Algorithmic Survey of Parametric Value Function Approximation

Matthieu Geist and Olivier Pietquin, *Senior Member, IEEE*

***Abstract*— Reinforcement learning (RL) is a machine learning answer to the optimal control problem. It consists of learning an optimal control policy through interactions with the system to be controlled, the quality of this policy being quantified by the so-called value function. A recurrent subtopic of RL concerns computing an approximation of this value function when the system is too large for an exact representation. This survey reviews state-of-the-art methods for (parametric) value function approximation by grouping them into three main categories: bootstrapping, residual, and projected fixed-point approaches. Related algorithms are derived by considering one of the associated cost functions and a specific minimization method, generally a stochastic gradient descent or a recursive least-squares approach.**

***Index Terms*— Reinforcement learning (RL), survey, value function approximation.**

I. Introduction

REINFORCEMENT learning (RL) [1]–[7] addresses the problem of the optimal control of a stochastic dynamic system. In this paradigm, an artificial agent learns an optimal control policy through interactions with the dynamic system (also considered as its environment). After each interaction, the agent receives an immediate scalar reward information and the optimal policy it searches for is the one that maximizes the expected cumulative reward over the long term. The system to be controlled is usually modeled as a Markovian decision process (MDP). An MDP is made up of a set of states (the different configurations of the system), a set of actions (which cause a change of the systems state), a set of Markovian transition probabilities (the probability to transit from one state to another under a given action; the Markovian property states that the probability depends on the current state–action pair and not on the path followed to reach it), a reward function associating a scalar to each transition, and a discounting factor that decreases long-term rewards influence. How the agent acts with the system is modeled by a policy which associates to each state a probability distribution over actions. The quality of such a policy is quantified by a value function which associates to each state the expected cumulative discounted reward from starting in the considered state and then following the given policy (expectation being done over all possible trajectories). An optimal policy is one that maximizes the associated value function for each state.

Thanks to the Markovian property, value functions can be (more or less easily) computed using the Bellman equations. The value function of a given policy satisfies the (linear) Bellman evaluation equation, and the optimal value function (which is linked to one of the optimal policies) satisfies the (nonlinear) Bellman optimality equation. These Bellman equations are very important for RL, as they allow computing the value function.

If the model (i.e., transition probabilities and the reward function) is known and if the state and action spaces are small enough, the optimal policy can be computed using dynamic programming. A first scheme, called policy iteration, consists of evaluating an initial policy (i.e., computing the associated value function using the linear Bellman evaluation equation) and then improving this policy, the new one being greedy with respect to the computed value function (it associates to each state the action that maximizes the expected cumulative reward obtained from starting in this state, applying this action, and then following the initial policy). Evaluation and improvement steps are iterated until convergence (which occurs in a finite number of iterations). A second scheme, called value iteration, consists of computing directly the optimal value function (using the nonlinear Bellman optimality equation and an iterative scheme based on the fact that the value function is the unique fixed point of the associated Bellman operator). The optimal policy is greedy with respect to the optimal value function.

RL aims at estimating the optimal policy without knowing the model and from interactions with the system. Value functions can no longer be exactly computed; they have to be estimated, which is the main scope of this paper. RL heavily relies on dynamic programming, in the sense that most approaches are some sorts of generalization of value or policy iteration. A first problem is that computing a greedy policy (required for both schemes) from a value function requires the model to be known. The action-value (or Q-) function alleviates this problem by encoding the value of state–action couples (instead of states only). It is defined, for a given policy and for a state–action couple, as the expected discounted cumulative reward starting in the given state, applying the given action, and then following the fixed policy. A greedy policy can thus be obtained by maximizing the Q-function over actions.

Manuscript received May 6, 2012; revised January 10, 2013; accepted February 11, 2013. Date of publication March 7, 2013; date of current version April 5, 2013. This work was supported in part by the EU FP7 FET Project ILHAIRE under Grant 270780 and the Région Lorraine, France.

M. Geist is with the IMS-MaLIS Research Group, Supélec, Metz 57070, France (e-mail: matthieu.geist@supelec.fr).

O. Pietquin is with the IMS-MaLIS Research Group, Supélec, Metz 57070, France, and also with the UMI 2958 (GeorgiaTech-CNRS) (e-mail: olivier.pietquin@supelec.fr).

Digital Object Identifier 10.1109/TNNLS.2013.2247418

There are two main approaches to estimate an optimal policy through value functions. The first one, based on value iteration, consists of estimating directly the optimal action–value function, which is then used to derive an estimate of the optimal policy (with the drawback that errors in the Q-function estimation can lead to a bad derived policy). The second one, based on policy iteration, consists of mixing the estimation of the Q-function of the current policy (policy evaluation) with policy improvement in a generalized policy iteration scheme (generalized in the sense that evaluation and improvement processes interact, independently of the granularity and other details). This scheme presents many variations. Generally, the Q-function is not perfectly estimated when the improvement step occurs (optimistic policy iteration). Each change in the policy implies a change in the associated Q-function; therefore, the estimation process can be nonstationary. The control policy can be derived from the estimated action-value function (e.g., using a Boltzmann distribution or an ϵ-greedy policy). There is generally an underlying dilemma between exploration and exploitation. At each time step, the agent should decide between acting greedily with respect to its uncertain and imperfect knowledge of the world (exploitation) and taking another action that improves this knowledge and possibly leads to a better policy (exploration). The policy can also have its own representation, which leads to actor–critic architectures (which are a form of policy iteration). The actor is the policy, and the critic is an estimated value or Q-function that is used to correct the policy representation.

All these approaches share a common subproblem: estimating the (action-) value function, of a given policy (evaluation operator) or the optimal one (optimality operator) directly. This issue is even more involved when state or action spaces are too large for a tabular representation, which implies the use of some approximate representation. Generally speaking, estimating a function from samples is addressed by the supervised learning paradigm. However, in reinforcement learning, the (action-) values are not directly observed, which renders the underlying estimation problem more difficult. Despite this, a number of (action-) value function estimation algorithms have been proposed in the past decades. The aim of this paper is to review a panel as large as possible of these algorithms by adopting a unifying view that classifies them into three main categories: bootstrapping approaches (Section III), residual approaches (Section IV), and projected fixed-point approaches (Section V). Each of them is related to a specific cost function, and algorithms are derived considering one of these costs and a specific way to minimize it (almost always a stochastic gradient descent or a recursive least-squares approach). Sections III–V are the core of this survey on parametric value function approximation, but additional important topics are briefly covered at the end of the article. Section VI shows how these approaches can be embedded in the more general control problem, Section VII discusses when choosing what algorithm (among surveyed methods and depending on the practical context), and Section VIII provides a brief [illegible] function not [illegible] parametric approaches or extensions to eligibility traces). The underlying formalism is presented in Section II.

II. Preliminaries

An MDP is a tuple $\{S, A, P, R, \gamma\}$, where S is the (finite) state space, A the (finite) action space, P: $s, a \in S \times A \to p(.|s,a) \in \mathcal{P}(S)$ the family of Markovian transition probabilities, R: $s, a, s' \in S \times A \times S \to r = R(s,a,s') \in \mathbb{R}$ the bounded deterministic reward function, and γ the discount factor weighting long-term rewards. According to these definitions, the system stochastically steps from state to state conditionally to the actions the agent performed. Let i be the discrete time step. To each transition (s_i, a_i, s_{i+1}) is associated an immediate reward r_i. The action selection process is driven by a policy $\pi: s \in S \to \pi(.|s) \in \mathcal{P}(A)$.

A. Bellman Equations

The quality of a policy is quantified by the value function $V^\pi \in \mathbb{R}^S$, defined as the expected discounted cumulative reward starting in a state s and then following the policy π

$$V^\pi(s) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, \pi\right].$$

Thanks to the Markovian property, the value function of a policy π satisfies the linear Bellman evaluation equation[1]

$$V^\pi(s) = \mathbb{E}_{s',a|s,\pi}[R(s,a,s') + \gamma V^\pi(s')]. \quad (1)$$

Let us define the Bellman evaluation operator $T^\pi: V \in \mathbb{R}^S \to T^\pi V \in \mathbb{R}^S$

$$[T^\pi V](s) = \mathbb{E}_{s',a|s,\pi}[R(s,a,s') + \gamma V(s')].$$

The operator T^π is a contraction and V^π is its unique fixed point

$$V^\pi = T^\pi V^\pi.$$

An optimal policy π^* maximizes the associated value function for each state: $\pi^* \in \text{argmax}_{\pi \in \mathcal{P}(A)^S} V^\pi$. The associated optimal value function, written as V^*, satisfies the nonlinear Bellman optimality equation

$$V^*(s) = \max_{a \in A} \mathbb{E}_{s'|s,a}[R(s,a,s') + \gamma V^*(s')].$$

The optimal value function is unique, but this is not necessarily the case for the optimal policy. Let us define the Bellman optimality operator $T^*: V \in \mathbb{R}^S \to T^*V \in \mathbb{R}^S$, as

$$[T^*V](s) = \max_{a \in A} \mathbb{E}_{s'|s,a}[R(s,a,s') + \gamma V(s')]. \quad (2)$$

The operator T^* is a contraction and V^* is its unique fixed point

$$V^* = T^*V^*.$$

The action-value (or Q-) function encodes the values of the state–action couples. This is useful in a model-free context.

[1] $\mathbb{E}_{x_1|x_2}$ denotes the expectation according to random variable x_1 conditioned on x_2

It is defined as the expected cumulative reward starting in s, applying a, and then following π:

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi\right].$$

The action-value function Q^π also satisfies the linear Bellman evaluation equation

$$Q^\pi(s,a) = \mathbb{E}_{s',a'|s,a,\pi}[R(s,a,s') + \gamma\, Q^\pi(s',a')].$$

It is clear that value and action-value functions are directly linked

$$V^\pi(s) = \mathbb{E}_{a|s,\pi}[Q^\pi(s,a)].$$

A Bellman evaluation operator related to the Q-function can also be defined. By a slight abuse of notation, it is also written as T^π, the distinction being clear from the context ($T^\pi\colon Q \in \mathbb{R}^{S\times A} \to T^\pi Q \in \mathbb{R}^{S\times A}$)

$$[T^\pi Q](s,a) = \mathbb{E}_{s',a'|s,a,\pi}[R(s,a,s') + \gamma\, Q(s',a')].$$

This operator is also a contraction and Q^π is its unique fixed point

$$Q^\pi = T^\pi Q^\pi.$$

The optimal action-value function Q^* satisfies the nonlinear Bellman optimality equation

$$Q^*(s,a) = \mathbb{E}_{s'|s,a}[R(s,a,s') + \gamma \max_{a'\in A} Q^*(s',a')].$$

The associated Bellman optimality operator is defined as (with the same slight abuse of notation)

$$T^*\colon Q \in \mathbb{R}^{S\times A} \to T^*Q \in \mathbb{R}^{S\times A}$$
$$[T^*Q](s,a) = \mathbb{E}_{s'|s,a}[R(s,a,s') + \gamma \max_{a'\in A} Q(s',a')].$$

This is still a contraction and Q^* is its unique fixed point

$$Q^* = T^*Q^*.$$

B. Relaxed Assumptions

As mentioned in Section I, an important subtopic of reinforcement learning is to estimate the (action-) value function of a given policy or directly the Q-function of the optimal policy (in all cases the fixed point of a Bellman operator) from samples, i.e., observed trajectories of actual interactions. More precisely, learning is done from a trajectory sampled according to a policy π, which can be seen as a set of transitions

$$\{(s_j, a_j, r_j, s_{j+1}, a_{j+1})_{1\le j\le i}\}. \tag{3}$$

A first problem is that the model (i.e., transition probabilities and reward function) is assumed to be unknown. Therefore, no Bellman operator (from which we search a fixed point for) can be computed. However, they can be estimated from transitions. The sampled Bellman value function evaluation operator $\hat{T}_j^\pi$, to be linked to (1), is defined for a transition (s_j, r_j, s_{j+1}) as

$$\hat{T}_j^\pi\colon V \in \mathbb{R}^S \to \hat{T}_j^\pi V \in \mathbb{R}\colon \hat{T}_j^\pi V = r_j + \gamma\, V(s_{j+1}). \tag{4}$$

Notice that a sampled Bellman optimality operator cannot be defined for the value function [see (2)], as the maximum depends on the expectation. Similarly, a sampled Bellman evaluation operator can be defined for the Q-function. For a transition $(s_j, a_j, r_j, s_{j+1}, a_{j+1})$, it is defined as

$$\hat{T}_j^\pi\colon Q \in \mathbb{R}^{S\times A} \to \hat{T}_j^\pi Q \in \mathbb{R}\colon \hat{T}_j^\pi Q = r_j + \gamma\, Q(s_{j+1}, a_{j+1}).$$

Last but not least, a sampled Bellman optimality operator can be defined for the Q-function (thanks to the fact that the maximum depends on the expectation and not the contrary). For a transition (s_j, a_j, r_j, s_{j+1}), we define

$$\hat{T}_j^*\colon Q \in \mathbb{R}^{S\times A} \to \hat{T}_j^* Q \in \mathbb{R}\colon \hat{T}_j^* Q = r_j + \gamma \max_{a\in A} Q(s_{j+1}, a).$$

A second problem is that the state space is usually too large to allow an exact representation of the (action-) value function. Therefore, an approximate representation should be adopted. This paper focuses on parametric approaches: the estimated value (respectively, action-value) function is of the form $\hat{V}_\theta$ (respectively $\hat{Q}_\theta$), where θ is the parameter vector; this estimate belongs to a hypothesis space

$$\mathcal{H} = \{\hat{V}_\theta \text{ (resp. } \hat{Q}_\theta)|\theta \in \mathbb{R}^p\}$$

which specifies the architecture of the approximation. For example, if the state space is sufficiently small, an exact tabular representation can be chosen for the value function. The estimate is thus of the form $\hat{V}_\theta(s) = e_s^T\theta$, with e_s being a unitary vector which is equal to 1 in the component corresponding to state s and 0 elsewhere. A common choice in RL is to adopt a linearly parameterized value function: in this case, $\hat{V}_\theta(s) = \phi(s)^T\theta$, with $\phi(s)$ a vector of basis functions (e.g., a linear radial basis function network) and θ the set of associated weights. More complex hypothesis spaces can be envisioned, such as neural networks. However, notice that some of the approaches reviewed in this paper do not allow handling nonlinear representations.

C. Survey Overview

Estimating a function from samples is a common topic of supervised learning. However, estimating a value function is a harder problem. Indeed, values are never directly observed, so it is not a regression problem. The available information is provided by the gathered rewards, of which the accumulation defines the value function. Therefore, supervised learning techniques cannot be directly applied to learn such a function.[2] This paper reviews state-of-the-art (parametric) value function estimation algorithms by grouping them into three categories. It will be shown that all approaches minimize the following empirical cost function (given for the Q-function, the value function being a special case):

$$\theta_i = \underset{\omega\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(\hat{T}_j \hat{Q}_\xi - \hat{Q}_\omega(s_j, a_j)\right)^2 \tag{5}$$

where $\hat{T}_j$ denotes either $\hat{T}_j^\pi$ or $\hat{T}_j^*$. Each of the three main approaches is obtained by instantiating the parameter vector $\xi \in \mathbb{R}^p$, as explained below and summarized in Table I.

[2]Actually, this is not quite true. One can perform Monte Carlo rollouts starting from a given set of states to obtain related estimated returns (i.e., estimates of the expected discounted cumulative reward for each state of the set) and use this in any supervised learning scheme. However, this would require being able to sample many possibly infinite trajectories for each state of the set, which is therefore not really practical.

TABLE I
SUMMARY

	Bootstrapping	Residual	Projected Fixed Point Direct/iterated	
Stochastic Gradient descent	TD-VFA [22] TDQ-VFA [23] QL-VFA [24]	R-SGD [9]	(nl)GTD2 ([15]) [14] (nl)TDC ([15]) [14] Greedy-GQ [17]	
(Recursive) Least-squares	FPKF [8]	GPTD [25] KTD [11]	LSTD [12] slLSTD [13]	LSPE [18] Q-OSP [19]
Other				Fitted-Q [21]

1) Bootstrapping Approaches ($\xi = \theta_{j-1}$): Bootstrapping approaches (reviewed in Section III) consist in treating value function approximation as a supervised learning problem and to derive an online algorithm. As values are not directly observable, they are replaced by an estimate computed using a sampled Bellman operator (bootstrapping refers to replacing an unobserved value by an estimate). As first noticed in [8], this corresponds to minimizing the cost function (5) instantiated with $\xi = \theta_{j-1}$. If it is minimized using a stochastic gradient descent, this provides the TD-Q and Q-learning algorithms [2], given that the sampled Bellman evaluation or optimality operator is considered. Using a (linear) recursive least-squares approach, this gives the fixed-point Kalman filter algorithm [8].

2) Residual Approaches ($\xi = \omega$): Residual approaches (reviewed in Section IV) consist in minimizing the squared error between the (action-) value function and its image through a Bellman operator. Practically, a sampled operator is used, which leads to biased estimates. This corresponds to minimizing the cost function (5) instantiated with $\xi = \omega$. If it is minimized using a stochastic gradient descent, it provides the residual algorithms of [9]. With a linear recursive least-squares approach, it is the parametric Gaussian process temporal differences algorithm [10], which can be extended using a statistical linearization approach to nonlinear parameterizations and to the optimality operator (the Kalman temporal differences framework [11]).

3) Projected Fixed-Point Approaches ($\xi = \theta_{i-1}/\theta_i$): The projected fixed-point approaches (reviewed in Section V) minimize the squared error between the (action-) value function and the image of this function under the (sampled) Bellman operator projected onto the hypothesis space. This is illustrated by Fig. 1 on page 854. It can be seen as searching for the fixed point of an operator defined as the composition of the projection with one of the Bellman operators. Computing directly this fixed point corresponds to minimizing the cost function (5) instantiated with $\xi = \theta_i$. Solved using a linear least-squares approach, it gives the least-squares temporal differences algorithm [12] (generalized using a statistical linearization approach in [13]). Solved using a stochastic gradient descent approach, it provides a bunch of algorithms: gradient temporal difference 2 and temporal difference with gradient correction [14], nonlinear extension of these algorithms [15], as well as GQ(λ), which is an extension to off-policy learning and eligibility traces [16], and greedy-GQ, an extension to the Bellman optimality operator [17]. Computing the fixed point associated to the composed operator in an iterative way corresponds to minimizing the cost function (5) instantiated with $\xi = \theta_{i-1}$. If solved using a linear least-squares approach, it is the least-squares policy evaluation algorithm [18], extended to the Bellman optimality operator in [19]. In a batch setting and using any supervised learning algorithm instead of the projection, it is the fitted-Q approach [20], [21].

Sections III–V focus on how one can learn the action-value function from samples. How these samples are generated depends on the way the value function approximator is embedded in a more general control scheme (the ultimate goal of RL being to estimate an optimal policy), which is discussed in Section VI.

III. BOOTSTRAPPING APPROACHES

Bootstrapping approaches deal with (action-) value function approximation as a supervised learning problem. The (action-) value function of interest is assumed to be observed (either the value or action-value function of a given policy π or directly the optimal Q-function), and it is projected onto the hypothesis space (minimizing $\|Q - \hat{Q}_\theta\|^2$) using a stochastic gradient descent or a recursive least-squares approach.

However, the resulting algorithms make use of a value that is actually not observed. Bootstrapping consists of replacing this missing observation by a pseudo-observation computed by applying a sampled Bellman operator to the current estimate of the (action-) value function. This is an easy way to understand how to derive algorithms. However, bootstrapping is not supervised learning; the actual minimized cost function is provided in Section III-C.

A. Bootstrapped Stochastic Gradient Descent

Algorithms presented in this section aim at estimating, respectively, the value function of a given policy (TD), the Q-function of a given policy (TD-Q), or directly the optimal action-value function (Q-learning) by combining the bootstrapping principle with a stochastic gradient descent over the associated empirical cost function [2].

1) TD With Function Approximation: TD with function approximation (TD-VFA) aims at estimating the value function V^π of a fixed policy π. Let the notation v_j^π depict a (possibly noisy) observation of $V^\pi(s_j)$. The empirical cost is

$$\hat{J}_{V^\pi}(\theta) = \sum_j \left(v_j^\pi - \hat{V}_\theta(s_j) \right)^2 . \quad (6)$$

More precisely, TD with function approximation minimizes this empirical cost function using a stochastic gradient descent: parameters are adjusted by an amount proportional to an approximation of the gradient of (6), only evaluated on a single training example. Let α_i be a learning rate satisfying the classical stochastic approximation criterion: $\sum_{i=1}^{\infty} \alpha_i = \infty$ and $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$. Parameters are updated according to the following Widrow–Hoff equation, given the ith observed state s_i

$$\begin{aligned}\theta_i &= \theta_{i-1} - \frac{\alpha_i}{2}\nabla_{\theta_{i-1}}\left(v_i^{\pi} - \hat{V}_{\theta}(s_i)\right)^2 \\ &= \theta_{i-1} + \alpha_i\left(\nabla_{\theta_{i-1}}\hat{V}_{\theta}(s_i)\right)\left(v_i^{\pi} - \hat{V}_{\theta_{i-1}}(s_i)\right).\end{aligned}$$

However, as mentioned above, the value of the state s_i is not observed. It is where the bootstrapping principle applies. The unobserved value v_i^{π} is replaced by an estimate computed by applying the sampled Bellman evaluation operator (4) to the current estimate $\hat{V}_{\theta_{i-1}}(s_i)$. Assume that not only the current state is observed, but also the whole transition (s_i, s_{i+1}) (sampled according to the policy π) as well as the associated reward r_i. The corresponding update rule is, therefore

$$\theta_i = \theta_{i-1} + \alpha_i\left(\nabla_{\theta_{i-1}}\hat{V}_{\theta}(s_i)\right)\left(\hat{T}_i^{\pi}\,\hat{V}_{\theta_{i-1}} - \hat{V}_{\theta_{i-1}}(s_i)\right)$$

with

$$\hat{T}_i^{\pi}\,\hat{V}_{\theta_{i-1}} = r_i + \gamma\,\hat{V}_{\theta_{i-1}}(s_{i+1}).$$

The idea behind using this sampled operator (and more generally behind bootstrapping) is twofold: if parameters are perfectly estimated and if the value function belongs to the hypothesis space, this provides an unbiased estimate of the actual value (the value function being the fixed point of the unsampled operator) and this estimate provides more information as it is computed using the observed reward. Under some assumptions, notably a linear parameterization hypothesis, TD with function approximation can be shown to be convergent [26] (and it converges to the same solution as LSTD, described in Section V-A). This is no longer the case when it is combined with a nonlinear function approximator[3] (a counterexample is exhibited in [26]).

2) TD-Q With Function Approximation: TD-Q with function approximation (TDQ-VFA) aims at estimating the Q-function of a fixed policy π. Combined with a control component such as an ϵ-greedy or a Gibbs policy, it provides the well-known SARSA algorithm (we use the name TD-Q to separate clearly policy evaluation from control, see Section VI). Notice that an MDP with a fixed policy defines a valued Markov chain over the state-action space; therefore value and Q-function evaluation are two similar problems. Let q_j^{π} be a (possibly noisy) observation of $Q^{\pi}(s_j)$. The considered algorithm aims at minimizing

$$\hat{J}_{Q^{\pi}}(\theta) = \sum_j \left(q_j^{\pi} - \hat{Q}_{\theta}(s_j, a_j)\right)^2.$$

TD-Q with function approximation also minimizes the related empirical cost function using a stochastic gradient descent. Parameters are thus updated as follows, given the ith state-action pair (s_i, a_i):

$$\begin{aligned}\theta_i &= \theta_{i-1} - \frac{\alpha_i}{2}\nabla_{\theta_{i-1}}\left(q_i^{\pi} - \hat{Q}_{\theta}(s_i, a_i)\right)^2 \\ &= \theta_{i-1} + \alpha_i\left(\nabla_{\theta_{i-1}}\hat{Q}_{\theta}(s_i, a_i)\right)\left(q_i^{\pi} - \hat{Q}_{\theta_{i-1}}(s_i, a_i)\right).\end{aligned}$$

As before, q_i^{π} is not observed and it is replaced by an estimate computed by applying the sampled Bellman evaluation operator to the current estimate $\hat{Q}_{\theta_{i-1}}(s_i, a_i)$. Assume that the whole transition $(s_i, a_i, s_{i+1}, a_{i+1})$, a_{i+1} being sampled according to policy π, as well as the associated reward r_i, is observed. Parameters are therefore updated according to

$$\theta_i = \theta_{i-1} + \alpha_i\left(\nabla_{\theta_{i-1}}\hat{Q}_{\theta}(s_i, a_i)\right)\left(\hat{T}_i^{\pi}\,\hat{Q}_{\theta_{i-1}} - \hat{Q}_{\theta_{i-1}}(s_i, a_i)\right)$$

with

$$\hat{T}_i^{\pi}\,\hat{Q}_{\theta_{i-1}} = r_i + \gamma\,\hat{Q}_{\theta_{i-1}}(s_{i+1}, a_{i+1}).$$

From a practical point of view, using the Q-function instead of the value function is of interest because it does not require the model to be known in order to derive a greedy policy. Convergence results holding for TD with function approximation apply rather directly to TD-Q with function approximation.

Introduced as above, TD-Q estimates the action-value function of the policy π followed to sample the trajectory (3) used to feed the algorithm (on-policy learning). However, one may want to estimate the Q-function of some different target policy π_t (off-policy learning). This can be simply done by using the same transitions and replacing $\hat{T}_i^{\pi}$ by $\hat{T}_i^{\pi_t}$. This means that the transition $(s_i, a_i, s_{i+1}, a_{i+1})$ should be replaced by the transition $(s_i, a_i, s_{i+1}, a_{i+1}^t)$, with a_{i+1}^t being sampled according to $\pi_t(.|s_{t+1})$. Also, one can consider an alternative bootstrap for $q_i^{\pi_t}$ (with $\pi = \pi_t$ in the on-policy case). Instead of using $\hat{T}_i^{\pi_t}\,\hat{Q}_{\theta_{i-1}} = r_i + \gamma\,\hat{Q}_{\theta_{i-1}}(s_{i+1}, a_{i+1}^t)$, one can replace $q_i^{\pi_t}$ by

$$r_i + \gamma\,\mathbb{E}_{a|\pi_t, s_{i+1}}[\hat{Q}_{\theta_{i-1}}(s_{i+1}, a)].$$

This is also an unbiased estimate of $[T^{\pi_t}\hat{Q}_{\theta_{i-1}}](s_i, a_i)$, which can be used whenever π_t is known and which may lead to better estimates [28]. These ideas (off-policy and alternative bootstrap) are quite general and can be applied to any policy evaluation algorithm presented in this survey (to any algorithm estimating a Q-function and derived from the Bellman evaluation operator). However, notice that, if there is a too large mismatch between the stationary distributions induced by the policies π and π_t, the Q-function will be badly estimated by the off-policy approach. For convergence properties of off-policy learning, see [29] and references therein.

3) Q-Learning With Function Approximation: Q-learning with function approximation (QL-VFA) aims at estimating directly the optimal action-value function Q^*. Let q_j^* be a (possibly noisy) observation of $Q^*(s_j)$. This algorithm aims at minimizing the empirical cost function

$$\hat{J}_{Q^*}(\theta) = \sum_j \left(q_j^* - \hat{Q}_{\theta}(s_j, a_j)\right)^2.$$

The same approach is used, and parameters are recursively estimated using a stochastic gradient descent. Given the ith

Despite this, one of the important empirical successes of RL is based on TD-Q with linear function approximation [27].

state-action pair (s_i, a_i), parameters should be updated according to

$$\theta_i = \theta_{i-1} - \frac{\alpha_i}{2}\nabla_{\theta_{i-1}}\left(q_i^* - \hat{Q}_\theta(s_i,a_i)\right)^2 = \theta_{i-1} + \alpha_i\left(\nabla_{\theta_{i-1}}\hat{Q}_\theta(s_i,a_i)\right)\left(q_i^* - \hat{Q}_{\theta_{i-1}}(s_i,a_i)\right).$$

As for the preceding algorithms, the bootstrapping principle is applied to estimate the unobserved q_i^* value, using the sampled Bellman optimality operator now, which assumes that the transition (s_i, a_i, s_{i+1}) as well as associated reward r_i is observed. Notice that Q-learning with function approximation is an off-policy algorithm, as it also evaluates a policy (the optimal one in this case) from samples generated according to a different policy. Practically, transitions can be sampled according to any sufficiently explorative policy. Parameters are updated as

$$\theta_i = \theta_{i-1} + \alpha_i\left(\nabla_{\theta_{i-1}}\hat{Q}_\theta(s_i,a_i)\right)\left(\hat{T}_i^*\hat{Q}_{\theta_{i-1}} - \hat{Q}_{\theta_{i-1}}(s_i,a_i)\right)$$

with

$$\hat{T}_i^*\hat{Q}_{\theta_{i-1}} = r_i + \gamma\max_{a\in A}\hat{Q}_{\theta_{i-1}}(s_{i+1},a).$$

Under some assumptions, notably a linear parameterization and the fact that the sampling policy π is explorative enough (it should visit all state-action pairs), QL-VFA can be shown to be convergent [30].

4) Summary View: These algorithms can be formalized using the same unified notation. First, value and action-value function evaluation (TD and TD-Q with function approximation) are somehow redundant. As $V^\pi(s) = \mathbb{E}_{a|\pi,s}[Q^\pi(s,a)]$, any algorithm aiming at estimating a Q-function can easily be specialized to the related value function, as long as the policy is known (thus, this does not apply to Q-learning with function approximation). Practically, it consists of replacing the Q-function by the value function and state–action pairs by states. Consequently, value function estimation is not considered anymore in this paper. Let $\hat{T}_j$ denote either the sampled evaluation or optimality operator, depending on the context. Let also q_j be either q_j^π or q_j^*, also depending on the context. TD-Q and Q-learning with function approximation aim at minimizing the following empirical cost function, which is instantiated by specifying if evaluation or direct optimization is considered

$$\hat{J}(\theta) = \sum_j\left(q_j - \hat{Q}_\theta(s_j,a_j)\right)^2.$$

As before, parameters are estimated using a stochastic gradient descent and by applying the bootstrapping principle, which leads to the following update:

$$\theta_i = \theta_{i-1} + \alpha_i\left(\nabla_{\theta_{i-1}}\hat{Q}_\theta(s_i,a_i)\right)\left(\hat{T}_i\hat{Q}_{\theta_{i-1}} - \hat{Q}_{\theta_{i-1}}(s_i,a_i)\right).$$

A practical algorithm is instantiated by specifying which of the sampled Bellman operator is used for $\hat{T}$, i.e., $\hat{T}^\pi$ or $\hat{T}^*$. Algorithms have been detailed so far for the sake of clarity. However, this summarizing point of view is adopted in the rest of this paper. Notice also that this update is actually a Widrow–Hoff equation of the following form:

$$\theta_i = \theta_{i-1} + K_i\delta_i. \tag{7}$$

In this expression, $\delta_i = \hat{T}_i\hat{Q}_{\theta_{i-1}} - \hat{Q}_{\theta_{i-1}}(s_i,a_i)$ is the so-called TD error, which is the reward prediction error given the current estimate of the action-value function (it depends on which Bellman operator is considered) and $K_i = \alpha_i\nabla_{\theta_{i-1}}\hat{Q}_\theta(s_i,a_i)$ is a gain indicating in which direction the parameter vector should be corrected in order to improve the estimate. Most of (online) algorithms presented in this paper satisfy a Widrow–Hoff update.

B. Bootstrapped Recursive Least-Squares

The fixed-point Kalman filter (FPKF) [8] also seeks at minimizing the empirical cost function linking (actually unobserved) action-values to the estimated Q-function (still with a bootstrapping approach)

$$\hat{J}_i(\theta) = \sum_{j=1}^{i}\left(q_j - \hat{Q}_\theta(s_j,a_j)\right)^2.$$

However, the parameterization is assumed to be linear and a (recursive) least-squares approach is adopted instead of the stochastic gradient descent used for preceding algorithms. The considered hypothesis space is of the form

$$\mathcal{H} = \{\hat{Q}_\theta\colon (s,a)\in S\times A\to\phi(s,a)^T\theta\in\mathbb{R}|\theta\in\mathbb{R}^p\}$$

where $\phi(s,a)$ is a feature vector (to be chosen before hand). For a given state–action couple (s_j,a_j), $\phi(s_j,a_j)$ is shortened as ϕ_j. The corresponding empirical objective function can thus be rewritten as

$$\hat{J}_i(\theta) = \sum_{j=1}^{i}\left(q_j - \phi_j^T\theta\right)^2.$$

Thanks to linearity in parameters, this cost function is convex and has a unique minimum

$$\theta_i = \operatorname*{argmin}_{\theta\in\mathbb{R}^p}\hat{J}_i(\theta).$$

This optimization problem can be solved analytically by zeroing the gradient of $\hat{J}_i(\theta)$; this is the principle of the least-squares method. Parameters are thus estimated as

$$\theta_i = \left(\sum_{j=1}^{i}\phi_j\phi_j^T\right)^{-1}\sum_{j=1}^{i}\phi_j q_j.$$

Let us write $P_i^{-1} = \sum_{j=1}^{i}\phi_j\phi_j^T$. The Sherman–Morrison formula allows updating directly the inverse of a rank-1 perturbed matrix

$$P_i = P_{i-1} - \frac{P_{i-1}\phi_i\phi_i^T P_{i-1}}{1+\phi_i^T P_{i-1}\phi_i}.$$

This allows estimating parameters recursively

$$\theta_i = \theta_{i-1} + \frac{P_{i-1}\phi_i}{1+\phi_i^T P_{i-1}\phi_i}\left(q_i - \hat{Q}_{\theta_{i-1}}(s_i,a_i)\right).$$

As for algorithms presented in Section V-B2, the bootstrapping principle is applied and the unobserved q_i action-value is replaced by the estimate $\hat{T}_i\hat{Q}_{\theta_{i-1}}$

$$\theta_i = \theta_{i-1} + \frac{P_{i-1}\phi_i}{1+\phi_i^T P_{i-1}\phi_i}\left(\hat{T}_i\hat{Q}_{\theta_{i-1}} - \hat{Q}_{\theta_{i-1}}(s_i,a_i)\right).$$

This equation is actually a Widrow–Hoff update (7). The temporal difference error is still $\delta_i = \hat{T}_i \hat{Q}_{\theta_{i-1}} - \hat{Q}_{\theta_{i-1}}(s_i, a_i)$; this prediction error term is actually common to all algorithms aiming at estimating the action-value function.The gain depends on the fact that a least-squares minimization has been considered

$$K_i = \frac{P_{i-1}\phi_i}{1 + \phi_i^T P_{i-1}.\phi_i}.$$

This gain can be compared to the one obtained using a stochastic gradient descent approach, presented in Section III-A. With a linear parameterization, the corresponding gain is $K_i = \alpha_i \phi_i$. Therefore, FPKF can be seen as a variation of TD-Q or Q-learning for which there is some automatic coordinate-based step-size adaptation.

If the sampled Bellman evaluation operator is considered, the FPKF update rule specializes as

$$\theta_i = \theta_{i-1} + K_i \left(r_i + \gamma \phi_{i+1}^T \theta_{i-1} - \phi_i^T \theta_{i-1} \right).$$

If the sampled Bellman optimality operator is considered, this update rule specializes as

$$\theta_i = \theta_{i-1} + K_i \left(r_i + \gamma \max_{a \in A} \left(\phi(s_{i+1}, a)^T \theta_{i-1} \right) - \phi_i^T \theta_{i-1} \right).$$

This algorithm can be shown to be convergent under some assumptions, for both sampled operators (evaluation and optimality in the case of optimal stopping problems). See [8] for details.

C. Summary

The principle of bootstrapping algorithms is to derive an online algorithm that minimizes a supervised-learning-based cost function

$$\theta_i = \underset{\omega \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} (q_j - \hat{Q}_\omega(s_j, a_j))^2.$$

As values are not observed, the bootstrapping principle is applied: the unobserved q_j value is replaced by some estimate. Given that we are looking for a fixed point of one of the Bellman operators, this estimate is provided by $\hat{T}_j \hat{Q}_{\theta_{j-1}}$. Therefore, bootstrapping approaches solve the following optimization problem (as first noticed in [8]):

$$\theta_i = \underset{\omega \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(\hat{T}_j \hat{Q}_{\theta_{j-1}} - \hat{Q}_\omega(s_j, a_j) \right)^2.$$

This corresponds to (5) instantiated with $\xi = \theta_{j-1}$. Practical algorithms are then derived, given that this cost function is minimized using a stochastic gradient descent or a recursive least-squares approach, and given what Bellman operator is considered. This is summarized in the first column of Table I.

IV. Residual Approaches

Residual approaches aim at finding an approximation of the fixed point of the Bellman operator by minimizing the distance between the (action-) value function and its image through one of the Bellman operators. The associated cost function is

$$J(\theta) = \| \hat{Q}_\theta - T \hat{Q}_\theta \|^2.$$

Practically, learning is done using samples, and the Bellman operator is replaced by a sampled Bellman operator, the model (particularly transition probabilities) being unknown. The associated empirical cost function is, therefore

$$\hat{J}(\theta) = \sum_j \left(\hat{Q}_\theta(s_j, a_j) - \hat{T}_j \hat{Q}_\theta \right)^2. \tag{8}$$

A common drawback of all approaches aiming at minimizing this cost function is that they produce biased estimates of the (action-) value function. Basically, this is due to the fact that the expectation of a square is not the square of the expectation

$$\begin{aligned} &\mathbb{E}_{s_{j+1}|s_j,a_j,\pi}[(Q(s_j, a_j) - \hat{T}_j Q)^2] \\ &= (Q(s_j, a_j) - (TQ)(s_j, a_j))^2 + \mathrm{Var}_{s_{j+1}|s_j,a_j,\pi}(\hat{T}_j Q). \end{aligned}$$

There is an unwanted variance term acting as a penalty factor which favors smooth functions. Though such penalties are commonly used for regularization, this one is harmful here as it cannot be controlled [31]. All methods presented below can be modified so as to handle this problem; this will be shortly discussed. However, it is important to notice that any algorithm aiming at minimizing this cost presents this bias problem.

A. Residual Stochastic Gradient Descent

The so-called residual algorithms (R-SGD for residual stochastic gradient descent) have been introduced in [9]. Their principle is to minimize the empirical cost function (8) using a stochastic gradient descent. The corresponding update rule is, therefore

$$\theta_i = \theta_{i-1} - \frac{\alpha_i}{2} \nabla_{\theta_{i-1}} \left((\hat{Q}_\theta(s_i, a_i) - \hat{T}_i \hat{Q}_\theta)^2 \right)$$

with

$$\begin{aligned} &\nabla_{\theta_{i-1}} \left((\hat{Q}_\theta(s_i, a_i) - \hat{T}_i \hat{Q}_\theta)^2 \right) \\ &= \left(\nabla_{\theta_{i-1}} \left(\hat{Q}_\theta(s_i, a_i) - \hat{T}_i \hat{Q}_\theta \right) \right) \left(\hat{T}_i \hat{Q}_{\theta_{i-1}} - \hat{Q}_{\theta_{i-1}}(s_i, a_i) \right). \end{aligned}$$

Here again, this update is actually a Widrow–Hoff equation [see (7)] with $\delta_i = \hat{T}_i \hat{Q}_{\theta_{i-1}} - \hat{Q}_{\theta_{i-1}}(s_i, a_i)$ and $K_i = \alpha_i \nabla_{\theta_{i-1}} (\hat{Q}_\theta(s_i, a_i) - \hat{T}_i \hat{Q}_\theta)$. If the sampled Bellman evaluation operator is considered, the gain and temporal difference error are given by

$$\begin{aligned} K_i &= \alpha_i \nabla_{\theta_{i-1}} \left(\hat{Q}_\theta(s_i, a_i) - \gamma \hat{Q}_\theta(s_{i+1}, a_{i+1}) \right) \\ \delta_i &= r_i + \gamma \hat{Q}_{\theta_{i-1}}(s_{i+1}, a_{i+1}) - \hat{Q}_{\theta_{i-1}}(s_i, a_i). \end{aligned}$$

A first problem arises when the sampled Bellman optimality operator is considered. In this case, we have

$$\begin{aligned} K_i &= \alpha_i \nabla_{\theta_{i-1}} \left(\hat{Q}_\theta(s_i, a_i) - \gamma \max_{a \in A} \hat{Q}_\theta(s_{i+1}, a) \right) \\ \delta_i &= r_i + \gamma \max_{a \in A} \hat{Q}_{\theta_{i-1}}(s_{i+1}, a) - \hat{Q}_{\theta_{i-1}}(s_i, a_i). \end{aligned}$$

Here, the gradient of the max operator must be computed, i.e., $\nabla_{\theta_{i-1}} (\max_{a \in A} \hat{Q}_\theta(s_{i+1}, a))$. This is not straightforward;

if this gain is introduced in [9], no solution is provided to compute it.[4] Another problem is that these algorithms compute biased estimates of the (action-) value function, as explained above. This is inherent to all approaches that minimize a residual cost function using a sampled Bellman operator. In order to handle this problem, it is proposed in [9] to use a double sampling scheme. Let us consider the Bellman evaluation operator. Two transitions are independently generated from the state-action couple (s_i, a_i): $(s_i, a_i, r'_i, s'_{i+1}, a'_{i+1})$ and $(s_i, a_i, r''_i, s''_{i+1}, a''_{i+1})$. One of these transitions is used to compute the gain, and the other one to compute the TD error

$$K_i = \alpha_i \nabla_{\theta_{i-1}} \left(\hat{Q}_\theta(s_i, a_i) - \gamma \hat{Q}_\theta(s'_{i+1}, a'_{i+1}) \right)$$
$$\delta_i = r_i + \gamma \hat{Q}_{\theta_{i-1}}(s''_{i+1}, a''_{i+1}) - \hat{Q}_{\theta_{i-1}}(s_i, a_i).$$

These two transitions being sampled independently, taking the expectation of $K_i \delta_i$ leads to the use of the true (i.e., unsampled) Bellman operator, without the variance term contrary to the use of the same transition in both gain and TD error. However, this suggests that transitions can be sampled on demand (e.g., using a simulator), which might be a strong assumption.

B. Residual Least-Squares

In this section, methods based on a least-squares minimization of cost function (8) are reviewed. The parametric Gaussian process temporal differences [10] algorithm minimizes it by assuming a linear parameterization as well as the Bellman evaluation operator, and the Kalman temporal differences framework [11] generalizes it to nonlinear parameterizations as well as to the Bellman optimality operator thanks to a statistical linearization approach [33].

1) Gaussian Process Temporal Differences: Assuming a linear parameterization and the Bellman evaluation operator, the cost function (8) is linear and can be rewritten as

$$J_i(\theta) = \sum_{j=1}^{i} (r_j + \gamma \phi_{j+1}^T \theta - \phi_j^T \theta)^2. \tag{9}$$

Thanks to the linearity in parameters, it can be solved using a recursive least-squares approach: zeroing the gradient ($\nabla_\theta J_i(\theta) = 0$) provides a batch estimate that can be made recursive by using the Sherman–Morrison formula. To any least-squares problem can be linked a so-called observation model, linking outputs to inputs through the hypothesized parametric model plus some white observation noise n_j. In this case, it is simply the sampled Bellman evaluation equation

$$r_j = \phi_j^T \theta - \gamma \phi_{j+1}^T \theta + n_j.$$

With a unitary noise (i.e., n_j is of unitary variance, $P_{n_j} = 1$), the corresponding least-squares problem is provided by (9).

The (parametric) Gaussian process temporal differences (GPTD) algorithm [10] considers the same observation model, with a not necessarily unitary noise n_j. The effect of the noise variance is to weight the square terms in the minimized cost function: this is the slight difference with cost (9)

$$J_i(\theta) = \sum_{j=1}^{i} \frac{1}{P_{n_j}} \left(r_j + \gamma \phi_{j+1}^T \theta - \phi_j^T \theta \right)^2.$$

If one has some prior knowledge about the noise n_i, this can be used to obtain this simple weighted least-squares problem. If not, it is sufficient to consider a constant variance (therefore weighting no longer affects the solution of this optimization problem).

Let us write $\Delta\phi_j = \phi_j - \gamma \phi_{j+1}$. The unique parameter vector minimizing the above convex cost function can be computed analytically by zeroing the gradient with respect to the parameter vector

$$\theta_i = \underset{\theta \in \mathbb{R}^p}{\operatorname{argmin}} J_i(\theta) = \left(\sum_{j=1}^{i} \frac{1}{P_{n_j}} \Delta\phi_j \Delta\phi_j^T \right)^{-1} \sum_{j=1}^{i} \frac{1}{P_{n_j}} \Delta\phi_j r_j.$$

Let us write $P_i = (\sum_{j=1}^{i} 1/P_{n_j} \Delta\phi_j \Delta\phi_j^T)^{-1}$. Thanks to the Sherman–Morrison formula, P_i can be computed iteratively and the parameters estimated recursively. Let θ_0 and P_0 be some priors; the (parametric) GPTD algorithm is given by

$$\theta_i = \theta_{i-1} + \frac{P_{i-1} \Delta\phi_i}{P_{n_i} + \Delta\phi_i^T P_{i-1} \Delta\phi_i} \left(r_i - \Delta\phi_i^T \theta_{i-1} \right)$$
$$P_i = P_{i-1} - \frac{P_{i-1} \Delta\phi_i \Delta\phi_i^T P_{i-1}}{P_{n_i} + \Delta\phi_i^T P_{i-1} \Delta\phi_i}.$$

One can recognize the temporal difference error $\delta_i = r_i - \Delta\phi_i^T \theta_{i-1}$ and a gain

$$K_i = P_{i-1} \Delta\phi_i / (P_{n_i} + \Delta\phi_i^T P_{i-1} \Delta\phi_i),$$

to be linked again with the generic Widrow–Hoff update (7). Notice that P_i is actually a variance matrix quantifying the uncertainty over current parameters estimation (it is the variance of the parameter vector conditioned on past i observed rewards). This is not clear from the proposed least-squares-based derivation; however, it is direct by adopting a Bayesian perspective [25]. Notice that this interpretation of P_i as being a variance matrix can be useful for handling the dilemma between exploration and exploitation [10].

As all other residual methods, GPTD produces biased estimates of the value function when transitions are stochastic. To alleviate this problem, a colored observation noise n_j can be used instead of the classical white noise assumption (a noise being white if $\forall i \neq j$, n_i, and n_j are independent, and a colored noise is any non-white noise) [34]. This noise allows removing the bias (because it leads to minimizing another cost function, linking state estimates to Monte Carlo samples of the discounted return [34], [10, Ch. 4.4.3]), but it also induces a memory effect which prevents from learning in an off-policy manner, much like eligibility traces do (see [35]). These developments are not pursued here (they rely mainly on Bayesian inference and eligibility traces, just the latter topic being briefly addressed in Section VIII).

The GPTD framework was originally introduced in a different nonparametric manner [25]: the value function is modeled

[4] Indeed, $\max_{\hat{Q}_\theta(s_{i+1}, a_{i+1})} \rho(s_i, a_i, r_i, s_{i+1}, a_{i+1}, \theta)$ is generally nondifferentiable with respect to θ; a gradient descent algorithm can be designed using subdifferentials [32].

as a Gaussian process, and a kernel-based online sparsification scheme is used to obtain a practical algorithm. As we focus on parametric value function approximation, the algorithm we review is the parametric GPTD [10, Sec. 4.3]. There also exists an LSTD-based variation of GPTD (see Section V-A for LSTD), derived from a maximum likelihood interpretation of the LSTD algorithm [10, Sec. 4.5]. In the parametric case, it is actually an ℓ_2-regularized form of LSTD, so it is not further developed here.

2) Kalman Temporal Differences: As the GPTD framework, the Kalman temporal differences framework [11] also seeks at minimizing the cost function (8), not necessarily considering a unitary noise variance too. Let us write $\hat{T}_i Q = r_i + \gamma \hat{P}_i Q$ with

$$\hat{P}_i Q = \begin{cases} Q(s_{i+1}, a_{i+1}) \text{ (if sampled evaluation op)} \\ \max_{a \in A} Q(s_{i+1}, a) \text{ (if sampled optimality op).} \end{cases}$$

The estimated parameters should minimize

$$\hat{J}_i(\theta) = \sum_{j=1}^{i} \frac{1}{P_{n_j}} \left(r_j - \left(\hat{Q}_\theta(s_j, a_j) - \gamma \hat{P}_j \hat{Q}_\theta \right) \right)^2. \quad (10)$$

Contrary to GPTD, KTD does not assume either a linear parameterization or the sampled evaluation operator. Instead, it makes use of a derivative-free linearization scheme (the derivative-free aspect allows considering the sampled optimality operator), the so-called statistical linearization [33]. The basic idea is to linearize this cost function (taking the distribution of parameters into account), which can subsequently be solved using a classical recursive least-squares approach.

Linearization of (10) goes through the linearization of the associated observation model (to be considered as a function of θ, the state-action couple being fixed)

$$r_j = \hat{Q}_\theta(s_j, a_j) - \gamma \hat{P}_j \hat{Q}_\theta + n_j. \quad (11)$$

Assume that it is evaluated in n sampled parameter vectors $\theta^{(k)}$ of associated weights w_k (how to sample them practically and efficiently being addressed later)

$$\left(\theta^{(k)}, r_j^{(k)} = \hat{Q}_{\theta^{(k)}}(s_j, a_j) - \gamma \hat{P}_j \hat{Q}_{\theta^{(k)}} \right)_{1 \le k \le n}.$$

The following statistics of interest are defined:

$$\bar{\theta} = \sum_{k=1}^{n} w_k \theta^{(k)}, \quad \bar{r}_j = \sum_{k=1}^{n} w_k r_j^{(k)} \quad (12)$$

$$P_\theta = \sum_{k=1}^{n} w_k (\theta^{(k)} - \bar{\theta})(\theta^{(k)} - \bar{\theta})^T$$

$$P_{\theta r_j} = \sum_{k=1}^{n} w_k (\theta^{(k)} - \bar{\theta})(r_j^{(k)} - \bar{r}_j)^T = P_{r_j \theta}^T$$

$$P_{r_j} = \sum_{k=1}^{n} w_k (r_j^{(k)} - \bar{r}_j)^2. \quad (13)$$

Statistical linearization consists of linearizing the nonlinear observation model (11) around $\bar{\theta}$ (with P_θ being actually the [illegible] point of view. It [illegible] $r_j = A_j \theta + b_j + u_j$, u_j being a noise, by minimizing the sum of square errors between values of nonlinear and linearized functions in the regression points

$$(A_j, b_j) = \underset{A,b}{\operatorname{argmin}} \sum_{k=1}^{n} \left(e_j^{(k)} \right)^2 \text{ with } e_j^{(k)} = r_j^{(k)} - \left(A\theta^{(k)} + b \right).$$

The solution of this optimization problem is given by

$$A_j = P_{r_j \theta} P_\theta^{-1} \text{ and } b_j = \bar{r}_j - A_j \bar{\theta}.$$

Moreover, it is easy to check that the covariance matrix of the error is given by

$$P_{e_j} = \sum_{k=1}^{n} \left(e_j^{(k)} \right)^2 = P_{r_j} - A_j P_\theta A_j^T.$$

The nonlinear observation model (11) can thus be replaced by the following equivalent linear observation model:

$$r_j = A_j \theta + b_j + u_j \text{ with } u_j = e_j + n_j.$$

Notice that the linearization error is taken into account through the noise e_j. Noises e_j and n_j being independent, the variance of P_{u_j} is given by $P_{u_j} = P_{e_j} + P_{n_j}$. Given this statistically linearized observation model, the least-squares problem can be rewritten in a linear form, as

$$\theta_i = \underset{\theta \in \mathbb{R}^p}{\operatorname{argmin}} \left(\sum_{j=1}^{i} \frac{1}{P_{u_j}} \left(r_j - A_j \theta - b_j \right)^2 \right)$$
$$= \left(\sum_{j=1}^{i} \frac{1}{P_{u_j}} A_j^T A_j \right)^{-1} \sum_{j=1}^{i} A_j \left(r_j - b_j \right).$$

With this cost function, the higher the statistical linearization error of a given transition (quantified by P_{u_j} through P_{e_j}), the less the corresponding square term contributes to the cost. Using the Sherman–Morrison formula, a recursive update of this estimate can be obtained (expressing the gain K_i directly as a function of the statistics of interest defining A_i and b_i and assuming some priors θ_0 and P_{θ_0}), as

$$K_i = \frac{P_{\theta_i r_i}}{P_{u_i} + P_{r_i}}$$
$$\theta_i = \theta_{i-1} + K_i (r_i - \bar{r}_i)$$
$$P_{\theta_i} = P_{\theta_{i-1}} - K_i \left(P_{v_i} + P_{r_i} \right) K_i^T. \quad (14)$$

Here again, the gains magnitude is directly linked to the linearization error, and large errors will result in small updates (this can be seen as an automatic learning rate). Equation (14) corresponds also to a Widrow–Hoff update, with $r_i - \bar{r}_i$ being a statistically linearized temporal difference error ($\bar{r}_i$ is a prediction of the reward, given the currently estimated Q-function and the experienced transitions; the algorithm does not optimize a myopic TD error).

How to actually sample parameter vectors in order to perform statistical linearization is addressed now. With this recursive estimation, θ_{i-1} (the previous estimate) and P_{i-1} (the associated uncertainty matrix, as explained in Section IV-B1) are known, and the issue is to compute A_i and b_i. A first thing is to choose the point around which to linearize and with which magnitude. It is legitimate to

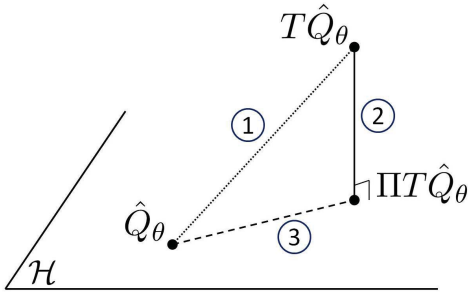

Fig. 1. Projected fixed-point principle.

sample around the previous estimate θ_{i-1} and with a spread related to the uncertainty of these estimates. In other words, n parameter vectors are sampled such that $\bar{\theta}_i = \theta_{i-1}$ and $P_{\theta_i} = P_{i-1}$. Notice that $\bar{\theta}_i$, the point around which linearization is performed in order to update parameters, is different from θ_i, the updated parameter vector. There remains the choice of how parameter vectors are sampled. A natural idea would be to assume a Gaussian distribution of mean θ_{i-1} and variance P_{i-1} and to compute statistics of interest [(12),(13)] using a Monte Carlo approach. However, this would be particularly inefficient. Actually, the problem of sampling these points can be stated as follows: how to sample a random variable (here the parameter vector) of known mean and variance (here θ_{i-1} and P_{i-1}) in order to compute accurate estimates of first- and second-order moments of a nonlinear mapping of this random variable (here $\hat{Q}_\theta(s_i, a_i) - \gamma \hat{P}_i \hat{Q}_\theta$). The unscented transform [36] provides a solution to this problem and is used by the KTD framework[5] Full details are provided in [11].

KTD generalizes GPTD in the sense that it allows handling nonlinearities: nonlinear parameterization thanks to the linearization, and the (sampled) Bellman optimality operator thanks to the fact that this linearization scheme does not rely on a gradient computation. Notice that KTD reduces to parametric GPTD if a linear parameterization as well as the sampled Bellman evaluation operator are considered. As other residual approaches, KTD suffers from the bias problem when system transitions are stochastic. In order to handle this issue, a colored noise model (based on the idea of eligibility traces) can be used [40] (which is actually a generalization of the noise proposed in [34] for GPTD). However, as mentioned in Section IV-B1, this induces some memory effects which prevent learning in an off-policy manner. Consequently, the sampled Bellman optimality operator can no longer be considered in this setting, because of its off-policy aspect. Using a colored noise also leads to minimizing a different cost function. As for GPTD, these developments are not pursued here, see the corresponding papers [34], [35], [40]. Notice that the available uncertainty information (matrix P_i) can be useful for the dilemma between exploration and exploitation [41], [42].

[5]Notice that other approximation schemes can be considered instead of the unscented transform, such as the scaled unscented transform [37], approximation schemes based on Sterling interpolation [38], or, more generally, sigma-point approximations [39].

C. Summary

Residual approaches aim at finding an approximation of the fixed point of one of the Bellman operators by minimizing the distance between the (action-) value function and its image through one of the Bellman operators. The associated theoretical cost function is $J(\theta) = \|\hat{Q}_\theta - T\hat{Q}_\theta\|^2$.

Practically, learning is done using samples, and the Bellman operator is replaced by its sampled counterpart, the model being unknown. The associated empirical cost function is therefore

$$\theta_i = \underset{\omega \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(\hat{T}_j \hat{Q}_\omega - \hat{Q}_\omega(s_j, a_j)\right)^2.$$

This is exactly (5) instantiated with $\xi = \omega$. Minimizing this cost function by considering one of the Bellman operators and using either a stochastic gradient descent or a recursive least-squares approach (possibly generalized using a statistical linearization) provides one of the presented algorithms (see the second column of Table I).

A common drawback of all approaches aiming at minimizing this cost function is that they produce biased estimates of the (action-) value function: minimizing the empirical cost function (8) does not lead to minimizing the theoretical cost function $\|\hat{Q}_\theta - T\hat{Q}_\theta\|^2$ asymptotically. All methods presented in Section IV can be modified so as to handle the problem: a double sampling scheme is suggested in Section IV-A, and original methods based on coloring the observation noise in Section IV-B1 and B2. Nevertheless, it is important to notice that any algorithm aiming at minimizing this cost presents this bias problem.

V. Projected Fixed-Point Approaches

Projected fixed-point approaches seek to minimize the distance between the estimated action-value function and the projection (the projection operator being written Π) of the image of this function under a Bellman operator onto the hypothesis space $\mathcal{H}$

$$J(\theta) = \|\hat{Q}_\theta - \Pi T\hat{Q}_\theta\|^2 \text{ with } \Pi f = \underset{\hat{f} \in \mathcal{H}}{\operatorname{argmin}} \|f - \hat{f}\|^2. \quad (15)$$

This is illustrated in Fig. 1. The action-value function estimate $\hat{Q}_\theta$ lies in the hypothesis space $\mathcal{H}$. Its image under a Bellman operator $T\hat{Q}_\theta$ does not necessarily lie on this hypothesis space. Residual approaches of Section IV try to minimize the distance between these two functions, i.e., the line ① in Fig. 1, with the drawback that using a sampled Bellman operator leads to biased estimates, as discussed before. The function $T\hat{Q}_\theta$ can be projected onto the hypothesis space, this projection minimizing the distance between $T\hat{Q}_\theta$ and the hypothesis space (line ② in Fig. 1). Projected fixed-point methods aim at minimizing the distance between this projection and $\hat{Q}_\theta$, represented by line ③ in Fig. 1.

Contrary to bootstrapping and residual approaches, least-squares-based algorithms have been introduced before stochastic-gradient-based ones for solving the projected fixed-point problem. Thus, they are reviewed first.

A. Least-Squares-Based Approaches

This section reviews algorithms that use a least-squares approach to minimize the empirical cost linked to (15)

$$\theta_i = \underset{\theta\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(\hat{Q}_\theta(s_j,a_j) - \hat{Q}_{\omega_\theta}(s_j,a_j)\right)^2 \quad (16)$$

$$\text{with } \omega_\theta = \underset{\omega\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(\hat{Q}_\omega(s_j,a_j) - \hat{T}_j\hat{Q}_\theta\right)^2. \quad (17)$$

Obviously, cost related to (16) is minimized for $\theta = \omega_\theta$ (assuming that this equation has a solution, which can at least be ensured for the evaluation operator and a linear parametrization). Therefore, the nested optimization problems (16) and (17) can be summarized as $\theta_i = \omega_{\theta_i}$

$$\theta_i = \underset{\omega\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(\hat{Q}_\omega(s_j,a_j) - \hat{T}_j\hat{Q}_{\theta_i}\right)^2. \quad (18)$$

Notice that, as θ_i appears in both sides of this equation, this is not a pure quadratic cost function but a fixed-point problem. The least-squares temporal differences (LSTD) algorithm [12] assumes a linear parameterization and the (sampled) Bellman evaluation operator in order to solve the above optimization problem. The statistically linearized LSTD (slLSTD) algorithm [13] generalizes it to nonlinear parameterizations and to the (sampled) Bellman optimality operator thanks to a statistical linearization process (the generalization from LSTD to slLSTD being quite close to the generalization from GPTD to KTD).

1) Least-Squares Temporal Differences: LSTD[6] assumes a linear parameterization as well as the sampled Bellman evaluation operator. Using the same notations as before, the optimization problem (18) can be rewritten as

$$\theta_i = \underset{\omega\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(r_j + \gamma\,\phi_{j+1}^T\theta_i - \phi_j^T\omega\right)^2.$$

Thanks to linearity in ω (linear parameterization assumption), this can be analytically solved, as

$$\theta_i = \left(\sum_{j=1}^{i}\phi_j\phi_j^T\right)^{-1}\sum_{j=1}^{i}\phi_j\left(r_j + \gamma\,\phi_{j+1}^T\theta_i\right).$$

Thanks to linearity in θ_i (linear parameterization and evaluation operator assumptions), the parameter vector can be isolated

$$\theta_i = \left(\sum_{j=1}^{i}\phi_j\left(\phi_j - \gamma\,\phi_{j+1}\right)^T\right)^{-1}\sum_{j=1}^{i}\phi_j r_j. \quad (19)$$

Equation (19) defines the (batch) LSTD estimate. Let us write $M_i^{-1} = \sum_{j=1}^{i}\phi_j\Delta\phi_j^T$. Thanks to the Sherman–Morrison formula, a recursive form of this estimation process can be obtained (assuming some priors θ_0 and M_0), as

$$K_i = \frac{M_{i-1}\phi_i}{1 + \left(\phi_i - \gamma\,\phi_{i+1}\right)^T M_{i-1}\phi_i}$$
$$\theta_i = \theta_{i-1} + K_i\left(r_i + \gamma\,\phi_{i+1}^T\theta_{i-1} - \phi_i^T\theta_{i-1}\right)$$
$$M_i = M_{i-1} - K_i\left(M_{i-1}^T\left(\phi_i - \gamma\,\phi_{i+1}\right)\right)^T.$$

Once again, K_i is a gain and $r_i + \gamma\,\phi_{i+1}^T\theta_{i-1} - \phi_i^T\theta_{i-1}$ a temporal difference error, to be linked to the Widrow–Hoff update (7).

LSTD can be slightly modified to have an improved computational cost [45] (assuming that the features are sparse, which is not necessarily the case), and it can be "mixed" with a residual approach [46].

2) Statistically Linearized LSTD: The slLSTD algorithm [13] generalizes LSTD: it does not assume either a linear parameterization or the Bellman evaluation operator. The corresponding optimization problem is therefore (this equation being valid thanks to the subsequent linearization)

$$\theta_i = \underset{\omega\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(r_j + \gamma\,\hat{P}_j\hat{Q}_{\theta_i} - \hat{Q}_\omega(s_j,a_j)\right)^2. \quad (20)$$

How slLSTD generalizes LSTD is very close to how KTD generalizes GPTD: a statistical linearization is performed, which allows solving this optimization problem analytically and recursively. Equation (20) can be linked to the following observation model (n_j, being here a unitary white and centered observation noise):

$$r_j + \gamma\,\hat{P}_j\hat{Q}_{\theta_i} = \hat{Q}_\omega(s_j,a_j) + n_j. \quad (21)$$

The noise is chosen unitary to strengthen parallel to LSTD, but extension to nonunitary noise is straightforward. As for KTD, a statistical linearization is performed. However, here two different quantities have to be linearized: $\hat{Q}_\omega(s_j,a_j)$ and $\hat{P}_j\hat{Q}_{\theta_i}$.

Assume that n parameter vectors $\omega^{(k)}$ of associated weights w_k are sampled (using the unscented transform), and that their images are computed

$$\left(\omega^{(k)}, q_j^{(k)} = \hat{Q}_{\omega^{(k)}}(s_j,a_j)\right)_{1\le k\le n}.$$

Using the statistical linearization process explained in Section IV-B2 and defining the statistics of interest according to (12) and (13), the following linear observation model is obtained:

$$\hat{Q}_\omega(s_j,a_j) = A_j\omega + b_j + e_j$$
$$\text{with}$$
$$A_j = P_{q_j\omega}P_\omega^{-1},\ b_j = \bar{q}_j - A_j\bar{\omega}$$
$$\text{and } P_{e_j} = P_{q_j} - A_jP_\omega A_j^T. \quad (22)$$

Recall that the noise e_j is centered and can be sampled as $e_j^{(k)} = q_j^{(k)} - (A_j\omega^{(k)} + b_j)$. The term $\hat{P}\hat{Q}_{\theta_i}(s_j,a_j)$ also needs to be linearized. Assume that n parameter vectors $\theta_i^{(k)}$ of

[6] Notice that LSTD has not been originally introduced as a projected fixed-point method (like in [15]), but as a residual least-squares debiased using instrumental variables (see [45] for details).

associated weights ω_k are sampled (again using the unscented transform), and that their images are computed

$$\left(\theta_i^{(k)},\, p_{q_j}^{(k)} = \hat{P}\,\hat{Q}_{\theta_i^{(k)}}(s_j, a_j)\right)_{1\leq k\leq n}.$$

Using the statistical linearization process explained in Section IV-B2 and defining the statistics of interest accordingly to (12)–(13), the following linear observation model is obtained:

$$\hat{P}\,\hat{Q}_{\theta_i}(s_j, a_j) = C_j\theta_i + d_j + \epsilon_j \quad \text{with} \quad C_j = P_{p_{q_j}\theta_i} P_{\theta_i}^{-1},\; d_j = \bar{p}_{q_j} - C_j\bar{\theta}_i \tag{23}$$

and

$$P_{\epsilon_j} = P_{p_{q_j}} - C_j P_{\theta_i} C_j^T. \tag{24}$$

Recall that the noise ϵ_j is centered and can be sampled as $\epsilon_j^{(k)} = p_{q_j}^{(k)} - (C_j\theta_i^{(k)} + d_j)$. Notice also that $\bar{\theta}_i$ is not equal to θ_i *a priori*.

Linearized models (22) and (23) can be injected into observation model (21), giving

$$\begin{aligned} & r_j + \gamma\,\left(C_j\theta_i + d_j + \epsilon_j\right) = A_j\omega + b_j + e_j + n_j \\ \Leftrightarrow\; & r_j + \gamma\,\left(C_j\theta_i + d_j\right) = A_j\omega + e_j - \gamma\,\epsilon_j + n_j. \end{aligned}$$

The linearization error is taken into account in the centered noise u_j of variance P_{u_j}

$$u_j = n_j + e_j - \gamma\,\epsilon_j \text{ and } P_{u_j} = \mathbb{E}[u_j^2].$$

This equivalent observation model leads to the following optimization problem, which can be solved analytically:

$$\theta_i = \underset{\omega\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \frac{1}{P_{u_j}} \left(r_j + \gamma\,\left(C_j\theta_i + d_j\right) - \left(A_j\omega + b_j\right)\right)^2 \tag{25}$$

$$= \left(\sum_{j=1}^{i} \frac{1}{P_{u_j}} A_j^T\left(A_j - \gamma\,C_j\right)\right)^{-1} \sum_{j=1}^{i} \frac{1}{P_{u_j}} A_j\left(r_j + \gamma\,d_j - b_j\right). \tag{26}$$

This is the statistically linearized variation of the optimization problem (20) [which explains the similarity between the LSTD batch estimate (19) and (26)]. Also, similar to what happens with KTD, the statistical linearization error is taken into account through the noise variance P_{u_j} (the bigger the statistical linearization error, the lesser the impact of the related sample on the learning process). The Sherman–Morrison formula allows again deriving a recursive estimation of θ_i. Assume that some priors θ_0 and M_0 are chosen; the slLSTD algorithm is defined as

$$\begin{aligned} K_i &= \frac{M_{i-1}A_i^T}{P_{u_i} + \left(A_i - \gamma\,C_i\right) M_{i-1} A_i^T} \\ \theta_i &= \theta_{i-1} + K_i\left(r_i + \gamma\,d_i - b_i - \left(A_i - \gamma\,C_i\right)\theta_{i-1}\right) \\ M_i &= M_{i-1} - K_i\left(M_{i-1}^T\left(A_i - \gamma\,C_i\right)^T\right)^T. \end{aligned}$$

Notice that, once again, this satisfies the Widrow–Hoff update, with a gain K_i and a temporal difference error $r_i + \gamma\,d_i - b_i - (A_i - \gamma\,C_i)\theta_{i-1}$ (which can be shown to simplify as $r_i + \gamma\,\bar{p}_{q_i} - \hat{p}_{q_i}$). Given [illegible] there still remains to choose how to sample parameter vectors (related to ω and θ_i) in order to compute A_i, b_i, C_i, d_i, and P_{u_i}.

As for KTD, the unscented transform is used to sample these parameter vectors. The parameter vector ω to be considered is the solution of (25), i.e., the solution of the fixed-point problem $\theta_i = \omega_{\theta_i}$. In this recursive estimation context, it is legitimate to linearize around the last estimate θ_{i-1}. The mean being chosen, the only remaining choice is the associated variance P_{i-1}. In [13], it is proposed to use the same variance matrix as would have been provided by a statistically linearized recursive least squares [47] used to perform supervised learning of the approximate action-value function given true observations of the Q-values. The fact that the unobserved action-values are not used to update the variance matrix tends to legitimate this choice. The associated matrix update is

$$P_i = P_{i-1} - \frac{P_{i-1}A_i^T A_i P_{i-1}}{1 + A_i P_{i-1} A_i^T}.$$

The same approach is used to compute C_i and d_i, coming from the statistical linearization of $\hat{P}_i\hat{Q}_{\theta_i}$. As before, the linearization is performed around the last estimate θ_{i-1} and considering the matrix variance Σ_{i-1} provided by a statistical linearized recursive least squares that would perform a supervised regression of $\hat{P}_i\hat{Q}_{\theta_i}$, i.e.,

$$\Sigma_i = \Sigma_{i-1} - \frac{\Sigma_{i-1}C_i^T C_i \Sigma_{i-1}}{1 + C_i\Sigma_{i-1}C_i^T}.$$

With these choices being made, A_i, b_i, C_i, and d_i can be computed, see [13] for details. A last thing is to compute the variance P_{u_i} of the noise $u_i = n_i + e_i - \gamma\,\epsilon_i$. The noise n_i is independent of others, and the variance of $e_i - \gamma\,\epsilon_i$ can be computed using the unscented transform, which provides an analytical expression for this variance.

Mixing all these elements, the slLSTD can be built. Notice that it can be easily shown that, with a linear parameterization and the (sampled) Bellman evaluation operator, slLSTD indeed reduces to LSTD [13] (this relies on the fact that the unscented transform is no longer an approximation for a linear mapping). Notice also that slLSTD being a projected fixed-point approach, it does not suffer from the bias problem even if the observation noise is assumed to be white (contrary to GPTD/KTD).

B. Stochastic Gradient Descent-Based Approaches

Algorithms presented in this section aim at minimizing the same cost function

$$J_i(\theta) = \underset{\theta\in\mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(\hat{Q}_\theta(s_j, a_j) - \hat{Q}_{\omega_\theta}(s_j, a_j)\right)^2$$

with

$$\hat{Q}_{\omega_\theta} = \hat{\Pi}\hat{T}\,\hat{Q}_\theta. \tag{27}$$

However, here a stochastic gradient descent approach is considered instead of the least-squares approach of the above section. Algorithms presented in Section V-B1, namely gradient temporal difference 2 (GTD2) and temporal difference

with gradient correction (TDC) [14], assume a linear parameterization and the (sampled) Bellman evaluation operator. Algorithms presented in Section V-B.2, namely nonlinear GTD2 (nlGTD2) and nonlinear TD (nlTDC) [15], extend them to the case of a nonlinear parameterization. The (linear) TDC algorithm has also been extended to eligibility traces [16] and to the Bellman optimality operator [17], these extensions being briefly presented in Section V-B.3.

1) Gradient Temporal Difference 2, Temporal Difference With Gradient Correction: GTD2 and TDC algorithms [14] aim at minimizing the cost function (27) while considering the Bellman evaluation operator, and they differ on the route taken to express the gradient followed to perform the stochastic gradient descent. Both methods rely on a linear parameterization, and are based on a reworked expression of the cost function. Let

$$\hat{\mathbf{Q}} = \left(\hat{Q}(s_1, a_1) \cdots \hat{Q}(s_i, a_i)\right)^T.$$

The cost function (27) can be rewritten as

$$J_i(\theta) = \|\hat{\mathbf{Q}}_\theta - \hat{\mathbf{Q}}_{\omega_\theta}\|^2 = (\hat{\mathbf{Q}}_\theta - \hat{\mathbf{Q}}_{\omega_\theta})^T(\hat{\mathbf{Q}}_\theta - \hat{\mathbf{Q}}_{\omega_\theta}). \quad (28)$$

Let also Φ_i (respectively, Φ_i') be the $p \times i$ matrix whose columns are the features $\phi(s_j, a_j)$ (respectively, $\phi(s_{j+1}, a_{j+1})$)

$$\Phi_i = \left[\phi(s_1, a_1) \ldots \phi(s_i, a_i)\right]$$
$$\text{and } \Phi_i' = \left[\phi(s_2, a_2) \ldots \phi(s_{i+1}, a_{i+1})\right].$$

Let R_i be the set of observed reward,

$$R_i = \left(r_1 \ldots r_i\right)^T.$$

As the parameterization is linear and as the Bellman evaluation is considered, the Q-values and their images through the sampled operator are given as

$$\hat{\mathbf{Q}}_\theta = \Phi_i^T \theta$$
$$\hat{T}\hat{\mathbf{Q}}_\theta = R_i + \gamma \left(\Phi_i'\right)^T \theta.$$

$\hat{\mathbf{Q}}_{\omega_\theta}$ is the projection of $\hat{T}\hat{\mathbf{Q}}_\theta$ onto the hypothesis space. Let us write Π_i the empirical projection, so

$$\hat{\mathbf{Q}}_{\omega_\theta} = \Phi_i^T \omega_\theta = \Pi_i \hat{T}\hat{\mathbf{Q}}_\theta \text{ with } \Pi_i = \Phi_i^T \left(\Phi_i \Phi_i^T\right)^{-1} \Phi_i.$$

The cost function (28) can thus be rewritten as

$$J_i(\theta) = \left\| \left(\Phi_i^T \theta - \Pi_i \left(R_i + \gamma \left(\Phi_i'\right)^T \theta\right)\right)\right\|^2.$$

Two basic properties of projection operator are useful here. First, $\Pi_i \Phi_i^T \theta = \Phi_i^T \theta$ (the hypothesis space is invariant under the projection operator). Second, $\Pi_i \Pi_i^T = \Pi_i$. Using these relationships, the cost can be rewritten as

$$J_i(\theta) = (\Phi_i^T \theta - R_i - \gamma (\Phi_i')^T \theta)^T \Pi_i (\Phi_i^T \theta - R_i - \gamma (\Phi_i')^T \theta).$$

Let $\delta_j(\theta) = r_j + \gamma \phi_{j+1}^T \theta - \phi_j^T \theta$ be the temporal difference error; $J_i(\theta)$ is finally given as

$$J_i(\theta) = \left(\sum_{j=1}^{i} \phi_j \delta_j(\theta)\right)^T \left(\sum_{j=1}^{i} \phi_j \phi_j^T\right)^{-1} \left(\sum_{j=1}^{i} \phi_j \delta_j(\theta)\right). \quad (29)$$

Notice that a GTD algorithm has been first introduced by considering a slightly different cost function [48]

$$J_i'(\theta) = \left(\sum_{j=1}^{i} \phi_j \delta_j(\theta)\right)^T \left(\sum_{j=1}^{i} \phi_j \delta_j(\theta)\right).$$

This explains why the algorithm of [14] is called GTD2.

The negative gradient of the cost function (29) is given by

$$-\frac{1}{2}\nabla_\theta J_i(\theta) = \left(\sum_{j=1}^{i} \left(\phi_j - \gamma \phi_{j+1}\right) \phi_j^T\right) \left(\sum_{j=1}^{i} \phi_j \phi_j^T\right)^{-1} \left(\sum_{j=1}^{i} \delta_j(\theta) \phi_j\right). \quad (30)$$

In order to avoid a bias problem, a second modifiable parameter vector $\omega \in \mathbb{R}^p$ is used to form a quasi-stationary estimate of the term $(\sum_{j=1}^{i} \phi_j \phi_j^T)^{-1}(\sum_{j=1}^{i} \delta_j(\theta)\phi_j)$, this being called the weight-doubling trick. Parameter vector θ is updated according to a stochastic gradient descent, as

$$\theta_i = \theta_{i-1} + \alpha_i \left(\phi_i - \gamma \phi_{i+1}\right) \phi_i^T \omega_{i-1}.$$

There remains to find an update rule for ω_i. In order to obtain an $O(p)$ algorithm, it is also estimated using a stochastic gradient descent [14]. One can remark that ω_i is actually the solution of a linear least-squares optimization problem

$$\omega_i = \left(\sum_{j=1}^{i} \phi_j \phi_j^T\right)^{-1} \sum_{j=1}^{i} \delta_j(\theta) \phi_j$$
$$= \underset{\omega \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(\phi_j^T \omega - \delta_j(\theta)\right)^2. \quad (31)$$

This suggests the following update rule for ω_i (minimization of (31) using a stochastic gradient descent):

$$\omega_i = \omega_{i-1} + \beta_i \phi_i \left(\delta_i(\theta_{i-1}) - \phi_i^T \omega_{i-1}\right).$$

Learning rates satisfy the classical stochastic approximation criterion. Moreover, they are chosen such that $\beta_i = \eta \alpha_i$ with $\eta > 0$. The GTD2 algorithm is thus

$$\theta_i = \theta_{i-1} + \alpha_i \left(\phi_i - \gamma \phi_{i+1}\right) \phi_i^T \omega_{i-1}$$
$$\omega_i = \omega_{i-1} + \beta_i \phi_i \left(\delta_i(\theta_{i-1}) - \phi_i^T \omega_{i-1}\right)$$

with

$$\delta_i(\theta) = r_i + \gamma \phi_{i+1}^T \theta - \phi_i^T \theta.$$

Under some assumptions, this algorithm can be shown to be convergent to the fixed point of ΠT [14]. Notice that, if this algorithm is derived from the gradient of an objective function, it is not a true stochastic gradient method because the expected weight update direction may differ from the direction of the negative gradient of the objective function (it is a pseudo-gradient method) [4].

By expressing the gradient in a slightly different way, another algorithm called TDC can be derived. Rewrite (30) as

$$-\frac{1}{2}\nabla_\theta J_i(\theta) = \left(\sum_{j=1}^{i}\delta_j(\theta)\phi_j\right) - \gamma\left(\sum_{j=1}^{i}\phi_{j+1}\phi_j^T\right)\left(\sum_{j=1}^{i}\phi_j\phi_j^T\right)^{-1}\left(\sum_{j=1}^{i}\delta_j(\theta)\phi_j\right).$$

This gives rise to the following update for θ, ω being updated as before:

$$\theta_i = \theta_{i-1} + \alpha_i\phi_i\delta_i(\theta_{i-1}) - \alpha_i\gamma\,\phi_{i+1}\phi_i^T\omega_{i-1}.$$

This algorithm is called TD with gradient correction because the first term, $\alpha_i\phi_i\delta_i(\theta_{i-1})$, is the same as for TD with function approximation (see Section III-A) under a linear function approximation architecture, and the second term, $-\alpha_i\gamma\,\phi_{i+1}\phi_i^T\omega_{i-1}$, acts as a correction. For TDC, learning rates α_i and β_i are chosen such as satisfying the classic stochastic approximation criterion, and such that $\lim_{i\to\infty}\frac{\alpha_i}{\beta_i} = 0$. This means that θ_i is updated on a slower timescale. The idea behind this is that ω_i should look stationary from the θ_i point of view. The TDC algorithm can be summarized as follows:

$$\theta_i = \theta_{i-1} + \alpha_i\phi_i\delta_i(\theta_{i-1}) - \alpha_i\gamma\,\phi_{i+1}\phi_i^T\omega_{i-1}$$
$$\omega_i = \omega_{i-1} + \beta_i\phi_i\left(\delta_i(\theta_{i-1}) - \phi_i^T\omega_{i-1}\right)$$

with

$$\delta_i(\theta) = r_i + \gamma\,\phi_{i+1}^T\theta - \phi_i^T\theta.$$

This algorithm can also be shown to be convergent (also to ΠT) under some assumptions [14].

2) Nonlinear Extensions: In [15], GTD2 and TDC are extended to the case of a general nonlinear parameterization $\hat{Q}_\theta$, as long as it is differentiable with respect to θ. The corresponding hypothesis space $\mathcal{H} = \{\hat{Q}_\theta|\theta \in \mathbb{R}^p\}$ is a differentiable submanifold onto which projecting is generally not computationally feasible. One may assume that the parameter vector θ is slightly updated in one step (given that the learning rates are usually small), which causes the surface of the submanifold to be close to linear. Therefore, projection is done onto the tangent plane defined as $\mathcal{TH} = \{(s,a) \in S\times A \to \omega^T\nabla_\theta\hat{Q}_\theta(s,a)|\omega \in \mathbb{R}^p\}$. In other words, linearization of $\hat{Q}_\theta(s,a)$ by a first order Taylor expansion. The corresponding projection operator Π_i^θ can be obtained as in Section V-B1, the tangent space being a hyperplane

$$\Pi_i^\theta = \left(\Phi_i^\theta\right)^T\left(\Phi_i^\theta\left(\Phi_i^\theta\right)^T\right)^{-1}\Phi_i^\theta$$
$$\text{with } \Phi_i^\theta = \left[\nabla_\theta\hat{Q}_\theta(s_1,a_1)\ \ldots\ \nabla_\theta\hat{Q}_\theta(s_i,a_i).\right]$$

The corresponding cost function can therefore be derived as in Section V-B1, with basically feature vectors $\phi(s,a)$ being replaced by $\nabla_\theta\hat{Q}_\theta(s_j,a_j)$, and the temporal difference error

$$r_j + \gamma\,\hat{Q}_\theta(s_{j+1},a_{j+1}) - \hat{Q}_\theta(s_j,a_j)$$

$$J_i(\theta) = \left(\sum_{j=1}^{i}\phi_j^\theta\delta_j(\theta)\right)^T\left(\sum_{j=1}^{i}\phi_j^\theta(\phi_j^\theta)^T\right)^{-1}\left(\sum_{j=1}^{i}\phi_j^\theta\delta_j(\theta)\right).$$

The gradient of this cost function is [15]

$$-\frac{1}{2}\nabla_\theta J_i(\theta) = \left(\sum_{j=1}^{i}\left(\phi_j^\theta - \gamma\,\phi_{j+1}^\theta\right)\left(\phi_j^\theta\right)^T\right)\omega_i + h(\theta,\omega_i) \quad (32)$$

$$= \sum_{j=1}^{i}\delta_j(\theta)\phi_j^\theta - \gamma\left(\sum_{j=1}^{i}\phi_{j+1}^\theta\left(\phi_j^\theta\right)^T\right)\omega_i + h(\theta,\omega_i) \quad (33)$$

with

$$\omega_i = \left(\sum_{j=1}^{i}\phi_j^\theta\left(\phi_j^\theta\right)^T\right)^{-1}\left(\sum_{j=1}^{i}\delta_j(\theta)\phi_j^\theta\right)$$

and

$$h(\theta,\omega) = -\sum_{j=1}^{i}\left(\delta_j(\theta) - \left(\phi_j^\theta\right)^T\omega\right)\left(\nabla^2\hat{Q}_\theta(s_j,a_j)\right)\omega.$$

GTD2 and TDC are generalized to nlGTD2 and nlTDC using a stochastic gradient descent on the above cost function. Parameter vector ω_i is updated as in Section V-B1

$$\omega_i = \omega_{i-1} + \beta_i\phi_i^{\theta_{i-1}}\left(\delta_i(\theta_{i-1}) - \left(\phi_i^{\theta_{i-1}}\right)^T\omega_{i-1}\right).$$

The nonlinear GTD2 algorithm performs a stochastic gradient descent according to (32)

$$\theta_i = \theta_{i-1} + \alpha_i\left(\left(\phi_i^{\theta_{i-1}} - \gamma\,\phi_{i+1}^{\theta_{i-1}}\right)\left(\phi_i^{\theta_{i-1}}\right)^T\omega_{i-1} - h_i\right)$$

with

$$h_i = \left(\delta_i(\theta_{i-1}) - \left(\phi_i^{\theta_{i-1}}\right)^T\omega_{i-1}\right)\left(\nabla^2_{\theta_{i-1}}\hat{Q}_\theta(s_i,a_i)\right)\omega_{i-1}.$$

Learning rates are chosen as for the TDC algorithm, i.e., satisfying the classic stochastic approximation criterion and such that $\lim_{i\to\infty}\frac{\alpha_i}{\beta_i} = 0$, which means that θ is updated on a slower timescale than ω. The nonlinear TDC algorithm performs a stochastic gradient descent according to (33)

$$\theta_i = \theta_{i-1} + \alpha_i\left(\phi_i^{\theta_{i-1}}\delta_i(\theta_{i-1}) - \gamma\,\phi_{i+1}^{\theta_{i-1}}\left(\phi_i^{\theta_{i-1}}\right)^T\omega_{i-1} - h_i\right)$$

with

$$h_i = \left(\delta_i(\theta_{i-1}) - \left(\phi_i^{\theta_{i-1}}\right)^T\omega_{i-1}\right)\left(\nabla^2_{\theta_{i-1}}\hat{Q}_\theta(s_i,a_i)\right)\omega_{i-1}.$$

Learning rates are chosen as above. Both nlGTD2 and nlTDC can be shown to be convergent to an undefined local minimum under some assumptions [15].

3) Extensions of TDC: The TDC algorithm (see Section V-B1 has been extended to eligibility traces in [16]. Moreover, this algorithm, called GQ(λ), allows off-policy learning, i.e., learning the value of one target policy while following another behavioral policy. This new algorithm (for which some convergence guarantees are provided) still minimizes the empirical cost function linked to (15). However, instead of the T^π Bellman operator considered so far, an eligibility-based T_λ^π operator is used (λ being the eligibility factor), this operator being defined as the expected λ return. See [16] for more details. Using eligibility traces induces a memory effect which prevents learning in an off-policy manner without caution, see [35] for example. To cope with this problem, importance sampling [49] is used in [16] (the idea of using importance sampling for value function approximation having been first introduced in [35]). They present GQ(λ) as an extension of Q-learning, which can be misleading. Actually, they consider off-policy learning (with a known and fixed target policy), but not the Bellman optimality operator.

Nevertheless, the TDC algorithm has also been extended to this operator [17] (this new algorithm being called Greedy-GQ). To do so, they consider the Bellman evaluation operator T^{π_θ} for a policy π_θ, which depends on the currently estimated action-value function (through parameters θ). Therefore, the considered policy is nonstationary (it evolves with parameters estimation). If π_θ is greedy with respect to the learnt value function, then it is equivalent to considering the Bellman optimality operator. However, in this case, there are some nondifferentiability problems (due to the max operator), and Fréchet subdifferentials are used to provide the algorithm [17]. A convergence analysis for these algorithms is also provided [17].

C. Iterative Projected Fixed Point

Methods presented so far in Section V aim at minimizing the distance between the action-value function and the projection of the image of this function under a Bellman operator onto the hypothesis space

$$J(\theta) = \|\hat{Q}_\theta - \Pi T \hat{Q}_\theta\|^2.$$

This can be interpreted as trying to find a fixed point of the operator ΠT, which is the composition of the projection operator Π and of one of the Bellman operators T. Assuming that this operator is a contraction (which is not always the case, it depends on the projection operator), there exists a unique fixed point which can be found by iterating the application of the ΠT operator

$$\hat{Q}_{\theta_i} = \Pi T \hat{Q}_{\theta_{i-1}}.$$

Methods presented in this section adopt this point of view to provide algorithms.

1) Fitted-Q: Fitted-Q is a batch algorithm. It assumes that a set of N transitions is available beforehand

$$\{(s_j, a_j, r_j, s_{j+1}, a_{j+1})\}_{1\le j\le N}.$$

An initial Q-function $\hat{Q}_{\theta_0}$ is considered, and estimates are refined by iterating the (sampled) ΠT operator

$$\hat{Q}_{\theta_i} = \hat{\Pi}\hat{T}\hat{Q}_{\theta_{i-1}}, \quad \forall i > 0.$$

Some important comments have to be made here. A sampled Bellman operator is used because, as usual transition probabilities are unknown. Most of time, fitted-Q suggests the sampled Bellman optimality operator, but this approach is of course still valid for the sampled Bellman evaluation operator. The Π operator indeed represents any supervised learning algorithm (which can be more or less directly seen as a projection). Notice that the representation of the estimated action-value function is not necessarily parametric (e.g., using a tree-based supervised learning algorithm [21]).

A practical example is given now. Assume that at iteration i the estimate $\hat{Q}_{\theta_{i-1}}$ is available. First, compute the image of this estimate through the sampled Bellman (here optimality) operator. This consists of computing the following training base, composed of state-action couples and estimated optimal Q-values: $\{(s_j, a_j, r_j + \gamma \max_{a\in A} \hat{Q}_{\theta_{i-1}}(s_{j+1}, a))\}_{1\le j\le N}$. A supervised learning algorithm is then used on this set, associating inputs (s_j, a_j) to estimated outputs $r_j + \gamma \max_{a\in A} \hat{Q}_{\theta_{i-1}}(s_{j+1}, a)$. This iteration is repeated until a stopping criterion is met (e.g., a maximum number of steps or little changes in the representation).

The fitted-Q idea probably dates back to [50]. Its convergence properties have been analyzed in [20], which reasons on the contraction property of the ΠT operator. In [51], its performance bounds in ℓ_p norm is analyzed. This is particularly judicious: if performance bounds of supervised learning algorithms are very often analyzed in ℓ_p norm, this is not the case for (approximate) dynamic programming which is most of the time analyzed in ℓ_∞ norm. Fitted-Q has been considered with many different function approximators. For example, see [52] for fitted-Q with a kernel-based regression, [53] for fitted-Q with neural networks trained by backpropagation, or [21] for fitted-Q with tree-based methods.

2) Least-Squares Policy Evaluation: The least-squares policy evaluation (LSPE) algorithm has been proposed[7] in [18]. It is directly introduced using the concept of eligibility traces, but this aspect is left out in this article. The LSPE algorithm can be roughly seen as a fitted-Q algorithm using a linear parameterization, the (sampled) Bellman evaluation operator, and for which a new training sample is added to the training set at each iteration. Thanks to linearity (linear parameterization and evaluation operator), an efficient online algorithm can be obtained.

The LSPE algorithm solves recursively $\hat{Q}_{\theta_i} = \Pi_i \hat{T} \hat{Q}_{\theta_{i-1}}$, the Π_i projection operator being defined in Section V-B1. Given the linearity, this can be rewritten as

$$\theta_i = \operatorname*{argmin}_{\theta\in\mathbb{R}^p} \sum_{j=1}^{i} \left(\phi_j^T\theta - r_j - \gamma\,\phi_{j+1}^T\theta_{i-1}\right)^2$$

[7] Actually, if the name LSPE has been introduced in [18] (in a multistep value-iteration context), the related algorithm was first introduced in [54], where it is built upon λ-policy-iteration.

$$= \left(\sum_{j=1}^{i} \phi_j \phi_j^T\right)^{-1} \sum_{j=1}^{i} \phi_j \left(r_j + \gamma\, \phi_{j+1}^T \theta_{i-1}\right)$$

$$= \theta_{i-1} + \left(\sum_{j=1}^{i} \phi_j \phi_j^T\right)^{-1} \sum_{j=1}^{i} \phi_j (r_j - (\phi_j - \gamma\, \phi_{j+1})^T \theta_{i-1}).$$

All terms involved can be computed recursively and efficiently, using notably the Sherman–Morrison formula

$$B_i^{-1} = \left(\sum_{j=1}^{i} \phi_j \phi_j^T\right)^{-1} = B_{i-1}^{-1} - \frac{B_{i-1}^{-1} \phi_i \phi_i^T B_{i-1}^{-1}}{1 + \phi_i^T B_{i-1}^{-1} \phi_i}$$

$$A_i = \sum_{j=1}^{i} \phi_j \left(\phi_j - \gamma\, \phi_{j+1}\right)^T = A_{i-1} + \phi_i \left(\phi_i - \gamma\, \phi_{i+1}\right)^T$$

$$b_i = \sum_{j=1}^{i} \phi_j r_j = b_{i-1} + \phi_i r_i.$$

The LSPE update can therefore be expressed in a form close to a Widrow–Hoff update

$$\theta_i = \theta_{i-1} + B_i^{-1} \left(b_i - A_i \theta_{i-1}\right).$$

Actually, the way LSPE is presented here differs from [18] in the sense that the B_i^{-1} matrix is originally scaled with a learning rate. The algorithm presented here is the case where the learning rate is chosen constant and equal to 1. This algorithm (with a constant learning rate equal to one) is shown to be convergent in [55]. Notice that ideas behind LSPE have other applications [56] and can also be linked to variational inequalities [57].

3) Q-Learning for Optimal Stopping Problems: The Q-learning for optimal stopping problems (Q-OSP) [19] extends LSPE to the (sampled) Bellman optimality operator, the parameterization being still linear. This algorithm is originally presented in the case of optimal stopping problems, which are a restrictive class of Markovian decision processes. However, it is presented here in the general case.

The derivation of this algorithm is the same as for the LSPE, by considering the optimality operator instead of the evaluation operator

$$\theta_i = \operatorname*{argmin}_{\theta \in \mathbb{R}^p} \sum_{j=1}^{i} \left(\phi_j^T \theta - r_j - \gamma \max_{a \in A} \left(\phi(s_{j+1}, a)^T \theta_{i-1}\right)\right)$$

$$= \left(\sum_{j=1}^{i} \phi_j \phi_j^T\right)^{-1} \sum_{j=1}^{i} \left(r_j + \gamma \max_{a \in A} \left(\phi(s_{j+1}, a)^T \theta_{i-1}\right)\right).$$

The matrix $\left(\sum_{j=1}^{i} \phi_j \phi_j^T\right)^{-1}$ can still be computed recursively and efficiently. However, this is not the case for the term $\sum_{j=1}^{i} (r_j + \gamma \max_{a \in A} (\phi(s_{j+1}, a)^T \theta_{i-1}))$, which requires remembering the whole trajectory and needs to be computed at each iteration. In [19], this algorithm is shown to be convergent for optimal stopping problems and under some assumptions, and approximate error bounds for off-line variations for the case of finite-state finite-MDPs are also proposed.

D. Summary

Projected fixed-point approaches seek at estimating a fixed point of the composed operator ΠT, Π being the projection onto the hypothesis space $\mathcal{H}$ and T being one of the Bellman operators. This fixed point can be computed directly, which corresponds to minimizing $\|\hat{Q}_\theta - \Pi T \hat{Q}_\theta\|^2$. The associated optimization problem is given by

$$\theta_i = \operatorname*{argmin}_{\omega \in \mathbb{R}^p} \sum_{j=1}^{i} \left(\hat{Q}_\omega(s_j, a_j) - \hat{T}_j \hat{Q}_{\theta_i}\right)^2.$$

This is (5) instantiated with $\xi = \theta_i$. If solved with a recursive least-squares approach, it gives the LSTD algorithm (or its statistical-linearization-based generalization). If solved using a stochastic gradient descent, it can provide GTD2 and TDC as well as their generalization to nonlinear parametrization, and their extensions to off-policy learning and to the Bellman optimality operator (GQ(λ) and greedy GQ).

Other approaches search for this fixed point using an iterative method: $\hat{Q}_{\theta_i} = \Pi T \hat{Q}_{\theta_{i-1}}$. The corresponding optimization problem is

$$\theta_i = \operatorname*{argmin}_{\omega \in \mathbb{R}^p} \sum_{j=1}^{i} \left(\hat{Q}_\omega(s_j, a_j) - \hat{T}_j \hat{Q}_{\theta_{i-1}}\right)^2.$$

This is (5) instantiated with $\xi = \theta_{i-1}$. Solved using a least-squares approach, it provides LSPE or Q-OSP, given what Bellman operator is considered. In a batch setting and using any supervised learning algorithm as the projection, it is fitted-Q. See the third column of Table I.

VI. Control

The ultimate goal of reinforcement learning is the control problem, i.e., computing an optimal control policy. We show here how the studied value function approximators integrate in the more general control problem.

A. Batch Learning

Assume that a set of trajectories, sampled according to some (explorative enough) behaviorial policy, is available. In a batch setting, the aim is to estimate an optimal policy from the available data. To do so, approximate dynamic programming (ADP) can be used.

Approximate policy iteration (API) is, generally speaking, a policy iteration scheme where the policy evaluation step is approximated. API is initialized with some policy π_0. At each time step k, policy π_k is evaluated using any approximate policy evaluation algorithm fed with the available data (i.e., any action-value function approximator based on the Bellman evaluation operator and used in an off-policy setting, as explained in Section III-A2). This gives an estimate $\hat{Q}_\theta^{\pi_k}$, which is used to improve the policy, i.e., $\pi_{k+1}(s) = \operatorname{argmax}_a \hat{Q}_\theta^{\pi_k}(s, a)$, notably for any s in the dataset. The process iterates until some stopping criterion is satisfied. A classic API algorithm is LSPI (least-squares policy iteration) [43], which uses LSTD as the value function approximator. However, any other policy evaluation algorithm could be used (TD-Q, GPTD, TDC, etc.).

Notice that, instead of working with an imposed set of data, new trajectories could be generated at each iteration (generally, using the current estimated policy slightly perturbed, in order to ensure variety of samples).

Approximate value iteration (AVI) consists of computing directly an estimate of the optimal value function. Generally speaking, any value function approximator based on the Bellman optimality operator (such as Q-learning, greedy-GQ, KTD, or slLSTD, for example) can serve as an AVI algorithm, even though fitted-Q is perhaps the best known. Given the available dataset and a value function approximator (based on T^*), an estimate $\hat{Q}^*_\theta$ of the optimal action-value function is computed, and it is used to provide an estimated optimal policy: $\hat{\pi}^*(s) = \text{argmax}_a \hat{Q}^*_\theta(s, a)$.

Notice that other ADP algorithms exist, such as those based on linear programming [58], [59], on modified policy iteration [60], or on conservative policy iteration [61]. However, these do not necessarily rely on a value function approximation algorithm.

B. Online Learning

Assume that the goal is now to learn in an online fashion, i.e., improving the policy while interacting with the system. This can be handled by an asynchronous API approach: at each time step, the greedy action (with respect to the currently estimated Q-function) is applied and the action-value function is updated using any policy evaluation algorithm (this implies using an online value function approximator, generally in an on-policy setting).

However, learning online induces a dilemma between exploration and exploitation and the greedy action should not be applied at each time step. Instead, from time to time an explorative action (suboptimal according to the currently estimated action-value function but with the potential to improve this estimate) should be chosen. A classic asynchronous API approach is SARSA: the Q-function is updated according to the TD-Q algorithm (on-policy setting) and the applied action is chosen according to an ϵ-greedy policy (greedy action with probability $1 - \epsilon$, random action else). However, any policy evaluation algorithm could be used instead (such as R-SGD, GPTD, LSTD, etc.), and other action selection schemes could be envisioned (such as a Gibbs sampling approach, for example).

Notice that learning control online induces nonstationarity problems: as the evaluated policy depends on the currently estimated Q-function, it actually changes at each time step, and the value function approximator should take this into account. While gradient-based policy evaluation algorithms handle this well, thanks to the use of a learning rate, this is not the case for least-squares-based algorithms. A simple solution to this problem is to add a forgetting factor to the minimized cost function

$$\theta_i = \underset{\omega \in \mathbb{R}^p}{\text{argmin}} \sum_{j=1}^{i} \beta^{i-j} \left(\hat{T}_j \hat{Q}_\xi - \hat{Q}_\omega(s_j, a_j)\right)^2$$

where $0 < \beta \leq 1$ is the forgetting factor. Notice that more general schemes for handling nonstationarity are possible through the original formulation of the KTD framework [11].

Similarly, it is possible to consider asynchronous AVI. This is exactly the same approach, replacing the policy evaluation algorithm by a value function approximator based on the Bellman optimality operator (this approximator must be able to learn in an online fashion, which could exclude fitted-Q). Asynchronous AVI tends to be less cautious than asynchronous API, in the sense that the estimated Q-function does not take into account the explorative steps of the action selection scheme (this cautious aspect also depends heavily on the exploration strategy). This is well illustrated by the cliff-walking task [2, Ch. 6.5].

C. Actor–Critic Algorithms

The online approaches discussed so far use an explicit representation for the action-value function, but not for the policy (which depends explicitly on the Q-function). Actor–critic methods [62]–[66] use an explicit representation for both these objects (Q and π). This approach as several advantages. Notably, it allows considering, in a principled way, continuous actions (which are a problem due to the greedy and the max operators of other approaches) and it somehow implicitly handles the dilemma between exploration and exploitation.

Assume that the policy π_w is parameterized by some vector w. Actor–critic algorithms aim at maximizing the value on a distribution d_0 of starting state: find w such that $\rho(w) = \mathbb{E}_{s \sim d_0}[V^{\pi_w}]$ is maximum. This can be done by performing a gradient ascent (or possibly a natural gradient ascent [65]): $w_{i+1} = w_i + \alpha_i \nabla\rho(w_i)$. Let d^π denote the so-called discounted stationary distribution, $d^\pi(s) = (1 - \gamma) \sum_{i \geq 0} \gamma^i P(s_i = s | s_0, \pi)$; the gradient admits an analytical expression [64]

$$\nabla\rho(w) = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\pi_w}, a \sim \pi_w(.|s)}[Q^{\pi_w}(s, a) \nabla_w \pi_w(a|s)].$$

Generally speaking, actor–critic approaches maximize ρ using a (natural) gradient ascent, and the gradient $\nabla\rho$ is estimated using an approximation of the Q-function Q^{π_w} (instead of the unavailable true Q-function).

However, one cannot use any approximation $\hat{Q}_\theta$ of Q^{π_w} [63], [64]. First, $\hat{Q}_\theta$ must be a good approximation of Q^{π_w} (in a least-squares sense, see [64] for details). Second, the parameterizations of $\hat{Q}_\theta$ and π_w must be "compatible", meaning that they should satisfy [63], [64]

$$\nabla_\theta \hat{Q}_\theta(s, a) = \nabla_w \ln \pi_w(a|s).$$

Therefore, choosing one parameterization imposes the other.

At this stage, one can remark that, if the parameterization is compatible, then $\mathbb{E}_{a \sim \pi(.|s)}[\hat{Q}_\theta(s, a)] = 0$: $\hat{Q}_\theta$ is not a Q-function, but an advantage function (generally defined as $A(s, a) = Q(s, a) - V(s)$). Algorithms studied in this survey do not allow learning directly such a function, and the literature offers variations of existing algorithms to learn $\hat{Q}_\theta(s, a)$ (e.g., see the LSTD-Q variation in [65]).

There is a simple solution to this problem. As shown in [67], it is possible to replace Q^{π_w} by a good approximation $\hat{Q}_{\theta,\xi}(s, a) = f_\theta(s, a) + g_\xi(s)$ with $\nabla_\theta f_\theta(s, a) =$

TABLE II
ALGORITHM PROPERTIES

	Policy Iteration (pi) Value Iteration (vi)	Linear (l) Nonlinear (nl)	Computational Complexity	Biased (b) Unbiased (ub)	ξ (see (5))	Stoch. grad. descent (sgd) rec. least-squares (rls)
TD-Q	pi	l/nl	$O(p)$	ub	θ_{j-1}	sgd
Q-learning	vi	l/nl	$O(p)$	ub	θ_{j-1}	sgd
FPKF	pi/vi	l	$O(p^2)$	ub	θ_{j-1}	rls
R-SGD	pi	l/nl	$O(p)$	b	ω	sgd
GPTD	pi	l	$O(p^2)$	b	ω	rls
KTD	pi/vi	l/nl	$O(p^2)$	b	ω	rls
GTD2/TDC	pi	l/nl	$O(p)$	ub	θ_i	sgd
Greedy-GQ	vi	l	$O(p)$	ub	θ_i	sgd
LSTD	pi	l	$O(p^2)$	ub	θ_i	rls
slLSTD	pi/vi	l/nl	$O(p^2)$	ub	θ_i	rls
LSPE	pi	l	$O(p^2)$	ub	θ_{i-1}	rls
Q-OSP	vi	l	*	ub	θ_{i-1}	rls
Fitted-Q	vi	l/nl	**	ub	θ_{i-1}	**

$\nabla_w \ln \pi_w(a|s)$ (no specific condition on g_ξ). This way, $\hat{Q}_{\theta,\xi}$ is actually an action-value function (not an advantage function), and all important theorems of [64] and [65] can be shown to hold. Therefore, one can plug its favorite parametric policy evaluation algorithm in an actor–critic approach, see [67] for details. However, notice that, if the compatibility condition is satisfied, the good approximation condition may not be perfectly satisfied. Nevertheless, it often works well in practice.

VII. ALGORITHMS COMPARISON

This section aims at comparing the surveyed algorithms. First, alternative taxonomies are provided. Then, we discuss when choosing what algorithm, depending on the context. Finally, we give some pointers to the literature, regarding experimental results.

A. Alternative Taxonomies

The surveyed algorithms have been divided into three categories (bootstrapping, residual, and projected fixed-point) and two subcategories (gradient-based or least-squares-based). Other taxonomies are possible, as summarized in Table II.

First, they estimate either the action-value function of a given policy (Q^π) or the optimal Q-function (Q^*) directly. In the first case, they can be used as the policy evaluation component of a more general API scheme, as explained above. In the second case, they provide directly the optimal policy through the estimated Q-function, which is an AVI scheme. This is depicted as pi/vi in Table II (equivalently, pi means that the Bellman evaluation operator is considered and vi means that the Bellman optimality operator is considered).

Some of the reviewed algorithms are only defined for linear parameterization of the action-value function, whereas others allow considering nonlinear architectures such as multilayered neural networks (mainly through a gradient descent or thanks to a statistical linearization). This is depicted as l/nl in Table II.

A point not discussed so far is the computational (and memory) complexity of the various algorithms. There are basically two cases. For stochastic gradient descent-based algorithms, complexity is in $O(p)$; and for recursive least-squares-based approach, complexity is in $O(p^2)$. Notice that for algorithms handling the Bellman optimality operator, these complexities have to be scaled by the number of actions (because of the max computation). Exceptions (to the two aforementioned cases) are fitted-Q (whose complexity depends on the considered supervised learning scheme) and Q-OSP (which is moreover linear in the length of the trajectory). Considering the sample efficiency of these algorithms, second-order approaches are usually more efficient (a notable exception being FPKF, see [68]).

Lastly, some algorithms produce biased estimates (residual approaches, if not combined with a double sampling scheme or with some colored noise), whereas the others provide unbiased estimates (bootstrapping and projected fixed-point). This is depicted as b/ub in Table II. We also recall how ξ is instantiated [see (5)] and whether the algorithm is based on a stochastic gradient descent (sgd) or a recursive least-squares (rls) approach.

B. Which Algorithm to Choose?

One can wonder when choosing what algorithm. Unfortunately, there is no easy answer to this. Nevertheless, we try to give some clues here.

First, one may choose between (asynchronous) API and AVI. We think that the choice between an online and a batch learning algorithm (that is asynchronous or not) is more linked to the studied problem than a real choice (is the data imposed? can we sample easily trajectories?). Choosing between API and AVI is a matter of taste (as far as we know, there is no definitive argument in favor of one or the other).

Second, one can choose between a residual and a bootstrapping or a projected fixed-point approach (bootstrapping and projected fixed-point approaches converge to the same limit when convergence occurs). Residual approaches are more stable and more predictable [69], but they suffer from the bias problem (which disappears when the transitions are deterministic). Other arguments in favor of each approach (residual versus projected fixed point) are developed in [70].

TABLE III
LINKS TO EXPERIMENTAL COMPARISONS

	gradBoot	FPKF	GPTD	KTD	gradPFP	LSTD	slLSTD	fitted-Q
gradBoot	[2], [27], [35]	[8], [68]	[68]	[11], [40], [72]	[14], [15], [17]	[12], [43], [68], [73]	[13]	[21]
FPKF	[8], [68]		[68]			[68]		
gradRes	[9]							
GPTD	[68]	[68]	[10], [25], [34], [42]	[11], [74]		[10], [68], [75]		
KTD	[11], [40], [72]		[11], [74]	[11], [76]		[11], [40], [41], [72]	[13]	[72]
gradPFP	[14], [15], [17]				[14], [15]		[13]	
LSTD	[12], [43], [68], [73]	[68]	[10], [68], [75]	[11], [40], [41], [72]		[43]	[13]	[21], [77]
slLSTD	[13]			[13]	[13]	[13]		
itPFP	[54], [68]	[68]	[68]			[68]		
Fitted-Q	[21]			[72]		[21], [77]		[20], [21], [51], [53]

Therefore, if the dynamic is (near-) deterministic (or even Lipschitz from our own experience), then a residual method could be chosen (stability argument); else, a projected fixed-point or bootstrap approach must be preferred.

If the residual approach is rejected, there is still the choice between bootstrapping and projected fixed-point approaches. Generally speaking, gradient-based bootstrapping algorithms offer less theoretical guarantees than their projected fixed-point counterparts, but they empirically learn faster and tuning meta-parameters is easier. This is no longer true for the least-squares-based algorithms: FPKF is much slower than LSTD (due to the bootstrapping), and they provide the same solution asymptotically.

Then, one can choose between a gradient-based and a least-squares-based approach. Least-squares approaches are more sample efficient (which can be theoretically argued in some cases, see, e.g., the finite sample analysis of LSTD [71]), but they also have a higher computational cost. So this choice is a tradeoff between sample efficiency and computational cost. If samples are costly to generate or imposed, we advocate the use of a least-squares approach. In the other hand, if there are many features (thus many parameters to be learned) and if the transitions are easy to generate, gradient-based methods may be preferred.

We have provided a unified derivation of surveyed algorithms, but most of them were introduced differently than in their original derivation. Notably, concerning GPTD [10] and KTD [11], the parameter vector to be learnt is modeled as a random variable of which mean and variance are estimated[8] Therefore, during learning, uncertainty information about parameters estimates is available. This can be used to compute some variance about the estimated action-value function, for any state–action couple. This can be useful for solving the dilemma between exploration and exploitation (e.g., it may be interesting to choose an action with a low estimated value if this estimate is uncertain or adopt an optimistic action selection). See [41] for a discussion on this subject. The same idea can be adapted to (sl)LSTD, less directly. See [13] for details.

One can also choose between linear and nonliear parameterization. If good linear features are available, it is certainly better to use them, thus to choose a linear parametric value function approximator. However, sometimes a multilayered artificial neural network can provide better results (generally speaking, the underlying hypothesis space may be richer). In this case, one has to choose an RL algorithm that is able to handle such nonlinearities. Notice that the only algorithms to come with theoretical guarantees in the nonlinear case are nlTDC and nlGTD2 (but KTD and slLSTD work well in practice, and learn faster, but at a higher computational cost).

Last but not least, the ease of implementation (or the availability of some library) and the ease of choosing the meta-parameters can also drive the choice of the algorithm. Generally speaking, linear least-squares-based algorithms are easier to tune (there is no learning rate).

C. Experimental Comparisons

Another point not addressed so far in this paper is the thorough experimental comparison of the reviewed algorithms. This is beyond the scope of this algorithmic survey, but Table III provides links to the literature, showing where a comparison between algorithm X and algorithm Y can be found (often within a control scheme). This table can be read line by line. Note that this survey does not pretend to be exhaustive (e.g., there are some application papers, notably for dialogue management in spoken dialog systems, but not all of them are reported). In Table III, gradBoot refers to gradient-based bootstrapping algorithms (TD-V, TD-Q, Q-learning), gradRes refers to gradient-based residual algorithms (R-SGD), gradPFP refers to gradient-based projected-fixed-point approaches (GTD2, TDC, and greedy-GQ), and itPFP refers to iterated fixed-point algorithms (LSPE and Q-OSP, fitted-Q being left out). If an algorithm is compared to itself in the table, it means either that it is actually a group of algorithms (as gradBoot, for example), or that this paper contains some illustrative example (as the illustration of the variance information provided by the GPTD algorithm in [10]) or some problem-specific experiment/large-scale application.

VIII. OTHER VALUE FUNCTION APPROXIMATORS

This paper has surveyed parametric value function approximators, but there exist other ways to estimate a value function. Some of them are briefly mentioned below.

[8] What allows us to derive it differently here is the close link between [illegible], and Bayesian filtering.

A. Extension to Eligibility Traces

TD learning with eligibility traces [2], known as TD(λ), provides a nice bridge between temporal difference learning and Monte Carlo learning. By controlling the bias/variance tradeoff [78], their use can significantly speed up learning. When the value function is approximated through a linear architecture, the depth λ of the eligibility traces is also known to control the quality of approximation [26].

Using eligibility traces amounts to looking for the fixed point of the following variation of the Bellman operator [1] (here we assume a fixed policy, i.e., the evaluation problem): $T^\lambda V = (1-\lambda)\sum_{i=0}^{\infty} \lambda^i T^{i+1} V$. As we have introduced a sampled Bellman operator to derive the surveyed algorithms, the same can be done for this T^λ operator. For a partial trajectory $(s_k, r_k, s_{k+1})_{j \le k \le i-1}$, we define $\hat{T}^\lambda_{j,i}$ the sampled and truncated operator (e.g., see [68])

$$\hat{T}^\lambda_{j,i} V = V(s_j) + \sum_{k=j}^{i} (\gamma\,\lambda)^{k-j} (r_k + \gamma\, V(s_{k+1}) - V(s_k)).$$

This being defined, the work done in this survey can be repeated by replacing $\hat{T}_j$ by $\hat{T}^\lambda_{j,i}$ in (5)

$$\theta_i = \underset{\omega \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{j=1}^{i} \left(\hat{T}^\lambda_{j,i} \hat{V}_\xi - \hat{V}_\omega(s_j, a_j) \right)^2.$$

Notice that this only holds for the evaluation problem (the Bellman optimality operator is not considered) and in the on-policy case. To handle the off-policy case, it is necessary to use also importance sampling. See [68] and [79], where this approach is used to derive new off-policy least-squares-based and eligibility-traces-based algorithms.

B. Nonparametric Approaches

Given an approximation architecture, this survey has shown that many approaches exist to learn the corresponding parameters. However, choosing a judicious hypothesis space is highly problem-dependent and is generally a difficult issue. Therefore, we think that research fields such as feature construction, feature selection, or, more generally, nonparametric value function approximation are of primary importance.

The GPTD algorithm [25] is kernel-based and nonparametric in its more general form (not presented here). Using the same idea (i.e., constructing the span of the feature space implicitly defined by a Mercer kernel), the LSTD algorithm has been kernelized [80] as well as the LSPE algorithm [81]. More generally, kernelized approaches to value function approximation are discussed in [82].

Feature selection via ℓ_1-regularization has also been envisioned for nonparametric value function approximation: adding an ℓ_1-penalty term imposes some parameters to be exactly zero, which allows handling the case $p \gg n$ (much more features than samples). Adding such a penalty term to value function approximation is not straightforward and there are many ways to do it. See [83] for an approach based on residual minimization and [84]–[88] for approaches based on projected fixed point ([88] provides a synthetic overview of these methods).

Another approach consists of generating features, based on the Bellman error [89]–[93] or upon graph structure built from observed trajectories through the state space [94]. State aggregation (e.g., [95] and [96]) can also be seen as a feature generation scheme. Other methods are based on local averagers or kernel smoothers (usually Nadaraya–Watson-like estimators) [52], [97], [98]. These are examples among others, and this general topic would deserve another survey.

C. Other Criteria

In this paper, the quality of a policy is quantified by the value function defined as the expected discounted cumulative reward, $V^\pi(s) = \mathbb{E}[\sum_{i \ge 0} \gamma^i r_i | s_0 = s, \pi]$. Other criteria can be envisioned, such as the finite horizon criterion (and the value function is then $V^\pi(s) = \mathbb{E}[\sum_{i=0}^{H} r_i | s_0 = s, \pi]$ for some predefined horizon H) or the average-reward criterion (for which the value function is defined as $V^\pi(s) = \lim_{H \to \infty} 1/H \mathbb{E}[\sum_{i=0}^{H} r_i | s_0 = s, \pi]$). The γ-discounted criterion considered in this paper is probably the most frequently used in the literature, but notice that done here cannot be directly adapted to the other criteria.

IX. Conclusion

In this paper, we reviewed a large part of the state of the art in parametric (action-) value function approximation. Basically, it has been shown that all these approaches can be derived from the unified cost function (5). Further, they can be categorized into three main classes, given the considered cost function (related to bootstrapping, residual, or projected fixed point). In each of these groups, they can be categorized given that the cost function is minimized using a stochastic gradient descent or a recursive least-squares approach (except fitted-Q, which can be considered with any supervised learning algorithm). Projected fixed-point approaches can be divided into two approaches, given that the cost function is directly minimized or that the underlying possible fixed point is searched for using an iterative scheme. All of this is summarized in Table I. Sections VI–VIII gave clues on how the surveyed value function approximators can be used for control, how they can be compared and what are the other possible (not covered) approaches for estimating a value function.

Acknowledgment

The authors would like to thank J. Fix and B. Scherrer for proofreading earlier versions of this paper, and the anonymous reviewers for useful comments and suggestions.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming* (Optimization and Neural Computation). Belmont, MA, USA: Athena Scientific, 1996.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 3rd ed. Cambridge, MA, USA: MIT Press, Mar. 1998.

[3] O. Sigaud and O. Buffet, *Markov Decision Processes and Artificial Intelligence*. New York, USA: Wiley, 2010.

[4] C. Szepesvári, *Algorithms for Reinforcement Learning*. San Mateo, CA, USA: Morgan & Claypool, 2010.

[5] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art* (Adaptation, Learning, and Optimization). New York, USA: Springer-Verlag, 2012.

[6] L. Busoniu, R. Babuska, B. D. Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators* (Automation and Control Engineering). Boca Raton, FL, USA: CRC Press, 2010.

[7] W. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. New York, USA: Wiley, 2007.

[8] D. Choi and B. Van Roy, "A generalized Kalman filter for fixed point approximation and efficient temporal-difference learning," *Discrete Event Dyn. Syst.*, vol. 16, no. 2, pp. 207–239, Apr. 2006.

[9] L. C. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *Proc. Int. Conf. Mach. Learn.*, 1995, pp. 30–37.

[10] Y. Engel, "Algorithms and representations for reinforcement learning," Ph.D. dissertation, Interdisciplinary Center Neural Comput., Hebrew Univ., Jerusalem, Israel, Apr. 2005.

[11] M. Geist and O. Pietquin, "Kalman temporal differences," *J. Artif. Intell. Res.*, vol. 39, no. 1, pp. 483–532, 2010.

[12] S. J. Bradtke and A. G. Barto, "Linear least-squares algorithms for temporal difference learning," *Mach. Learn.*, vol. 22, nos. 1–3, pp. 33–57, 1996.

[13] M. Geist and O. Pietquin, "Statistically linearized least-squares temporal differences," in *Proc. IEEE Int. Conf. Ultra Modern Control Syst.*, Oct. 2010, pp. 450–457.

[14] R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora, "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *Proc. Int. Conf. Mach. Learn.*, 2009, pp. 993–1000.

[15] H. Maei, C. Szepesvari, S. Bhatnagar, D. Precup, D. Silver, and R. S. Sutton, "Convergent temporal-difference learning with arbitrary smooth function approximation," in *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Cambridge, MA, USA: MIT Press, 2009, pp. 1204–1212.

[16] H. R. Maei and R. S. Sutton, "GQ(λ): A general gradient algorithm for temporal-differences prediction learning with eligibility traces," in *Proc. Conf. Artif. General Intell.*, 2010, pp. 1–6.

[17] H. R. Maei, C. Szepesvari, S. Bhatnagar, and R. S. Sutton, "Toward off-policy learning control with function approximation," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 1–8.

[18] A. Nedić and D. P. Bertsekas, "Least squares policy evaluation algorithms with linear function approximation," *Discrete Event Dyn. Syst., Theory Appl.*, vol. 13, nos. 1–2, pp. 79–110, Jan. 2003.

[19] H. Yu and D. P. Bertsekas, "Q-learning algorithms for optimal stopping based on least squares," in *Proc. Eur. Control Conf.*, Jul. 2007, pp. 1–15.

[20] G. Gordon, "Stable function approximation in dynamic programming," in *Proc. Int. Conf. Mach. Learn.*, 1995, pp. 1–8.

[21] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, Apr. 2005.

[22] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, 1988.

[23] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Dept. Eng., Cambridge Univ., Cambridge, U.K., Tech. Rep. CUED/F-INFENG/TR 166, 1994.

[24] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, 1992.

[25] Y. Engel, S. Mannor, and R. Meir, "Bayes meets Bellman: The Gaussian process approach to temporal difference learning," in *Proc. Int. Conf. Mach. Learn.*, 2003, pp. 154–161.

[26] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Trans. Autom. Control*, vol. 42, no. 5, pp. 674–690, May 1997.

[27] G. Tesauro, "Temporal difference learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.

[28] H. van Seijen, H. van Hasselt, S. Whiteson, and M. Wiering, "A theoretical and empirical analysis of expected Sarsa," in *Proc. IEEE Int. Symp. Adapt. Dynamic Program. Reinforce. Learn.*, Mar.–Apr. 2009, pp. 177–184.

[29] H. Yu, "Least squares temporal difference methods: An analysis under [illegible] Helsinki, Finland, Tech. Rep. C-2010-39, 2010.

[30] F. S. Melo, S. P. Meyn, and M. I. Ribeiro, "An analysis of reinforcement learning with function approximation," in *Proc. Int. Conf. Mach. Learn.*, 2009, pp. 664–671.

[31] A. Antos, C. Szepesvári, and R. Munos, "Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path," *Mach. Learn.*, vol. 71, no. 1, pp. 89–129, Apr. 2008.

[32] A. Y. Kruger, "On Fréchet subdifferentials," *J. Math. Sci.*, vol. 116, no. 3, pp. 3325–3358, 2003.

[33] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis* (Probability and Statistics). New York, USA: Wiley, 1984.

[34] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with gaussian processes," in *Proc. Int. Conf. Mach. Learn.*, 2005, pp. 201–208.

[35] D. Precup, R. S. Sutton, and S. P. Singh, "Eligibility traces for off-policy policy evaluation," in *Proc. Int. Conf. Mach. Learn.*, 2000, pp. 759–766.

[36] S. J. Julier and J. K. Uhlmann, "A new extension of the Kalman filter to nonlinear systems," in *Proc. 3rd Int. Symp. Aerosp. Defense Sens. Simul. Controls*, 1997, pp. 182–193.

[37] S. J. Julier, "The scaled unscented transformation," in *Proc. Amer. Control Conf.*, vol. 6. 2002, pp. 4555–4559.

[38] P. M. Nørgård, N. K. Poulsen, and O. Ravn, "New developments in state estimation for nonlinear systems," *Automatica*, vol. 36, no. 11, pp. 1627–1638, Nov. 2000.

[39] R. van der Merwe, "Sigma-point kalman filters for probabilistic inference in dynamic state-space models," Ph.D. dissertation, OGI School Science Eng., Oregon Health & Science Univ., Portland, OR, USA, Apr. 2004.

[40] M. Geist and O. Pietquin, "Eligibility traces through colored noises," in *Proc. IEEE Int. Conf. Ultra Modern Control Syst.*, Oct. 2010, pp. 458–465.

[41] M. Geist and O. Pietquin, "Managing uncertainty within the KTD framework," *J. Mach. Learn. Res. (W&C Proc.)*, vol. 15, pp. 157–168, Mar. 2011.

[42] L. Daubigney, M. Gasic, S. Chandramohan, M. Geist, O. Pietquin, and S. Young, "Uncertainty management for on-line optimisation of a POMDP-based large-scale spoken dialogue system," in *Proc. Annu. Conf. Int. Speech Commun. Assoc.*, Aug. 2011, pp. 1301–1304.

[43] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, Dec. 2003.

[44] T. Söderström and P. Stoica, "Instrumental variable methods for system identification," *Circuits Syst. Signal Process.*, vol. 21, no. 1, pp. 1–9, Jan.–Feb. 2002.

[45] A. Geramifard, M. Bowling, and R. S. Sutton, "Incremental least-squares temporal difference learning," in *Proc. 21st Nat. Conf. Artif. Intell.*, 2006, pp. 356–361.

[46] J. Johns, M. Petrik, and S. Mahadevan, "Hybrid least-squares algorithms for approximate policy evaluation," *Mach. Learn.*, vol. 76, nos. 2–3, pp. 243–256, Sep. 2009.

[47] M. Geist and O. Pietquin, "Statistically linearized recursive least squares," in *Proc. IEEE Int. Workshop Mach. Learn. Signal Process.*, Sep. 2010, pp. 272–276.

[48] R. S. Sutton, C. Szepesvari, and H. R. Maei, "A convergent O(n) Algorithm for off-policy temporal-difference learning with linear function approximation," in *Advances in Neural Information Processing Systems*, Vancouver, BC, Canada: MIT Press, 2008.

[49] B. D. Ripley, *Stochastic Simulation*, New York, USA: Wiley, 1987.

[50] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Develop.*, vol. 3, no. 3, pp. 210–229, Jul. 1959.

[51] R. Munos, "Performance bounds in L_p-norm for approximate value iteration," *SIAM J. Control Optim.*, vol. 46, no. 2, pp. 541–561. Jan. 2007.

[52] D. Ormoneit and Ś. Sen, "Kernel-based reinforcement learning," *Mach. Learn.*, vol. 49, nos. 2–3, pp. 161–178, Nov. 2002.

[53] M. Riedmiller, "Neural fitted q iteration—First experiences with a data efficient neural reinforcement learning method," in *Proc. Eur. Conf. Mach. Learn.*, 2005, pp. 317–328.

[54] D. P. Bertsekas and S. Ioffe, "Temporal differences-based policy iteration and applications in neuro-dynamic programming," Labs Information & Decision Systems, MIT, Cambridge, MA, USA, Tech. Rep. LIDS-P-2349, 1996.

[55] D. P. Bertsekas, V. Borkar, and A. Nedic, "Improved temporal difference methods with linear function approximation," in *Learning and Approximate Dynamic Programming*. Piscataway, NJ, USA: IEEE Press, 2004, pp. 231–235.

[56] D. P. Bertsekas and H. Yu, "Projected equation methods for approximate solution of large linear systems," *J. Comput. Appl. Math.*, vol. 227, no. 1, pp. 27–50, May 2009.

[57] D. P. Bertsekas, "Projected equations, variational inequalities, and temporal difference methods," in *Proc. IEEE Int. Symp. Adapt. Dynamic Program. Reinforce. Learn.*, 2009, pp. 1–28.

[58] D. P. de Farias and B. V. Roy, "The linear programming approach to approximate dynamic programming," *Oper. Res.*, vol. 51, no. 6, pp. 850–865, 2003.

[59] V. V. Desai, V. F. Farias, and C. C. Moallemi, "The smoothed approximate linear program," in *Advances in Neural Information Processing Systems*, New York, USA: Columbia Univ. Press, Aug. 2009.

[60] B. Scherrer, V. Gabillon, M. Ghavamzadeh, and M. Geist, "Approximate modified policy iteration," in *Proc. Int. Conf. Mach. Learn.*, 2012, pp. 1–21.

[61] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *Proc. 19th Int. Conf. Mach. Learn.*, 2002, pp. 267–274.

[62] A. Barto, R. Sutton, and C. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. 13, no. 5, pp. 834–846, Sep.–Oct. 1983.

[63] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM J. Control Optim.*, vol. 42, no. 4, pp. 1143–1166, 2003.

[64] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 1999, pp. 1057–1063.

[65] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, nos. 7–9, pp. 1180–1190, 2008.

[66] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Incremental natural actor-critic algorithms," in *Advances in Neural Information Processing Systems*, Vancouver, Canada: MIT Press, 2007.

[67] M. Geist and O. Pietquin, "Revisiting natural actor-critics with value function approximation," in *Proc. Int. Conf. Model. Decisions Artif. Intell.*, 2010, pp. 207–218.

[68] B. Scherrer and M. Geist, "Recursive least-squares learning with eligibility traces," in *Proc. Eur. Workshop Reinforce. Learn.*, 2011, pp. 11–12.

[69] R. Munos, "Error bounds for approximate policy iteration," in *Proc. Int. Conf. Mach. Learn.*, 2003, pp. 560–567.

[70] B. Scherrer, "Should one compute the temporal difference fix point or minimize the Bellman residual? The unified oblique projection view," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 1–8.

[71] A. Lazaric, M. Ghavamzadeh, and R. Munos, "Finite-sample analysis of LSTD," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 1–14.

[72] O. Pietquin, M. Geist, and S. Chandramohan, "Sample efficient on-line learning of optimal dialogue policies with Kalman temporal differences," in *Proc. Int. Joint Conf. Artif. Intell.*, Jul. 2011, pp. 1878–1883.

[73] J. A. Boyan, "Technical update: Least-squares temporal difference learning," *Mach. Learn.*, vol. 49, nos. 2–3, pp. 233–246, 1999.

[74] L. Daubigney, M. Geist, and O. Pietquin, "Off-policy learning in large-scale POMDP-based dialogue systems," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Mar. 2012, pp. 4989–4992.

[75] C. W. Phua and R. Fitch, "Tracking value function dynamics to improve reinforcement learning with piecewise linear function approximation," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 751–758.

[76] M. Keramati, A. Dezfouli, and P. Piray, "Speed/accuracy trade-off between the habitual and the goal-directed processes," *PLoS Comput. Biol.*, vol. 7, no. 5, p. e1002055, 2011.

[77] O. Pietquin, M. Geist, S. Chandramohan, and H. Frezza-Buet, "Sample-efficient batch reinforcement learning for dialogue management optimization," *ACM Trans. Speech Lang. Process.*, vol. 7, no. 3, pp. 7:1–7:21, 2011.

[78] M. Kearns and S. Singh, "Bias-variance error bounds for temporal difference updates," in *Proc. Conf. Learn. Theory*, 2000, pp. 142–147.

[79] B. Scherrer and M. Geist, "Recursive least-squares off-policy learning with eligibility traces," INRIA, France, Tech. Rep. hal-00644516, 2012.

[80] X. Xu, D. Hu, and X. Lu, "Kernel-based least squares policy iteration [illegible] vol. 18, no. 4, [illegible] 2007.

[81] T. Jung and D. Polani, "Kernelizing LSPE(λ)," in *Proc. IEEE Symp. Approx. Dynamic Program. Reinforce. Learn.*, Apr. 2007, pp. 338–345.

[82] G. Taylor, and R. Parr, "Kernelized value function approximation for reinforcement learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 1017–1024.

[83] M. Loth, M. Davy, and P. Preux, "Sparse temporal difference learning using LASSO," in *Proc. IEEE Int. Symp. Approx. Dynamic Program. Reinforce. Learn.*, Apr. 2007, pp. 352–359.

[84] J. Z. Kolter and A. Y. Ng, "Regularization and feature selection in least-squares temporal difference learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 521–528.

[85] J. Johns, C. Painter-Wakefield, and R. Parr, "Linear complementarity for regularized policy evaluation and improvement," in *Advances in Neural Information Processing Systems*, J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds. Durham, NC, USA: Duke Univ. Press, 2010, pp. 1009–1017.

[86] M. Geist and B. Scherrer, "ℓ_1-penalized projected Bellman residual," in *Proc. Eur. Workshop Reinforce. Learn.*, 2011, pp. 1–12.

[87] M. W. Hoffman, A. Lazaric, M. Ghavamzadeh, and R. Munos, "Regularized least squares temporal difference learning with nested ℓ_2 and ℓ_1 penalization," in *Proc. Eur. Workshop Reinforce. Learn.*, 2011, pp. 1–12.

[88] M. Geist, B. Scherrer, A. Lazaric, and M. Ghavamzadeh, "A Dantzig selector approach to temporal difference learning," in *Proc. Int. Conf. Mach. Learn.*, 2012, pp. 1–8.

[89] I. Menache, S. Mannor, and N. Shimkin, "Basis function adaptation in temporal difference reinforcement learning," *Ann. Oper. Res.*, vol. 134, no. 1, pp. 215–238, 2005.

[90] P. W. Keller, S. Mannor, and D. Precup, "Automatic basis function construction for approximate dynamic programming and reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2006, pp. 449–456.

[91] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman, "Analyzing feature generation for value-function approximation," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 1–8.

[92] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman, "An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 752–759.

[93] J. Wu and R. Givan, "Automatic induction of bellman-error features for probabilistic planning," *J. Artif. Intell. Res.*, vol. 38, no. 1, pp. 687–755, 2010.

[94] S. Mahadevan and M. Maggioni, "Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes," *J. Mach. Learn. Res.*, vol. 8, no. 10, pp. 2169–2231, 2007.

[95] D. Bertsekas and D. Castañon, "Adaptive aggregation methods for infinite horizon dynamic programming," *IEEE Trans. Autom. Control*, vol. 34, no. 6, pp. 589–598, Jun. 1989.

[96] S. Singh, T. Jaakkola, and M. Jordan, "Reinforcement learning with soft state aggregation," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 1995, pp. 361–368.

[97] J. Ma and W. B. Powell, "Convergence analysis of kernel-based on-policy approximate policy iteration algorithms for Markov decision processes with continuous, multidimensional states and actions," Dept. Oper. Res. Financial Eng., Princeton Univ., Princeton, NJ, USA, Tech. Rep. 08544, 2010.

[98] A. Barreto, D. Precup, and J. Pineau, "Reinforcement learning using Kernel-based stochastic factorization," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2011.

Matthieu Geist received the M.Sc. degree in electrical engineering and the M.Sc. degree in mathematics from Supélec, France, both in 2006, and the Ph.D. degree in mathematics from the Université Paul Verlaine, Metz, France, in 2009.

He is currently an Assistant Professor with the IMS-MaLIS Research Group, Supélec, which he joined in 2010. From 2007 to 2010, he was a member of the Measure and Control Laboratory (MC Cluster), ArcelorMittal Research, and a member of the CORIDA project-team of INRIA. He has authored or co-authored more than 50 papers in journals and conferences. His current research interests include statistical machine learning, particularly reinforcement learning, and applications to spoken dialogue systems.

Olivier Pietquin (M'01–SM'11) received the M.Sc. degree in electrical engineering and the Ph.D. degree from the Faculty of Engineering, University of Mons (FPMs), Mons, Belgium, in 1999 and 2004, respectively, and the Habilitation à Diriger des Recherches (French Tenure) from the University Paul Sabatier, Toulouse, France, in 2011.

He is currently a Professor with the Ecole Supérieure d'Electricité (Supélec), Metz, France. He joined the TCTS Laboratory, Signal Processing Department, FPMs, in 1999. In 2001, he was a Visiting Researcher with the Speech and Hearing Laboratory, University of Sheffield, Sheffield, U.K. From 2004 to 2005, he was a Marie-Curie Fellow with the Philips Research Laboratory, Aachen, Germany. He has been a Full Member of the UMI 2958 (joint lab with GeorgiaTech and CNRS) since 2010, where he coordinates the computer science departments and is the Head of the Machine Learning and Interactive Systems Group, and from 2007 to 2011, he was a member of the IADI INSERM Research Team that is involved in research on biomedical signal processing. He has authored or co-authored more than 100 papers in journals and conferences. His current research interests include spoken dialog systems evaluation, simulation and automatic optimization, machine learning, and speech and signal processing.

Dr. Pietquin has been of the IEEE Speech and Language Technical Committee since 2010.