufficient technical details as well as knowledge about their consequences for the numerical simulations. Very few articles of this sort are available. Hence the goal of this article is to provide the mathematical understanding of reinforcement learning that is needed from an engineering point of view. The first part of the article gives a sketch of existing classical approaches. However a comprehensive survey of RL-algorithms is far beyond the scope of this paper. The second part to be published in another article will focus on deep reinforcement learning and the analysis of cases where deep reinforcement learning is outperformed by other approaches.

2. Markov Decision Processes

This section briefly introduces into the mathematical framework behind reinforcement learning. To this end we denote by $\mathbb{E}[\cdot]$ the expectation. Further let $\mathbb{P}$ denote a probability measure and R_t a reward at time t, that can be observed or estimated. In this article all vectors are column vectors.

2.1 Basic Definitions

Intuitively spoken the general MDP-framework $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ has the following components[4]:

i) a set of time depend states S_t of a dynamic and sometimes stochastic system with values $s \in S$ that reprsents the information used to determine what happens next in the environment.

ii) a finite set $\boldsymbol{\mathcal{A}}$ of deterministic actions A_t with values a that as a consequence causes a transition between $s \to s'$. In the following let s' always be a successor state of s.

iii) a not necessary stationary state transition matrix $\mathcal{P} \in \mathbf{R}^{|S|\times|S|}$ where
$p^a_{s,s'} := \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
expresses the frequency of transitions between

different states s and s'. Hence our framework also models non-deterministic environments, where taking the same action from the same state could lead to two different outcomes. Additionally it is assumed that in case of $s \to s'$ the well known Markov property applies to $p^a_{s,s'}$.

iv) a scalar transition reward function $\mathcal{R}^a_s := \mathbb{E}[r_{t+1}(s')|S_t = s, A_t = a]$ that defines the goal of the algorithm

v) a discount factor $\gamma \in [0, 1]$ on the reward that represents the present value of future rewards, avoids infinite returns in cyclic processes and represents further fine-grained future uncertainties. Only for stationary RL-problems, all rewards should have the same influence.

vi) a state value function $v : \mathcal{S} \times \mathcal{A} \to \mathbf{R}$ with $v(s) := \mathbb{E}[R_t|S_t = s]$ to be optimized over a series of system states, that can be considered as performance metric wrt. a chosen policy, that that defines behavior of the agent.

Here $R_t : \boldsymbol{\mathcal{S}} \times \boldsymbol{\mathcal{A}} \times \boldsymbol{\mathcal{S}} \to \mathbf{R}$ is the immediate return such that

$$-\infty < R_t := \sum_t^T \gamma^t r_{t+1} < +\infty \tag{1}$$

where the algorithm receives a present or future, scalar (positive) reward or (negative) punishment $r_{t+1} \in \mathbf{R}$ for choosing action a_t in state s_t. Hence in RL we always assume that a goal can be described by the maximization of expected reward. In a narrow sense $\mathcal{P}$ can be considered as the model of the MDP and a full sequence of observed quantities

$$(s_t, a_t, r_t), (s_{t+1}, a_{t+1}, r_{t+1}), (s_{t+2}, a_{t+2}, r_{t+2}), \ldots$$

is called a trajectory $\mathcal{T}$.

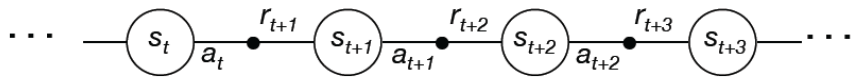


Figure 1. Scheme of a sequential decision process: the transfer $s_i \mapsto s_{i+1}$ by a_i due to γ_{i+1}, γr_{i+2}, $\gamma^2 r_{i+3}, \ldots$ defines the policy π.

In a more general setting where the information about the states s_i is incomplete, one has to substitute s_i by the partial observation ω_i about s_i.

For simplicity let us at first assume that $\mathcal{P}$ and $\mathcal{R}^a_s$ are known and the number of is finite system states $|S|$.Then the environment can be modeled by an abstract stochastic process called first-order Markov chain: If we observe the changing environment, we observe the Markov chain, which is determined by the trajectory. Recall that the essential Markov property requires that the probabilities of arriving in a state s_{t+1} and receiving a reward r_{t+1} in the third assumption of the general framework only depend on the state s_t and the action a_t.

The agent that interacts with the MDP is modeled in terms of a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ or a stochastic policy $\pi(a|s) : \mathcal{S} \times \mathcal{A} \to [0, 1]$, where

$$\pi(a|s) := \mathbb{P}[A_t = a|S_t = s]$$

denotes the conditional probability to choose action a in state s. The general task for a reinforcement algorithm that aims to learn or to plan is to realize a series of actions according to a policy that leads to the emergence of a Markov chain of external states with maximum expected return i.e.

$$\pi^* = \arg\max_\pi \mathbb{E}\Big[\sum_{t=0}^{|S|} r^a_t|\pi\Big]$$

Remarks: In general, there are different modes of reinforcement learning. First of all one can differentiate between model based and model free RL: the first is concerned with problems from planning, where the algorithm has full knowledge about the environment and intends to improve the policy without interaction with the environment. The second one is concerned with problems of evaluating and optimizing the future: In prediction problems the RL-algorithm tries to learn the optimal state value function from the responses to its own actions for the uniform random policy in an unknown environment. In adaptive control problems the RL-algorithm tries to learn the optimal value function over all possible policies in order to identify the optimal policy[75].

Recall that on-policy methods have to be separated from off-policy methods in RL. In the first case one attempts to evaluate or improve the policy that currently guides the decisions of the agent influencing the responding environment. In the second case the agent tries

to learn from series of states that are obtained from other policies that are only intended to generate trajectories in a state space. Non-policy based methods usually introduce a bias when used with a replay buffer since the trajectories are usually not obtained solely under the current policy. This makes off-policy methods sample efficient as they are able to make use of any experience.

2.2 The Bellman Equations

A Bellman equation expresses some recursive relationship e.g. between the value of a state and the value of its successor states and it can be used to efficiently solve RL-problems. It averages over all possibilities weighted by their probability of occurrence. Starting from $v(s) := \mathbb{E}[R_t|S_t = s]$ we can decompose the state value function in the immediate reward r_t and the discounted value of successor states $\gamma v(S_{t+1})$:

$$\begin{aligned} v(s) &= \mathbb{E}[r_t + \gamma R_{t+1}|S_t = s] \\ &= \mathbb{E}[r_t + \gamma v(S_{t+1})|S_t = s] \\ &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{s,s'}^a v(s') \end{aligned} \tag{2}$$

Using matrix notation the linear Bellman-type equation (3) follows, that obviously relates the state value function to itself via the problem dynamics:

$$\begin{aligned} v &= \mathcal{R}_s^a + \gamma \mathcal{P} v \qquad \Leftrightarrow \\ v &= (\mathbf{1} - \gamma \mathcal{P})^{-1} \mathcal{R}_s^a \end{aligned} \tag{3}$$

where $v \in \mathbf{R}^{|S|}$ and $\mathbf{1}$ is the unit matrix in $\mathbf{R}^{|S| \times |S|}$. For a finite state space, (3) yields a finite set of $|\boldsymbol{S}|$ linear equations, which obviously can be solved using standard numerical methods having computational complexity of $\mathcal{O}(|S|^3)$.

A cheaper alternative for large MDPs up to certain accuracy is the use of dynamic programming[6]. To that end we consider the right hand side of (3) as contractive operator $T(v)$ mapping one state value function v to another state value function. From that point of view v is the unique fixed point of a contraction mapping T in a metric and complete space[64]. Due to Banach's fixed point theorem we can find the optimal v at a linear convergence rate of γ by iteratively applying T to some initial state value function. Recall that in dynamic programming one assumes full knowledge of the transition, environment and reward models. A drawback of DP is that they involve operations over the entire state set of the MDP. If the state space is very large, this becomes computationally prohibitive.

Now we let things become a little bet more complex. In order to model the behavior of the algorithm, we introduce the policy $\pi : S \rightarrow \mathcal{A}$ and define

$$\begin{aligned} \mathcal{P}^\pi &:= \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P} \\ \mathcal{R}_s^\pi &:= \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}_s^a \\ v_\pi(s) &:= \mathbb{E}_\pi[R_t|S_t = s] \end{aligned} \tag{4}$$

then we can derive some Bellman equations analogously:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \qquad \Leftrightarrow \\ v_\pi &= (\mathbf{1} - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi \end{aligned} \tag{5}$$

In (5) it is not assumed that the policy is probabilistic, only the transitions $s \rightarrow s'$ have a distribution.

Let us consider the case of a stochastic policy $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$. Additionally we introduce the action-value or quality function $q_\pi : S \times \mathcal{A} \rightarrow \mathbf{R}$ with $q_\pi(s, a)$ as the expected return starting from state s, taking action a in following policy π:

$$q_\pi(s, a) := \mathbb{E}_\pi[R_t|S_t = s, A_t = a] \tag{6}$$

From that we conclude from (4) with (6):

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a)) \tag{7}$$

Now if we decompose (6) as in (2), we find:

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[r_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \\ &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a v_\pi(s') \end{aligned} \tag{8}$$

Now (8) and (7) give us:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a v_\pi(s') \right) \tag{9}$$

This is the *Bellman expectation equation* for the state-value function that allows us to rewrite the action-value function as:

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} p_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a') \tag{10}$$

This is the *Bellman expectation equation* for the action-value function. The optimal action value function specifies the best possible performance in the MDP and can be considered as the solution of the RL-problem. So finally we are interested in the following optimal value functions:

$$\begin{aligned} v^*(s) &= \max_\pi v_\pi(s) \\ q^*(s, a) &= \max_\pi q_\pi(s, a) \end{aligned}$$

It is well known that for any finite MDP, that there is an optimal, deterministic policy π^* s.t. $\forall s \in S$, $\forall \pi$ $v_\pi(s) \leq v_{\pi^*}(s)$. In that case we also have $v_{\pi^*}(s) = v^*(s)$ and $q_{\pi^*}(s, a) = q^*(s, a)$[58]. Here $q^*_\pi(s, a)$ can be considered as optimal control policy. An optimal policy π^* can be found by

$$\pi^* = \arg\max_{a \in \mathcal{A}} q^*(s, a) \tag{11}$$

Both optimal value functions are recursively related by the *Bellman optimality equation*[5]

$$v^*(s) = \max_{a \in \mathcal{A}} q^*(s, a).$$

Obviously one can use numerical expensive Monte Carlo methods by performing several simulations from the state space while following π to get some empirical estimates for $q_\pi(s, a)$ and $v_\pi(s)$. In practice for limited data the performance of this approach is low.

Extensions: In the past different types of MDPs have been characterized. MDPs have be classified as stationary if $\mathcal{P}$, $\mathcal{A}$ and the immediate rewards are not Time dependent. Otherwise the MDP is called non-stationary[50]. The number of states and actions can be finite, countable and simply Borel measurable[14]. The planning horizon |S| can be either finite or infinite[57]. The system states can be partially, only indirect or fully observable and the decision epochs can be discrete or continuous[70]. A partially observable Markov decision process (POMDP) is an MDP with hidden states. Roughly spoken it is a Hidden Markov model with actions[36]. Moreover semi-MDPs have been introduced, where an additional parameter of interest is the time spent in each transition. In practice optimal control primarily deals with continuous MDPs.

In the following this paper only considers stationary, discrete and complete observable MDPs with finite horizon. This is the most frequent case from an engineering point of view. The next chapters discuss different algorithmic, modelbased and model-free approaches to exploit information that can represented as MDP.

3. Critic-Only Methods for finite MDPs

Value Iteration and Policy Iteration are both model based critic-only algorithms, which are based on the idea to first find the optimal value function using $\mathbb{P}$ and then to derive an optimal policy from this value function. Examples are numerical methods to solve the system of S linear equations generated by the Bellman-equations directly. Fixpoint methods, that will be described below, are critic-only methods also.

3.1 Model-based Algorithms

Due to the nonlinearity of the Bellman optimality equations, there are in general no closed form solutions for MDP-problems, but many iterative schemes exist, that have to cope with different numerical and stochastic problems.

3.1.1 Value Iteration

The advantages of policy-based schemes are well known: They show better convergence properties and are effective even in higher dimensions. The bad news is that the evaluation of a policy is inefficient and comes with a high variance[64,50,23,70]. In the literature RL-problems with known MDP are based on a model of the environment. Hence criteria for the best policy π are known and such problems are called planning problems. Since in practice there are only finitely many policies in a finite-state, finite-action MDP, we expect termination of such algorithms in a finite number of iterations.

Let us introduce the value iteration[53] that also works with loopy, stochastic MDPs and compare it to the so called policy iteration.

Algorithm 1: Scheme of Value Iteration for RL (synchronous backups)

input: $\mathcal{P}, \mathcal{R}, \mathcal{S}$
k=0
Initialize $\forall s: v^{(k)}(s) := 0$.
while *no uniform convergence in* $v(s)$ **do**
  k=k+1
  for t=1 **to** $|S|$ **do**
    Compute the Bellman expectation backup:
    $v^{(k)}(s_t) \leftarrow \max_{a \in \mathcal{A}} \left\{ \mathcal{R}^a_{s_t} + \gamma \sum_{s' \in \mathcal{S}} p^a_{s_t,s'} v^{(k)}(s') \right\}$
  end
end

The optimal value function v^* can also be found by solving a linear programming problem[64].

Recall that asynchronous algorithms back up the values of states in any order whatsoever, using whatever values of other states happen to be available. In particu-

lar the values of some states may be backed up several times before the values of others are backed up once. In the case of a synchronous update, the algorithm can be viewed as implementing a "Bellman backup operator" that takes a current estimate of the value function, and maps it to a new estimate. Alternatively, we can also perform asynchronous updates where we would loop over the states (in some order), updating the values one at a time.

Note that here there is no explicit π^* and the optimality of the state value function is always restricted to the so called horizon $|S| < \infty$. However the corresponding optimal policy can be generated from

$$\pi^*(s) = \arg\max_{a\in\mathcal{A}} \left\{ \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} p_{s,s'}^a v^*(s') \right\}$$

The value iteration needs more iterations than the policy iteration but only a complexity of $\mathcal{O}(|\mathcal{A}||\mathcal{S}|^2)$ per iteration. It converges exponentially fast to the optimal value function, but only asymptotically for the discounted infinite horizon.

3.1.2 Function Approximation

Recall that for large state spaces the value functions can not have the form of a lookup table due to memory limitations. Then function approximation is the method of choice[25]. Its error reads as:

$$MSVE(\theta) := \sum_s d(s)(v_\pi - \hat{v}(s,\theta))^2 \qquad (12)$$

where $\theta \in \mathbf{R}^K$ with $K < |S|$ is a parameter or weight vector, the estimator $\hat{v}(s, \theta)$ is assumed to be differentiable wrt. θ and the occupation frequency $d(s)$ denotes the fraction of time the agent spent in s according to π:

$$\begin{aligned} d(s) &:= \frac{e(s)}{\sum_{s'\neq s} e(s')} \\ e(s) &:= h(s) + \sum_{\hat{s}} e(\hat{s}) \sum_{\hat{a}} \pi(\hat{a}|\hat{s}) \mathbb{P}(s|\hat{s},\hat{a}) \end{aligned} \qquad (13)$$

In (13) $h(s)$ denotes the probability that an episode begins in states s and $e(s)$ the average time steps spent in s according to π in a single episode. One option to minimize (12) is the local converging gradient method. Then we get:

$$\theta_{t+1} = \theta_t + \alpha(u(s_t) - \hat{v}(s_t,\theta_t))\nabla\hat{v}(s_t,\theta_t)$$

with $\mathbb{E}[u(s_t)] = v_\pi(s_t)$ since $v_\pi(s_t)$ is not available during computation. Here α is the step size parameter. If $u(s_t)$ is an unbiased estimator for every t than θ_t is guaranteed to converge to a local optimum for a decreasing series of values of α. A special and often used representation of $u(s_t)$ is given by linear projections:

$$v(s) \approx u_\theta(s) = \theta^T \phi(s)$$

where $\phi(s)$ is a feature vector. For most tasks the features are hand picked.

3.1.3 Policy Iteration

The pseudocode of policy iteration[7] reads as follows:

Algorithm 2: Scheme of Policy Iteration for RL (synchronous backups)

input: $\mathcal{P}, \mathcal{R}, \mathcal{S}$
Initialize $\pi^{(k)}(s)$ randomly for all states s. Set $k = 0$.
while *no uniform convergence in* $\pi(s)$ **do**
  k=k+1
  1.evaluation of π(k)(s): iterative application of Bellman expectation backup:
  for *t=1* **to** $|S|$ **do**
    $v_{\pi^{(k)}}(s_t) \leftarrow \sum_{a\in\mathcal{A}} \pi^{(k)}(a|s_t)\left(\mathcal{R}_{s_t}^a + \gamma \sum_{s'} p_{s_t s'}^a v_{\pi^{(k)}}(s')\right)$
  end
  and solve directly for vπ(k) with complexity O(|S|3).
  2. Improve the current policy by acting greedily with respect to $v_\pi(k)$:
  for *t=1* **to** $|S|$ **do**
    evaluate $q_{\pi^{(k)}}(s_t, a)$ using (8)
    solve $\pi^{(k)}(s_t) = \arg\max_{a\in\mathcal{A}} q_{\pi^{(k)}}(s_t, a)$.
  end
end

Policy iteration needs fewer iterations than the value iteration[53], but comes with a higher complexity O(|S|3) per iteration. Another advantage is that there is no restriction to a horizon. Policy iteration is guaranteed to converge and at convergence, the current policy and its value function are the optimal policy and the optimal value function.

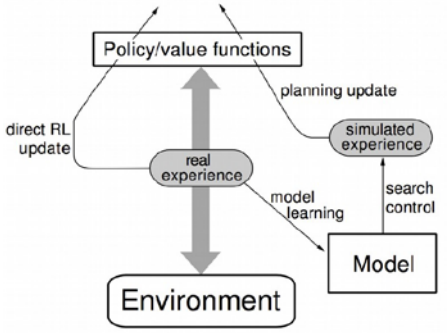
A more detailed discussion of the complexity of iterative methods of MDP can be found in[5,44,63,73]. Due to higher numerical costs several variants of policy iteration were discussed e.g. the modified policy iteration and asynchronous backups e.g.[56]. Both value iteration and policy iteration are standard algorithms for solving MDPs.

Approximation & Acceleration: Recently several new methods for finding optimal or approximately optimal policies for the MDP were invented, that intend to

cope with large system state spaces *S*, since these standard methods require excessive computation to get close-to-optimal solutions. Accelerating methods speed up the convergence of exact iterative methods by reducing the computational complexity e.g.[7,73,74,35,55]. Approximation methods give us the possibility to solve a wider class of MDP problems e.g.[13,12, 49, 76,8].

3.1.4 Dyna-Q

The Dyna-Q algorithm[59] integrates learning, acting, and planning, by not only learning from real experience, but also planning with simulated trajectories from a learned model. In this case the learning step uses real experience from the environment and planning step uses experience simulated by a model. This leads to the following architecture:



This idea leads to the following pseudocode:

Algorithm 3: Scheme of Dyna-Q

input: $\mathbb{P}, \mathcal{R}, \mathcal{S}$
Initialize $Q(s, a)$.
while *no uniform convergence in* $Q(s, s)$ **do**
  $a \leftarrow$ action for s, derived by Q, s, a e.g. ϵ-greedy
  observe *r'* and *s'*
  direct reinforcement learning step:
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$
  model learning step:
  $M(s, a) \leftarrow r, s'$
  planning step:
  for *i=1* **to** *N* **do**
    $a \leftarrow$ random state previously observed
    $s \leftarrow$ random action previously
    $r, s' \leftarrow M(s, a)$
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$
    $i++$
  end
end

Recall that in the planing step, the algorithm randomly samples only from stateaction pairs that have previously been experienced. Hence the learning model is never queried with a pair about which it has no prior information.

3.2 Model-free Algorithms

Obviously in real applications $\mathcal{P}$ or even $\mathcal{R}$ are unknown but must instead be estimated them from data. In other words the agent must learn from his experiences without having a model from the environment. But how do we know whether the action just taken is a good one, when it might have far reaching effects? How can we solve the prediction problem without MC-simulations?

3.2.1 Temporal Difference Learning

An obvious way of estimating the value function is to compute an average over multiple independent realizations started from the given state i.e. to use a MonteCarlo method (MC method). However the variance of the returns can be very high for MC methods. A second drawback is the interaction of algorithm and system: when estimation happens while interacting with the system, it might be impossible to reset the state of the system to some new state that is stochastically independent of its predecessor state. Hence a MC-method cannot be applied without introducing additional bias. Temporal Difference Learning (TD) address these issues by combining the dynamic programming approach with the MC-approach: there is no need for a model of the environment and updates are available at each state of the incremental procedure.

The stochastic approximation algorithm TD learns online and iteratively variants of the value functions directly from incomplete experience with some TD error. This methods differs from other approaches as it tries to minimize the error δ_t of temporal consecutive predictions instead of an overall prediction error. To that end TD uses some variance reducing bootstrap-methodology wrt. the sample data, since the update is based on an existing estimate. The simplest update rule for a value function in every state s is called TD($\lambda = 0$) and is for the case of the state value function given by:

$$v(s_t) \leftarrow v(s_t) + \alpha_t[r_{t+1} + \gamma v(s_{t+1}) - v(s_t)] \quad (14)$$

where $\alpha_t = ct^{-\eta}$ with $\eta \in [0.5,1]$, $v(s_{t+1}) - v(s_t)$ is the temporal difference and $c \geq 0$ denotes a learning rate at time *t*. The case of waiting only one step until the update is computed, is denoted by $\lambda = 0$. This parameter determines the trade-off between bias and variance of the update target and the trade-off has a large influence on the speed of learning. In (14) the state s_{t+1} can also be chosen according to some policy. Recall that TD($\lambda = 0$) must be computed after each transition between states. Obviously the choice of αt determines the convergence behavior since for (14) a fixpoint equation can be derived also[61].

Algorithm 4: Scheme of TD ($\lambda = 0$)

input: $\pi, \mathcal{R}, \mathcal{S}$
k=0
Initialize $v_\pi^{(k)}(s)$ arbitrarily, but according to π.
while *no uniform convergence in v*(*s*) **do**
  k=k+1
  for *t=1* **to** |*S*| **do**
    Choose at according to π for st.
    $v_\pi^{(k)}(s_t) \leftarrow v_\pi^{(k)}(s_t) + \alpha_t[r_{t+1} + \gamma v_\pi^{(k)}(s_{t+1}) - v_\pi^{(k)}(s_t)]$
    $s_t = s_{t+1}$
  end
end

In practice a constant step-size is used often a choice that is justified based on two grounds: First, the algorithm is often used in a non-stationary environment and second, for a small sample regime the algorithm the parameters converge in distribution. Hence the variance of the limiting distribution will be proportional to the step-size chosen. For methods that tune step-sizes automatically, see[49]. Here we introduce

$$\delta_t \quad := \quad r_t + \gamma v(s_{t+1}) - v(s_t)$$

denotes the so called TD error, that in fact is the target of the update rule. However it is always required that the reward or at least some utility of $s \rightarrow s'$ for the user is known. The obvious advantage is that TD can learn online after every step without knowing the final outcome. It is well known that TD has a low variance, but some bias and is sensitive to the initial values.

As a way to unify the Monte-Carlo-approach and TD($\lambda = 0$), λ can be chosen from [0, 1] that allows one to interpolate between the Monte-Carlo and TD($\lambda = 0$) approach[4]. This is helpful, when only partial knowledge of the state space is available or when function approximation is used to approximate the value functions in very large state spaces.

Multi-Step Bootstrapping: As a new feature TD(λ) for $\lambda \neq 0$ considers the so called multi-step return predictions

$$\mathbf{R}_{t:k} = \sum_{T=t}^{t+k} \gamma^{T-t} t_{T+1} + \gamma^{k+1} v_t(s_{T+k+1}) \tag{15}$$

where the mixing coefficients γ are functions of the exponential weights $(1-\lambda)\lambda^k$ with $k \geq 0$. TD($\lambda = 0$) converges to solution of max likelihood Markov model. Obviously by (15) so called eligibility traces of states s are introduced. The eligibility trace works as a short-term memory, usually lasting within an episode and assists the learning process, by affecting the weight vector. It helps with the issues of long-delayed rewards and non-Markov tasks.

Eligibility traces are used to speed up the slow convergence of the TD(λ). The slow convergence is due to the fact that only a single state-action pair is updated per time step. However in eligibility traces the value function is updated for all earlier states in the trajectory. Eligibility traces *z*(*s*) are not unique. Hence TD(λ) exists in correspondingly many multiple forms. As an example we note the TD(λ) update rule of the so called *accumulating traces*:

$$\begin{aligned} \delta_{t+1} &= r_{t+1} + \gamma v(s_{t+1}) - v_t(s_t) \\ z_{t+1}(s) &= \mathbf{1}_{\{S=s_t\}} + \gamma\lambda z_t(s) \\ v_{t+1}(s) &= v_t(s) + \alpha_t \delta_{t+1} z_{t+1}(s) \\ z_0(s) &= 0 \end{aligned}$$

The role of $z_t(s)$ is to modulate the influence of the TD error on the update of the value stored at state *s*. Recall that for $\lambda = 0$ we get

$$\lim_{\lambda \to 0}(1-\lambda)\sum_{k \geq 0} \lambda^k R_{t:k} = R_{t:0} = r_{t+1} + v_t(s_{t+1})$$

i.e. TD(0). On the other hand TD(1) corresponds to the Monte-Carlo-method. In practice, the best value of λ is determined by trial and error and can be changed even during the algorithm, without impacting convergence[64]. It is well known that *TD*($\lambda = 0$) converges in mean with probability equal one at higher rate than Monte-Carlo-Methods if α decreases sufficiently fast with the number of iterations[11].

Note that a new variant of temporal difference learning, the true online TD(λ) algorithm[2], has recently been proposed, that has better theoretical properties than conventional TD(λ) and has in faster learning also.

3.2.2 *Q*-Learning

Analogously, off-policy TD for control problems, the so called *Q*-learning, learns an optimal policy by approximating for a given a some variant of the well known action value function, with the following update rule for all states $s \in S$:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]. \quad (16)$$

It is well known[4], that for an operator $B : (\mathcal{S} \times \mathcal{A} \rightarrow \mathbf{R}) \rightarrow (\mathcal{S} \times \mathcal{A} \rightarrow \mathbf{R})$ with $Q_{t+1} = BQ_t$, B is a contraction mapping with $\|BQ_{t+1} - BQ_t\| \infty \leq \gamma \|Q_{t+1} - Q_t\| \infty$. This guarantees algorithmic convergence exponentially fast.

Obviously (16) can be interpreted as stochastic gradient descent where $\delta_t = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$ is the derivative of the Bellman error that measures the discrepancy between $Q^*(s, a)$ and

$$\sum_{s' \in \mathcal{S}} \mathbb{P}^a_{s,s'} \left(\mathcal{R}^a_{s,s'} + \max_{a' \in \mathcal{A}} Q^*(s', a') \right)$$

The learning rate α_t should decay (e.g., as $\alpha_t = 1/t$) as the number of updates goes to infinity[61]. The key observation is that unlike the optimal state values, the optimal action-values can be expressed as expectations, that allows one to estimate the action-values in an incremental manner. Also multi-step versions of *Q*-learning exist[4]. In a closed-loop situation, some frequently used strategies are to sample the actions following e.g. the s-greedy action selection scheme.

Recall that in an ϵ-greedy approach, an agent realizes a tradeoff between exploitation and exploration by selecting a so called greedy action s.t. $a = \arg\max_a Q(s, a)$ with probability $1 - \epsilon$, $\epsilon \in [0, 1]$ and selects in s a random action with probability ϵ. Thus the agent exploits the current value function estimation with probability $1 - \epsilon$ and explores with probability ϵ. Hence in the ϵ -greedy procedure, exploration and exploitation can be easily balanced. The disadvantage of the ϵ -greedy method is that very unfavorable actions can occur by chance. One way to avoid this is to select actions according to a probability distribution based on the already estimated $q(x, a)$ values. Greedy actions should have the highest execution probabilities. Procedures of this kind are called softmax procedures. At this time the action a is chosen according to the Boltzmann-distribution

$$\mathbb{P}_t(s, a) = \frac{\exp(Q(s, a)/\tau)}{\sum_{a' \neq a} \exp(Q(s, a')/\tau)}$$

where $\tau > 0$. If τ is large, then the action is chosen almost according to the uniform distribution. For $\tau \rightarrow 0$, the softmax procedure becomes the greedy procedure. The ϵ-greedy approach is motivated by the following theorem:

For every ϵ-greedy policy π, the s-greedy policy π' wrt. q_π is an improvement i.e. $v_{\pi'}(s) \geq v_\pi(s)$.

3.2.3 State-Action-Reward-State-Action

SARSA is an iterative TD-control method that learns a *Q*-function via action selection. Moreover SARSA balances between exploration and exploitation. The update rule for all states *S* reads as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right].$$

Note that if s_{t+1} is the last state, then $Q(s_{t+1}, a_{t+1})$ is set to zero. SARSA is an onpolicy algorithm which means that while learning the optimal policy it uses the current estimate of the optimal policy to generate the behavior. For a RobbinsMonroe sequence of step-sizes SARSA converges in the limit to an optimal policy as long as all state-action pairs are visited an infinite number of times. There are some modifications of that method in order to reduce its variance, see e.g.[60]. In comparison on-policy methods typically outperform off-policy methods.

Algorithm 5: Scheme of SARSA for On-Policy Control

input: $\mathcal{R}, \mathcal{S}$
Initialize $Q^{(k)}(s, a)$ arbitrarily. Set $k = 0$.
while *no uniform convergence in* $Q(s, a)$ **do**
  k=k+1
  Choose a_t according to π derived from $Q(s, a)$ (e.g. ϵ-greedy).
  for $t=1$ **to** $|S|$ **do**
    Take a_t and observe r_t and s'.
    Choose a_t according to π derived from $Q(s, a)$ (e.g. ϵ-greedy). $Q^{(k)}(s_t, a_t) \leftarrow Q^{(k)}(s_t, a_t) + \alpha_t[r_{t+1} + \gamma Q^{(k)}(s_{t+1}, a_{t+1}) - Q^{(k)}(s_t, s_t)]$
    $s_t = s_{t+1}$, $a_t = a_{t+1}$
  end
end

SARSA converges to the optimal action-value function under the following conditions: all state-action pairs are explored infinitely many time, the policy converges in a

greedy policy

$$\lim_{k\to\infty} \pi_k(a|s = \mathbf{1}_{a=\arg\max_{a'} Q_k(s,a')})$$

and the learning rate behaves like

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty.$$

In simulations it may become very difficult to fulfill these requirements.

4. Actor-Only Methods for finite MDPs

Actor-only methods search directly in policy space. Typically a class of policies is parameterized by a real-valued parameter vector θ. Fortunately a definition of such a class allows to integrate prior knowledge about the task and thus reduce the search complexity.

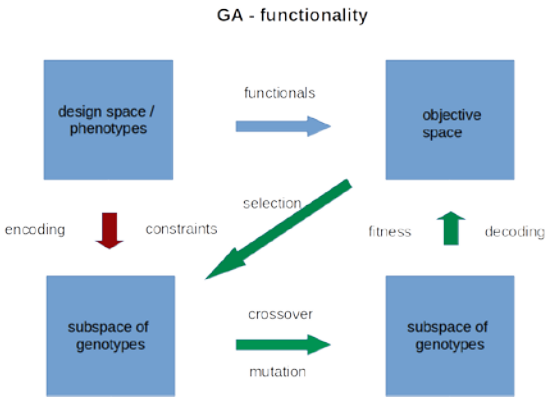
4.1 Model-free Methods

There are three major classes which emerged independently: genetic algorithms, evolutionary strategies[10], and genetic programming. Here for the sake of transparency we focus on the most popular genetic algorithms.

4.1.1 Evolutionary Algorithms

There are many classical, deep learning and hybrid combinations of evolutionary algorithms as a tool for finding good representations with polynomial complexity and approaches from the reinforcement learning algorithmic family, since they sometimes outperform TD-algorithms[42], can deal with partial observability and are in contrast to TD-methods independently from the Markov property. In this first part of the paper we restrict ourselves to classical approaches.[71] gives a survey of evolutionary methods to deep reinforcement learning approaches.

Evolutionary algorithms are randomized direct stochastic optimization algorithms inspired by concepts of evolution theory. They are robust methods dealing with a stochastic population of solution candidates, called phenotypes, that can be adapted by the suitable choice to evolutionary operators to the domain of every discrete optimization problem where the solution has a strict compositional structure[40]. To this end every problem needs to be translated to the evolutionary framework where the representations of the solution candidates are called *genotypes*:



Moreover the genetic operators *selection*, *crossover* and *mutation* must be tailored to the application on focus and convergence e.g. in genetic algorithms means an dramatic increase of homogeneity in the population of the solution candidates[10]. However the behavior of evolutionary algorithms heavily depends on the choice of the representation and the values of the hyperparameter that needs to be changed in every iteration. In fact their computation requires to solve a second optimization problem e.g. by an adaptive particle swarm method. The basic algorithms reads as:

Algorithm 6: (offline) Basic Genetic Algorithm

1)Encode the optimization problem *f* in terms of integer arrays.

2)Initialize a population of *N* solution candidates (chromosomes) randomly.

3)Calculate a fitness F_i for every individual *i* from the population.

4)Set population size, crossover and mutation rate and probability.**while** *some fitness value != termination criterion* **do**

> i)Preserve the best chromosomes of current population (elitism).
>
> ii)Selection of fitter chromosomes, skip unfit chromosomes.
>
> iii)Crossover of fitter chromosomes.
>
> iv)Break the dominance of elitists by mutation of some chromosomes.
>
> v)Increase rate of convergence by applying local search (hybrid) to generate final offspring.
>
> vi)Compute fitness of next generation of all solution candidates.

end

Get acceptable approximation of global minimum in polynomial time.

The general approach of combining evolutionary computing with reinforcement learning is a direct stochastic search in the space of policies for one the policy that maximizes the expected cumulative reward considered as objective.

In an early example[32] shows that a genetic algorithm can be applied to the problem of reinforcement learning by representing every action a in the language of the evolutionary framework by a gene's value. Then the evolutionary fitness of a policy reflects the expected accumulated fitness of $a \sim \pi_{theta.}$ Let k count the number of generations. Than the pseudocode reads as:

Algorithm 7: EARL

Initialize a population of policies $P(k)$ randomly.
Evaluate every policy from $P(k)$ in the objective space and set $k = 0$.
while *termination criterion is not fulfilled* **do**
  $k = k + 1$
  Select high-payoff policies for mating from $P(k - 1)$.
  Mate the selected policy according to some evolutionary strategy.
  Mutate some policies such that they are still feasible solutions. Merge $P(k - 1)$ and $P(k)$ according to some evolutionary strategy.
  Evaluate the merged $P(k)$.
end

In comparison to TD, that evaluates series of subsequent individual decisions, in evolutionary computing every single decision of an agent is evaluated independently. EARL is much better than a TD-method to cope with the occurence of rare states. However an obvious disadvantage of EARL is that this algorithms only works in offline-learning. Note that this kind of approach can be extended to the problem of cooperative co-evolution[48], where each chromosome represents a set of rules and at least two populations are evolved separately.

A more detailed survey of extending reinforcement learning algorithms to vast state spaces, partially observable environments, rarely occurring events and non-stationary environments is given in[16].

4.2 Model-based Methods

4.2.1 Policy Gradient Methods

Policy gradient methods learn a policy π parameterized by $\theta \in \mathbf{R}^d$, i.e.

$$\pi(a|s,\theta) = \mathbb{P}(a_t = a|s_t = s, \theta_t = \theta)$$

that selects actions without using additional information about the MDP, computing approximate estimations of gradients with respect to policy parameters. To this end a performance measure

$$J(\theta) := \mathbb{E}[Q_\pi(a,a)|\pi(a|s,\theta)]$$

is introduced. $J(\theta)$ is typically the value of the initial state $v_{\pi(\theta)}(s_{t=0})$. Then the gradient is given by

$$\omega_{t+1} = \omega_t + \alpha \nabla J(\omega_t)$$

The policy is often approximated by the so called Gibbs policy

$$\pi(a|s,\theta) = \frac{\exp(\theta^T)\phi(s,a)}{\sum_{a' \neq a} \exp(\theta^T)\phi(s,a')}$$

For that case the policy gradient theorem holds:

Let π and π' be deterministic policies with $Q(s, \pi'(s)) \geq v_\pi(s)$ for all $s \in S$, than $v_{\pi'}(s) \geq v_\pi(s)$ for all s. Moreover if $Q(s, \pi'(s)) > v_\pi(s)$ for one $s \in S$, than $v_{\pi'}(s') > v_\pi(s')$ for some $s' \in S$. Particularly this is true for $s' = s$.

This theorem applied to the policy gradient method using little algebra and the so called derivative trick $\nabla_\theta \pi(a|s, \theta) = \pi(a|s, \theta) \nabla_\theta \log \pi(a|s, \theta)$ gives the exact expression for the gradient:

$$\nabla J(\theta) = \sum_s d_\pi(s) \sum_a Q_\pi(s,a) \nabla_\theta \pi(a|s,\theta) \tag{17}$$

The performance gradient with respect to the policy parameters is estimated from interaction with the environment and the parameters are adapted by gradient ascent along $\nabla J(\theta)$. According to (17) only a sampling of this expression is needed.

Moreover the unknown value function $Q_\pi(s, a)$ in (17) can be replaced some approximator $f_\omega : \mathcal{S} \times \mathcal{A} \to \mathbf{R}^m$ with $\omega \in \mathbf{R}_m$ that satisfies the convergence condition

$$0 = \sum_s d_\pi(s) \sum_a \pi(s,a)[Q_\pi(s,a) - f_\omega(s,a)] \nabla_\omega f_\omega(s,a)$$

If f_ω satisfies the convergence condition and is compatible with the policy parametrization in the sense that it is linear in the corresponding features i.e.

$$\nabla_\omega f_\omega(s,a) = \nabla_\theta \pi(s,a)/\pi(s,a)$$

then

$$\nabla_\theta J(\pi) = \sum_s d_\pi(s) \sum_a \nabla_\theta \pi_\theta(s,a) f_\omega(s,a)$$

Different policy gradient methods[25,65,3] vary in the way the performance gradient is estimated and the value function is approximated. Cao provides an unified view of these effects based on a perturbation analysis[9].

Recall that at least an unbiased gradient can compute the right solution. Sufficient conditions for being unbiased are:

i) The value function approximator is compatible to the policy i.e.

$$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(a|s)$$

ii) The parameters *w* minimize the mean-squared error i.e.

$$\nabla_w \mathbb{E}_{\pi_\theta}[(Q_{\pi_\theta}(s,a) - Q_w(s,a))^2] = 0$$

where *w* denotes the parameter of the value function.

Compared with value-based methods, policy-based methods usually have better convergence properties, are more effective in high-dimensional or continuous action spaces, and can learn stochastic policies. However, policy-based methods usually converge at a speed that depends on the direction of the gradient to some local optimum. Furthermore a bad chosen step size leads to a bad policy, that controls the data sampling. Hence recovering is not guaranteed. Policy methods depend not on the the policy parametrization, but on the policy itself, they are more inefficient to evaluate, and typically encounter a high variance[18]. Other optimization methods can be used e.g. hill climbing, simplex methods or genetic algorithms. However gradient methods tend to be more successful e.g. in the case of the so called natural gradient methods that are also independent from the parametrization.

4.2.2 Reinforce

The policy gradient method REINFORCE[72] updates directly an deterministic or stochastic approximation $\pi(a|s;\theta)$ of the original policy in the direction of

$$\nabla_\theta \log \pi(a_t|s_t;\theta) R_t$$

where θ is the parameter vector of the approximation. This algorithm uses R_t as an unbiased sample of $Q(s_t, a_t)$. Given that the estimation of the gradient is unbiased, some advanced stochastic optimization techniques e.g. the stochastic gradient descent method converge to a local optimum. However trapping-problems and its numerical complexity make the algorithm in its naive form less attractive.

Since the rate of the convergence depends on the variance of the method a baseline $b_t(s_t)$, that is independent from action a, is subtracted in order to reduce the variance and hence accelerate its convergence. This is motivated by

$$\begin{aligned} 0 &= \nabla_\theta 1 = \nabla_\theta \int_{\mathcal{A}} \pi(s|a,\theta)da = \int_{\mathcal{A}} \nabla_\theta \pi(s|a,\theta)da \\ &= \int_{\mathcal{A}} \pi(s|a,\theta)\nabla_\theta \log \pi(s|a,\theta)da = \mathbb{E}_\pi\left[\nabla_\theta \log \pi(s|a,\theta)\right] \\ &= \mathbb{E}_\pi\left[b_t(s_t)\nabla_\theta \log \pi(s|a,\theta)\right] \end{aligned}$$

Hence we get as update step:

$$\nabla_\theta \log \pi(a_t|s_t;\theta)(Q(s_t,a_t) - b_t(s_t))$$

where $b_t(s_t)$ is chosen s.t. the expectation is shifted to zero. $b_t(s_t)$ can be estimated by Monte-Carlo-methods. This leads to the following pseudocode:

Algorithm 8: REINFORCE with baseline (episodic)

input: $\pi(a|s;\theta)$, $\hat{v}(s,w)$

parameter: $0 < \alpha, \beta$

output: $\pi(a|s;\theta)$

Initialize policy parameter θ and state value weights w

while *no uniform convergence* **do**

  generate a random episode $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T \sim \pi(\cdot|\cdot,\theta)$

  for $t=0$ to $T-1$ **do**

    $R_t \leftarrow$ return from step t

    $\delta \leftarrow Rt - \hat{v}(s,w)$

    $w \leftarrow w + \beta\delta\nabla_w \hat{v}(s_t,w)$

    $\theta \leftarrow \theta + \alpha\gamma^t\delta\nabla_\theta \log \pi(a_t|s_t,\theta)$

  end

end

4.2.3 Trust-Region Policy Optimization

In general abandoning the experience of former updates is not advisable. Hence let us consider some former estimated policy π^{old} and define the advantage functio wrt. π by $A_\pi(s,a) := Q_\pi(s,a) - V_\pi(s)$. The advantage function can significantly reduce the variance of policy gradient by the prize of some bias[20]. Additionally we can make another approach to the computation of the gradient of the performance measure using

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathcal{T}\sim\pi}\left[\frac{d_\pi(s)\pi(a|s,\theta)}{d_\pi(s)\pi^{old}(a|s)}\sum_t \nabla_\theta \pi(a_t|s_t,\theta)A_\pi(s,a)\right]$$

However the occupation frequencies are unknown. But we can compute the update in the quality measure. Then we get the main idea of the Trust-Region Policy

Optimization (TRPO)[21]:

$$
\begin{aligned}
& J(\pi) - J(\pi^{old}) \\
&= \mathbb{E}_{\mathcal{T}\sim\pi}\left[\sum_{t=0}\gamma^t r(s_t) - J(\pi^{old})\right] = \mathbb{E}_{\mathcal{T}\sim\pi}\left[\sum_{t=0}\gamma^t r(s_t) - V_{\pi^{old}}(s_0)\right] \\
&= \mathbb{E}_{\mathcal{T}\sim\pi}\left[\sum_{t=0}\gamma^t r(s_t) - \sum_{t=0}\left[\gamma^{t+1}V_{\pi^{old}}(s_{t+1}) - \gamma^t V_{\pi^{old}}(s_t)\right]\right] \\
&= \mathbb{E}_{\mathcal{T}\sim\pi}\left[\sum_{t=0}\gamma^t\Big(r(s_t) + \gamma V_{\pi^{old}}(s_{t+1}) - V_{\pi^{old}}(s_{t+1})\Big)\right] \\
&= \mathbb{E}_{\mathcal{T}\sim\pi}\left[\gamma^t\Big(Q_{\pi^{old}}(s,a) - V_{\pi^{old}}(s)\Big)\right] = \mathbb{E}_{\mathcal{T}\sim\pi}\left[\gamma^t A_{\pi^{old}}(s,a)\right]
\end{aligned}
$$

This is called the relative policy performance identity. It states that if we substitute the reward by the advantage function than independently of *pi* this will shift the objective by a constant. Using the discounted state visitation frequency

$$d(s|\pi) := (1-\gamma)\sum_{t=0}\gamma^t\mathcal{P}(s_t = s)$$

where s_t is chosen according to π, i.e. $T \sim \pi$, the relative policy performance identity can be rewritten as

$$J(\pi) - J(\pi^{old}) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d(s|\pi)}\mathbb{E}_{a\sim\pi(a|s)}A_{\pi^{old}}(s,a)$$

Now by applying importance sampling using the assumption $d(s|\pi_\theta) = d(s|\pi_\theta^{old})$, we get:

$$J(\pi) - J(\pi^{old}) \approx L_{\pi^{old}}(\theta) := \frac{1}{1-\gamma}\mathbb{E}_{s\sim d(s|\pi)}\mathbb{E}_{a\sim\pi(a|s)}\frac{\pi^\theta(a|s)}{\pi^{old}(a|s)}A_{\pi^{old}}(s,a)$$

The approximation quality of $L_{\pi old}(\theta)$ is determined by

$$J(\pi) - J(\pi^{old}) \approx L_{\pi^{old}}(\theta) := \frac{1}{1-\gamma}\mathbb{E}_{s\sim d(s|\pi)}\mathbb{E}_{a\sim\pi(a|s)}\frac{\pi^\theta(a|s)}{\pi^{old}(a|s)}A_{\pi^{old}}(s,a) \tag{18}$$

where *KL* is the Kullback-Laibler divergence. The implication of (18) is

$$J(\pi) - J(\pi^{old}) \geq L_{\pi^{old}}(\theta) - C\max_s KL(\pi^{old}||\pi_\theta)[s]$$

which means that we are interested in solving the optimization problem

$$\theta_{k+1} = \arg\max_\theta L_{\pi_{theta_k}}(\theta) - C\max_s KL(\pi^{old}||\pi_\theta)[s] \tag{19}$$

where *C* is unknown. The remaining task is to rewrite (19) as an constrained optimization problem, that can be solved by a trust-region method. Recall that here the Kullback-Laibler divergence can be sampled by a Monte-Carlo-Method.

4.2.4 Proximal Policy Optimization

A second way to solve (19) was proposed in[19] by the Proximal Policy Optimization (PPO) approach:

$$\mathbb{E}_{\pi^{old}}\left[\frac{\pi_\theta(a|s)}{\pi^{old}(a|s)}A_{\pi^{old}}(s,a) - C\max_s KL(\pi^{old}||\pi_\theta)[s]\right] \to \max_\theta$$

Since the computation of the Hessian is a numerical bad conditioned problem and the importance sampling coefficients

$$r(\theta) := \frac{\pi_\theta(a|s)}{\pi^{old}(a|s)}$$

tend to an unbounded growth, PPO uses clipping as a penalty for importance sampling i.e.

$$r^{clip}(\theta) := clip(r(\theta), 1-\epsilon, 1+\epsilon)$$

and switches to

$$\mathbb{E}_{\pi^{old}}\left[\min\Big(r(\theta)A_{\pi^{old}}(s,a), r(\theta)^{clip}A_{\pi^{old}}(s,a)\Big) - C\max_s KL(\pi^{old}||\pi_\theta)[s]\right] \to \max_\theta$$

Then using a stochastic gradient method wrt. θ leads to a much more stable procedure. However numerical experiments show that the influence of the KLterm is rather small.

5. Actor-Critic Methods for finite MDPs

We have seen in the last sections, that on the one hand in actor-only methods policies were directly modified with high variance and value function evaluation is dispensed. In general there are two ways in policy-based approaches for improvement: greedy improvement, where the current policy is moved towards the greedy policy underlying the *Q*-function estimate obtained from the critic, and policy gradient, that perform stochastic gradient descent on the performance surface of the parameterized policy. On the other hand low-variance critic-only algorithms evaluate value functions and a policy is only implicitly used. One drawback of actor-only methods is, that a new gradient is estimated independently of past estimates such that no learning in the sense of accumulation and consolidation of older information occurs. A well known disadvantage of critic-only

methods is that they do not try to optimize directly over a policy space and hence lack reliable guarantees in terms of near-optimality of the resulting policy.

Actor-critic architectures combine the vantages both approaches. They learn a value function and a policy, while the actor and critic are both represented explicitly and learned separately. The critic updates action-value function parameters e.g. by TD(λ) and the actor updates policy parameters in a direction suggested by the critic e.g. by policy gradient[65].

To be more precise about the general scheme the actor-critic algorithms implement a generalized policy iteration, alternating between a policy evaluation and a policy improvement step: Some critic evaluates the responses, estimates the value of the current policy and updates the action-value function:

$$w \leftarrow w + \alpha\delta\nabla_\theta V_w(a_t|s_t)$$

where *w* here denotes the parameter of the value function and where δ is the estimated error in evaluating the state s. The actor is responsible for generating control and updates the policy in the direction suggested by the critic:

$$w \leftarrow w + \alpha\delta\nabla_\theta V_w(a_t|s_t)$$

where G_t is the evaluation of long-term returned by the critic for st. In other words the role of the critic is to predict and the role of the actor is to evaluate and estimate the Q-function. This is illustrated in the next figure:

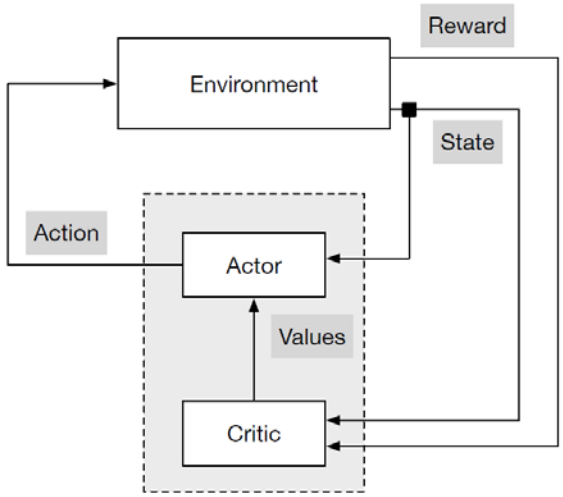


Figure 2. Actor-critic RL-architecture.

This architecture shows that in fact there are two different policies in use: the behavior policy of the actor is used to generate the samples and the typically stochastic target policy is evaluated by the critic and improved by the actor[1].

On the one hand these policies should not be identical. Hence update steps can make things worse. On the other hand this allows the critic to learn about the actions not preferred by the target policy and therefore improve the target policy. Since the actor uses *Q*-values to choose actions, the critic must estimate the *Q*-function. Note that the greater the difference between these policies is, the better is the performance of actor-critic method[52]. If the critic is modeled by a bootstrapping method this reduces the variance leading to a more stable behavior of the algorithm.

For small $\boldsymbol{S}$ the critic is a *Q*-function estimator and the actor is ϵ-greedy or a Boltzmann policy estimated using tabulars. For large $\boldsymbol{S}$ the critic and the actor use function approximation. In general actor-critic methods work with deterministic and stochastic policies, have better convergence properties and are more effective in high dimensional spaces. Moreover the policy space can be tailored to the problem. Recall that actor-critic architectures are more like a framework and can be combined with other approaches. Combined e.g. with a policy gradient method they will also have a high variance and might be expensive to compute.

Since the actor-critic architecture in fact is a special type of strategy that can be used to combine several types of already discussed approaches to solve subproblems occurring in reinforcement learning, in this section we restrict ourselves to the structural changes in this strategy and refer the reader to the details in the sections above.

5.1 Model-based Algorithms

One-Step-Actor-Critic (QAC): Note that θ always denotes the parameter of the policy and w the value function.

Algorithm 9: basic actor-critic scheme (episodic)

input: $\pi(a|s;\theta)$, $\hat{v}(s, w)$, $0 < \alpha, \beta$
output: $\pi(a|s;\theta)$
Initialize s, $I \leftarrow 1$
while *no uniform convergence* **do**
  for $t=0$ **to** $T-1$ **do**
    take action $a \sim \pi(\cdot|s, \theta)$, observe s', r
    $\delta \leftarrow r + \gamma\hat{v}(s', w) - \hat{v}(s, w)$ (if s' is terminal, $\hat{v}(s, w) = 0$)
    $w \leftarrow w + \beta\delta\nabla_w\hat{v}(s_t, w)$
    $\theta \leftarrow \theta + \alpha_t I\delta\nabla_\theta \log_\pi(a_t|s_t, \theta)$
    $I \leftarrow \gamma I$, $s \leftarrow s'$
  end
end

Sometimes this scheme also is known as one-step-actor-critic (QAC). For convergence it is required that the critic's estimate at least is asymptotically accurate. This condition is fulfilled if the step sizes are deterministic, non-increasing and satisfy the well known conditions[65]:

$$\sum_t \alpha_t = \infty \qquad \sum_t \alpha_t^2 < \infty$$

When the number of policies is small compared to the number of states, it is not useful that the critic attempts to compute the exact value function but a projection of the value function onto a low-dimensional subspace spanned by a set of basis functions determined by the parametrization of the actor.

Actor-critic architectures tend to be unstable due to an inaccurate step size adversely affecting the other and thus destabilize the learning. Recently[27] propose to regularize the step size of the actor by penalizing the TD-error of an highly inaccurate critic.

Advantage-Actor-Critic (A2C): A little more sophisticated actor-critic architecture emerges, if the critic is supposed to compute e.g. using TD-learning an advantage function $A\pi(s, a)$ already discussed above by estimating some approximations

$$V_\pi(s) \approx V_{w'}(s) = (w')^T\phi$$
$$Q_\pi(s,a) \approx Q_w(s,a) = (w)^T\psi$$

where ϕ and ψ denote the features. Since for the true $V\pi(s)$ the TD-error is an unbiased estimate of the advantage function, it can be used to compute the policy gradient, which only requires the w' parameters. What comes up, is the well known REINFORCE algorithm with the baseline correction described above.

Asynchronous Advantage-Actor-Critic (A3C): Recently in[66] a third architecture was proposed based on the idea that data sampling can be parallelized using several copies of the same agent using the same basic actor-critic approach. In a second step all computed gradients were passed to a main network that updates another actor-critic-pair using all these decorrelated gradients. Furthermore different exploration policies may be in use to maximize diversity.

There are further new actor-critic approaches published e.g.[62,67,28] that to discuss in detail is beyond the scope of this paper.

5.2 Model-free Algorithms

In[31] a robust Bootstrapped Dual Policy Iteration (BDPI) for continuous states and discrete actions, with an actor and several off-policy critics is introduced.

However, their approach uses a deep-Q-network. Since the introduction of deep learning approaches to reinforcement learning is postponed to the second part of this paper, we will skip the characterization of their work here and shift the discussion to the second part. Moreover[15] proposed an approach for an nonMarkovian domain, that also is beyond the here presupposed framework and must be done in another paper.

Although reinforcement learning already is a rich family of approaches and algorithms, obviously there are many open opportunities for future research.

References

1. A. Barto & R. Sutton & C. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 834 – 846, 1983.
2. H.v. Seijen & R. Sutton et. al. True online temporal-difference learning. *Journal of Machine Learning Research*, pages 1–40, 2016.
3. J. Baxter & P.L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, pages 319–350, 2001.
4. R. Sutton & A. Barto. Reinforcement Learning: An Introduction. 2018.
5. R. E. Bellman. *Dynamic Programming*. 1957.
6. D. Bertsekas. Dynamic programming and optimal control, Vol II. 2012.
7. D. P. Bertsekas. A new value iteration method for the average cost dynamic programming problem.

SIAM Journal on Control and Optimization, 36:742– 759, 1998.
8. H. Yu & D. P. Bertsekas. Basis function adaptation methods for cost approximation. Proc. of IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning, 2009.
9. X.R. Cao. A basic formula for online policy gradient algorithms. *IEEE Transactions on Automatic Control*, 50:696–699, 2005.
10. C. Coello Coello. *Evolutionary Algorithms for Solving Multi-Objective Problems.* 2007.
11. F. Dayan. The convergence of td(λ) for general λ. *Machine Learning*, 8:341– 362, 1992.
12. D. de Farias & B. V. Roy. On the existence of fixed points for appproximate value iteration and temporal-difference learning. *Journal of Optimization Theory and Applications*, 105:589–608, 2000.
13. D. de Farias & B. V. Roy. The linear programming approach to approximate dynamic programming. Operations Research, 51:850–856, 2003.
14. C. Derman. Finite State Markovian Decision Processes. 1970.
15. E. Mizutani & S. Dreyfus. Totally model-free actor-critic recurrent neuralnetwork reinforcement learning in non-markovian domains. *Annals of Operations Research*, pages 107–131, 2017.
16. M. Drugan. Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms. Swarm and Evolutionary *Computation*, pages 228–246, 2019.
17. O. Sigaud & O. Buffet (eds.). *Markov Decision Processes in Artificial Intelligence.* 2010.
18. D. Silver & A. Huang & C. Maddison *et al*. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.
19. J. Schulman & F. Wolski & P. Dhariwal et. al. Proximal policy optimization algorithms. page arxiv prepreint: https://arxiv.org/abs/1707.06347, 07 2017.
20. J. Schulman & P. Moritz & Pieter Abbeel et. al. High-dimensional continuous control using generalized advantage estimation. *ICML*, 2016.
21. J. Schulman & S. Levine & P. Abbeel et. al. Trust region policy optimization. *ICML*, pages 1889 – 1897, 2015.
22. L. Busoniu & R. Babuka et. al. *Reinforcement learning and dynamic programming using function approximators.* 2009.
23. L. C. Thomas & R. Harley et. al. Computational comparison of value iteration algorithms for discounted markov decision processes. *Operations Research Letters*, 2:72–76, 1983.
24. M. Ghavamzadeh & S. Mannor et. al. Bayesian reinforcement learning: A survey. Foundations and *Trends in Machine Learning*, 8:359–492, 2015.
25. R.S. Sutton & D. McAllester et. al. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, pages 1057–1063, 2000.
26. S. Levine & C. Finn et. al. End-to-end training of deep visuomotor policies. J*ournal of Machine Learning Research*, 17:1–40, 2016.
27. S. Parisi & V. Tangkaratt & J. Peters et. al. Td-regularized actor-critic methods. *Machine Learning*, pages 1–35, 2019.
28. T. Haarnoja & A. Zhou & P. Abbeel et. al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.
29. V. Mnih & K. Kavukcuoglu et. al. Playing atari with deep reinforcement learning. *Technical report, Google DeepMind*, page http://arxiv.org/abs/1312.5602, 2013.
30. Vincent Franc¸ois-Lavet & Peter Henderson et. al. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, page DOI: 10.1561/2200000071, 2018.
31. D. Steckelmacher & H. Plisnier & M. Diederik et.al. Sample-efficient modelfree reinforcement learning with off-policy critics. *European Conference on Machine Learning*, 2019.
32. D. Moriarty & A. Schultz & J. Grefenstette. Evolutionsary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, pages 241–276, 1999.
33. R.A. Howard. Dynamic Programming and Markov Processes. 1960.
34. S. Kim & M. E. Lewis & C. C. White III. Optimal vehicle routing with realtime traffic information. *IEEE Transactions on Intelligent Transportation Systems*, 6:178–188, 2005.
35. H. J. Kushner & A. J. Kleinman. Accelerated procedures for the solution of discrete markov control problems. *IEEE Transactions on Automatic Control*, 16:147–152, 1971.
36. V. Krishnamurthy. Partially Observed Markov Decision Processes. 2016.
37. Y. Adachia & T. Nosea & S. Kuriyama. Optimal inventory control policy subject to different selling prices of perishable commodities. *International Journal of Production Economics*, 60-61:389–394, 1999.
38. D. Vrabie & K. G. Vamvoudakis & F. L. Lewis. *Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles.* 2013.
39. Yuxi Li. Deep reinforcement learning: An overview. CoRR, abs/1701.07274, 2017.
40. S. Luke. *Essentials of Metaheuristics.* Lulu, second edition, 2013.
41. R.-R. Chen & S. Meyn. Value iteration and optimization of multiclass queueing networks. *Queueing Systems*, 32:65–97, 1999.
42. K. Stanley & R. Mikkulainen. *Evolving a roving eye for go.* 2004.
43. L. Kaelbling & M. Littman & A. Moore. Reinforcement learning: A survey. *Journal of Artificial*

Intelligence Research, 4:237–285, 1996.
44. M. L. Littman & J. Goldsmith & M. Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.
45. J. Kober & J. Peters. Policy search for motor primitives in robotics. *Advances in neural information processing systems*, pages 849–856, 2009.
46. M. Deisenroth & G. Neumann & J. Peters. A survey on policy search for robotics. *Robotics: Foundations and Trends*, 2:1–142, 2013.
47. S. V. Amari & L. McLaughlin & Hoang Pham. Cost-effective condition-based maintenance using markov decision processes. Proceedings of the RAMS. *Annual Reliability and Maintainability Symposium*, pages 464–469, 2006.
48. M. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. George Mason University, 1997.
49. A. George & W. Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, 65:167– 198, 2006.
50. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1994.
51. Reazul Hasan Russel. A short survey on probabilistic reinforcement learning. *arxiv*, page 1901.07010v1, 2019.
52. J. Peters & S. Schaal. Natural actor-critic. *Neurocomputing*, 71:1180–1190, 2008.
53. C. W. Zobel & W. T. Scherer. An empirical study of policy convergence in markov decision process value iteration. *Computers and Operations Research*, 32:127–142, 2005.
54. J. Schmidhuber. *A general method for multi-agent learning and incremental selfimprovement in unrestricted environments.*, chapter Yao, X. (ed.), Evolutionary Computation: Theory and Applications. 1996.
55. D. Wingate & K. D. Seppi. Prioritization methods for accelerating mdp solvers. *Journal of Machine Learning Research*, 6:851–881, 2005.
56. M. L. Puterman & M. C. Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 64:1127–1137, 78.
57. E.A. Feinberg & A. Shwartz. *Handbook of Markov Decision Processes: Methods and Applications*, chapter Total Reward Criteria. Kluwer, 2007.
58. M. Sugiyama. *Statistical Reinforcement Learning*. 2015.
59. R. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*, 1990.
60. S. Singh & R. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.
61. C. Szepesvari. *Algorithms for Reinforcement Learning*. 2010.
62. A. Pritzel *et al*. T. Lillicrap & J. Hunt. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016.
63. C. H. Paradimitriou & J. N. Tsitsiklis. The complexity of markov decision processes. *Mathematics of Operations Research*, 12:441–450, 1987.
64. D.P. Bertsekas & J.N. Tsitsiklis. *Neuro-Dynamic Programming*. 2006.
65. V.R. Konda & J.N. Tsitsiklis. On actor-critic algorithms. *SIAM Journal on Control and Optimization*, pages 1143–1166, 2003.
66. A.Badia & M.Mirza et. al. V. Mnih. Asynchronous methods for deep reinforcement learning. *Proceedings of The 33rd International Conference on Machine Learning*, pages 1928–1937, 2016.
67. S. Fujimoto & H. van Hoof & D. Meger. Addressing function approxima approximation error in actor-critic methods. in proceedings of the international conference on machine learning. *Conference paper ICML*, 2018.
68. M. Wiering & M. van Otterlo (eds.). Reinforcement Learning. 2012.
69. N. Gans & G. van Ryzin. Dynamic vehicle dispatching: Optimal heavy traffic performance and practical insights. *Operations Research*, 47:675–693, 1999.
70. D. J. White. *Markov Decision Processes.* 1994.
71. S. Whiteson. *Evolutionary Computation for Reinforcement Learning*, pages 325–355. Springer, 2012.
72. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229 – 256, 1992.
73. Y. Ye. A new complexity result on solving the markov decision problem. *Mathematics of Operations Research*, 2005:733–749, 38.
74. M. Herzberg & U. Yechiali. A k-step look-ahead analysis of value iteration algorithms for markov decision processes. *Europian Jornal of Operations Research*, 88:622–636, 1996.
75. Q. Hu & W. Yue. *Markov Decision Processes with Their Applications.* 2008.
76. M. A. Trick & S. E. Zin. Spline approximations to value functions. *Macroeconomic Dynamics*, 1:255–277, 1997.