Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Automatyki i Informatyki Stosowanej

Paweł Wawrzyński

Intensive Reinforcement Learning

Rozprawa Doktorska w Dziedzinie Informatyki

Promotor Pracy
prof. nzw. dr hab. inż. Andrzej Pacut

Warszawa 2005

Warsaw University of Technology
Department of Electronics and Information Technology
Institute of Control and Computation Engineering

Paweł Wawrzyński

Intensive Reinforcement Learning

Dissertation
Submitted in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Computer Science

Thesis Supervisor
Professor Andrzej Pacut

May 2005

Streszczenie

W niniejszej pracy problem uczenia się przez wzmacnianie jest przedyskutowany w języku statystyki jako zagadnienie estymacji. Zaproponowana zostaje rodzina algorytmów uczenia się przez wzmacnianie które wyznaczają politykę sterowania w procesie obliczeniowym przetwarzającym całą dostępną historię interakcji między sterwnikiem a urządzeniem. Aproksymacja stochastyczna jako mechanizm odpowiedzialny za zbieżność algorytmów uczenia się przez wzmacnianie jest zastąpiona przez estymację opartą na całej próbie. Zaprezentowane badania symulacyjne pokazują, że uzyskane w ten sposób algorytmy są w stanie zidentyfikować parametery nietrywialnych sterowników w ciągu kilkudziesięciu minut sterowania urządzeniem. Czyni je to wielokrotnie wydajniejszymi od istniejących odpowiedników.
Słowa kluczowe: Sztuczna Inteligencja, Uczenie się Maszyn, Uczenie się przez Wzmocnianie, Przybliżone Programowanie Dynamiczne, Sieci neuronowe, Sterowanie adaptacyjne.

Abstract

The Reinforcement Learning (RL) problem is analyzed in this dissertation in the language of statistics as an estimation issue. A family of RL algorithms is introduced. They determine a control policy by processing the entire known history of plant-controller interactions. Stochastic approximation as a mechanism that makes the classical RL algorithms converge is replaced with batch estimation. The experimental study shows that the algorithms obtained are able to identify parameters of nontrivial controllers within a few dozens of minutes of control. This makes them a number of times more efficient than their existing equivalents.
Keywords: Artificial Inteligence, Machine Learning, Reinforcement Learning, Approximate Dynamic Programming, Neural Networks, Adaptive Control.

Acknowledgements

I wish to thank my advisor, Professor Andrzej Pacut for introducing me into the world of science.

Contents

List of Tables

Notational conventions

- Vectors and matrices are denoted with the use of standard mathematical font: x, w, etc.
- For the vector function $f : \mathbb{R}^n \mapsto \mathbb{R}^m$, the matrix of its partial derivatives will be denoted by
$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$
- Parametric aproximators are denoted with the use of wide tilde. E.g. $\widetilde{f}(x; w)$ is an approximator parameterised by the vector w whose input is x.
- Predicates in formulae
$$[\textit{predicate}] = \begin{cases} 1 & \text{if the } \textit{predicate} \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$
e.g. $\mathrm{sign}(x) = -[x < 0] + [x > 0]$.
- Parameterized distributions are denoted by $\varphi(z; \theta)$ where φ is a density (or a probability in the discrete case) and θ is the parameter. The caption $Y \sim \varphi(\cdot\ ; \theta)$ means that the random variable Y is drawn with the use of the distribution $\varphi(\cdot\ ; \theta)$. Let r be a predicate, f be a function, and they both depend on Y. When the probability of $r(Y)$ and the expected value of $f(Y)$ is calculated for $Y \sim \varphi(\cdot\ ; \theta)$, the probability is denoted by $P_\theta(r(Y))$ and the expected value is denoted by $\mathcal{E}_\theta f(Y)$.
- The normal distribution with mean θ and covariance matrix C is denoted by $N(\theta, C)$.
- Control policies (or simply "policies") are denoted by π. If a policy depends on a parameter, it is denoted by $\pi(w)$, where w is the parameter.

Chapter 1

Introduction

Bodies of living organisms constitute very complex mechanical systems. So far, no methods are known that enable to design control mechanisms for artificial systems characterized by such complexity. It is believed that living organisms *learn* with the use of trials and errors to control their bodies. The topic of this dissertation may be understood as biologically inspired methodology of automatic trial and error optimization of control.

From the practical point of view, we deal with the following problem. Suppose there is a controller together with a plant characterized by unknown dynamics. The controller, in order to work properly, has to optimize its parameters. The only way to do so available to the controller is an interaction with the plant, its observation, and analysis of consequences of executed control actions.

Modern methodology of control system design in most cases involves some preliminary knowledge of the controlled plant. One extremal setting is that the dynamics of the controlled plant is known entirely in the sense that its exact motion equations are available and there are no external disturbances. That may require some preliminary tests with the use of the working plant. However, before the design of the control system is begun, all the necessary knowledge about the plant's dynamics is gathered. This setting prevails in the practice of control system design. The field of automatic control provides satisfying tools for a controller design in this case. The designer does not have to engage the plant because everything he would like to know about it and its control he may check in derivations and/or simulations. Whenever he knows less than enough about the plant, the control system must be optimized while it is working. In this case methods of *adaptive control* [2, 40] must be in use.

Usually the designer of the controller knows something about the controlled system. Usually some of its components have been produced by humans. For instance the designer may know motion equations of the system yet he may be uncertain about values of some parameters. In case that their values imply the controller's behavior (the *control policy*), they may be estimated when the plant is working and the control takes place. The estimates determine then the control policy. This methodology is called *indirect adaptive control.* Alternatively, observation of the control process may imply the controller's behavior directly without the model estimation. This methodology of control optimization is called *direct adaptive control.* In this dissertation we deal with direct adaptive control of a plant whose dynamics is initially unknown, yet we suspect that it is too complex to try to build its model.

1.1 Reinforcement Learning

From the point of view of modern engineering the exhibited by natural organisms ability to control motion of their bodies is indeed impressive. Suppose we were to design a control system for a humanoid robot that is to dig pits in the ground with a shovel. Human's skeleton is, from the point of view of robotics, a manipulator whose number of degrees of freedom exceeds one hundred[1]. We are completely unable to provide a model of its motion. In fact, the modeling of a manipulator whose number of degrees of freedom exceeds 10, constitutes an enormous challenge. Furthermore, even if we had the model, we would still have to describe the desired behavior of the robot. In robotics this means that we are to provide trajectories of each of 120 degrees of freedom. It is even difficult to imagine how such a task could be completed. Nevertheless, almost each adult human is able to learn to dig pits in the ground with a shovel.

The biologically inspired methodology of control systems optimization discussed in the presented dissertation is reinforcement learning (RL). We shall understand here RL as follows. Let us consider a *plant.* At each moment it is in a certain *state* that characterizes a current position and dynamics of the plant as well as a current objective of its operation[2]. At each moment a *controller* applies a control stimulus to the plant and receives a numeric *reward* (or *reinforcement*). The closer the plant is to attain its current objective, the bigger the reward is. The goal of controller's reinforcement learning is to find with the use of trials and errors the optimal way of determining control stimuli on the basis of states. Optimality means here that starting from each state the controller may expect to receive the biggest possible sum of future (discounted) rewards.

[1]There are 204 bones, 460 muscles, and over one hundred joints in human body; the number of joints is difficult to provide, it depends on how one defines a single joint.

[2]Note that in control theory the *state* does not include the objective of the plant's operation. Here we understand the *state* as all the current information the controller may need to control the plant.

Barto and Dietterich are perhaps the most appropriate people to explain relations between the field of reinforcement learning and other sciences. In [5] they write: "The term reinforcement comes from studies of animal learning in experimental psychology, where it refers to the occurrence of an event, in the proper relation to a response, that tends to increase the probability that the response will occur again in the same situation. The simplest reinforcement learning algorithms make use of the commonsense idea that if an action is followed by a satisfactory state of affairs, or an improvement in the state of affairs, then the tendency to produce that action is strengthened, that is, reinforced. This is the principle articulated by Thorndike in his famous *Law of Effect* [46]. Instead of the term reinforcement learning, however, psychologists use the terms *instrumental conditioning*, or *operant conditioning*, to refer to experimental situation in which what an animal actually does is a critical factor in determining the occurrence of subsequent events. These situations are said to include *response contingencies*, in contrast to Pavlovian, or classical, conditioning situations in which the animal's responses do not influence subsequent events, at least not those controlled by the experimenter."

It is no wonder that the very idea of designing control systems that adapt like natural "controllers" do is sound and has existed in control systems community for a very long time; see e.g. [39, 21, 55]. However, the field of reinforcement learning [4, 44] seems to be founded outside this community and is driven by the idea of creation of artificial intelligent systems [24, 35, 48] that adapt to their environments. The perceived control engineering problems seem to be of less relevance for RL. This dissertation stands somewhere between these approaches. Our research is driven both by the idea of creation artificial intelligent systems and by perceived control problems.

In fact, reinforcement learning, as well as other forms of direct adaptive control, introduces an approach to control system design that is alternative to the standard, model based methodology. We do not bother with the model of a process, but design a controller that *learns* to control. Potentially, we replace the procedure lasting for months and involving system modeling and identification, building of a simulator, and designing of the controller, with the process of controller learning, which lasts for maybe several hours and involves only human supervision. As far as the author is aware, at the beginning of 21^{st} century there is nobody, who makes money designing control systems that learn to control physical devices with the use of trials and errors. This dissertation belongs to publications intended to bring the moment of appearance of such people nearer.

Obviously several questions emerge. The most important one concerns security. There are some devices, which are expensive, unique and easy to destroy by inappropriate control. Is it possible to "learn" to control such machines? On the other hand, if we think of cheap, mass-produced robots, the destruction of one of them during the controller training may not matter.

1.2 Scope of the work

This dissertation deals with reinforcement learning methods applicable for control system training. The area of reinforcement learning is not homogenous. From variety of approaches, the spirit of the dissertation is closest to the one associated with randomized Actor-Critic algorithms [4, 16, 45, 17].

The theses of the presented dissertation are as follows.

1. A natural theoretic base for prevailing reinforcement learning algorithms is stochastic approximation. Such methods utilize information obtained from each consecutive control step to construct a gradient estimator of a certain global performance index and use it to alter the parameters of the control policy.

   We claim that one may also design an RL algorithm based on batch estimation. It is possible to construct an estimator of the global performance index parameterized by the policy parameters. We claim that optimization of this estimator with respect to the parameters leads to the optimization of the performance.

2. The objective of each reinforcement learning algorithm is to exploit observations of the control process to optimize the control policy. Let us analyze an RL algorithm as a procedure that receives certain input data (observations of a control process) and produces certain output data (parameters of a control policy). All but few RL algorithms are so constructed that they process the input data sequentially, piece by piece.

   We claim that it is possible to design much faster algorithms when the above sequentiality constraint is discarded. Such algorithms may optimize the estimator of the performance index with respect to the policy parameters. They may have properties very similar to those of the methods based on stochastic approximation, yet they exploit available data much more efficiently.

3. We design a family of algorithms that put information of occurring control steps in a database and optimize the control policy in a computational process that is conducted simultaneously with the control process. The introduced algorithms have the form of certain optimization issues. We propose numerical methods to tackle these particular issues.

> We claim that the proposed algorithms are feasible and substantially faster than the comparable ones. When applied to problems of moderate complexity, they are able to optimize the control policy far before the stored data fill in the memory capacity of a typical computer.

This document is organized as follows. Chapter 2 introduces the RL problem. It distinguishes *short-term-memory* and *long-term-memory* RL algorithms depending on whether they process the occurring control steps consecutively or not. The chapter also overviews main approaches in the field of RL. RL methods we introduce here are based on properties of certain probabilistic distributions. These are discussed in Chapter 3. Subsequent chapter discusses existing short-term-memory randomized Actor-Critic algorithms in the language that will be further utilized to present new methods. Chapter 5 constitutes the core of the dissertation. It introduces a basic long-term-memory Actor-Critic algorithm. Subsequent chapter extends the basic method to an entire family of algorithms. Chapter 7 contains an experimental study that compares existing methods to the ones proposed here. Chapter 8 concludes and overviews directions of possible future work.

Chapter 2

The problem of Reinforcement Learning

We put the initial issue of control optimization into the framework of reinforcement learning (RL). The first formulation of RL comes from artificial intelligence (see [44] for exhaustive insight). It was then reformulated in the language of automatic control (e.g. [7]) which is used below.

This chapter has also the intention of presenting the background of methods introduced in the following chapters. There are three trends in RL we are aware of, and we try to borrow from all of them. In the first and probably the most influential approach, RL is viewed as a computer implementation of adaptation observed in natural organisms. It is mainly inspired by biology and psychology. Within the second approach the objective of a learning control system is to identify a model of the controlled plant and determine a policy by means of approximate Dynamic Programming. Within the third approach the RL problem is solved by direct search of the policy space that takes little advantage of the particular features of the problem. We overview compactly each of these approaches. We also refer an interested reader to [7, 44, 38] for deeper insight.

2.1 Problem definition

In the RL framework, a learning controller interacts with a plant over a series of time steps $t = 1, 2, 3, \dots$. At each time t, the controller observes the plant's *state*, $x_t \in \mathcal{X}$, and chooses a control action, $u_t \in \mathcal{U}$, which causes the plant to transition to state x_{t+1}. The controller receives then a *reinforcement* or *reward*, $r_{t+1} \in \mathbb{R}$. The next state and reinforcement depend only on the preceding state and action, and they depend on these in a stochastic manner, namely

$$\begin{aligned} x_{t+1} &\sim P_{x'|x,u}(\cdot|x_t, u_t) \\ r_{t+1} &\sim P_{r|x,u}(\cdot|x_t, u_t) \end{aligned}$$

where $P_{x'|x,u}$ and $P_{r|x,u}$ are certain stationary conditional distributions. The control action selection is governed by a *policy*, π, that determines the probability distribution of controls for each state. The policy maps the state space to the space of probability distributions over $\mathcal{U}$.

$$\pi : \mathcal{X} \mapsto \{P : \mathfrak{B}(\mathcal{U}) \mapsto [0,1]\}$$

where $\mathfrak{B}(\mathcal{U})$ denotes a family of Borel subsets of $\mathcal{U}$. The entire stochastic dependence between the current action and previous ones is carried by the current state. Technically that means that at each moment the action is drawn independently only on the basis of the state.

Let $\gamma \in [0,1)$ be a discount factor. The controller, starting from the state x_t at the moment t receives a *return* equal to the sum of discounted future rewards

$$R_t = \sum_{i\geq 0} \gamma^i r_{t+1+i}$$

For the initial state x and the policy π in use, one defines the *value function* $V^\pi(x)$ as the expected return, namely

$$V^\pi(x) = \mathcal{E}\left(R_t \middle|\, x_t = x, \text{policy} = \pi\right) \tag{2.1}$$

where $\mathcal{E}$ denotes the expectation operator.

Let the first state x_1 be drawn from the distribution P_1. We assume there is a family of policies Π and for the fixed $\pi \in \Pi$, the sequence of states $\{x_t\}$ forms a Markov chain $\{x_t\}$. Suppose $\{x_t\}$ has a stationary distribution $\eta^\pi(\cdot)$. The objective is to find the policy $\pi \in \Pi$ that maximizes the averaged value function

$$\Phi(\pi) = \int_{x\in\mathcal{X}} V^\pi(x)\, \mathrm{d}\eta^\pi(x)$$

provided that $P_{x'|x,u}$ and $P_{r|x,u}$ are initially unknown and it is only possible to observe plant–controller interactions and modify the controller's policy $\pi \in \Pi$.

Note that in artificial intelligence, the controller does not apply control stimuli to the plant, but rather an *agent* interacts with its *environment* by performing *actions*. Its objective is to find a policy that advises the best action in each state, that is the policy that gives the largest $V^\pi(x)$ in each state x.

In automatic control applications of RL, usually $\mathcal{X} = \mathbb{R}^{n_\mathcal{X}}$, $\mathcal{U} = \mathbb{R}^{n_\mathcal{U}}$, and "state" describes the plant's current position and velocity as well as their desired values. In this context "reinforcement" defines negative distance between the current and the desired values. A good control policy is hence the one that enables to follow desired trajectories as close as possible. Through the entire dissertation, $\mathcal{X}$ and $\mathcal{U}$ will have the above meanings unless stated otherwise.

For instance, suppose we look for parameters for a PID controller that keeps a certain variable close to its desired trajectory. Since each such controller is defined by 3 real numbers, $\Pi = \mathbb{R}^3$. If we denote by ϵ the momentary error, the state vector must contain ϵ, $\dot{\epsilon}$, and $\int \epsilon \, \mathrm{d}t$, but may also contain other variables describing position of the entity that the PID controller is to regulate. Consequently, $\mathcal{X} = \mathbb{R}^m$, where $m \geq 3$. The action is equal to a real-valued stimulus applied, hence $\mathcal{U} = \mathbb{R}$. In control theory we usually assume that the regulation takes place in continuous time, however in digital implementation time is discrete. This is also the case here. The control objective is to minimize root-mean-squared error, hence we define the reward as negative squared error, namely $r_{t+1} = -\epsilon^2_{t+1}$.

Let us consider a more advanced example that concerns an n degrees-of-freedom robotic manipulator. The manipulator grabs items at certain positions and carries them to others. State of the manipulator describes positions and velocities of all its degrees of freedom and ultimate position of the manipulator's end effector (where an item is or is carried to). State vector is hence $(2n + 3)$-dimensional. An action determines forces (or torques) applied at each degree of freedom. It is n-dimensional. Reinforcement may be here defined as negative Euclidean distance between the current and desired the position of the end effector. We may enrich this problem in a number of ways. For instance, it is a natural requirement for the plant to reach its goals as economically as possible. In this case we simply subtract a penalty for energy utilization from the reinforcement.

2.2 Short-term-memory vs. long-term-memory solutions

The most popular algorithms of reinforcement learning such as *Q-Learning* [50] and Actor-Critic methods [4, 45, 17] are based on the same generic loop:

1. Observe the plant's state.

2. Suggest an action to perform.

3. Observe consequences of the action, that is the immediate reward and the next state.

4. Basing on information just gathered, adjust controller's parameters with some finite (usually very small) set of operations.

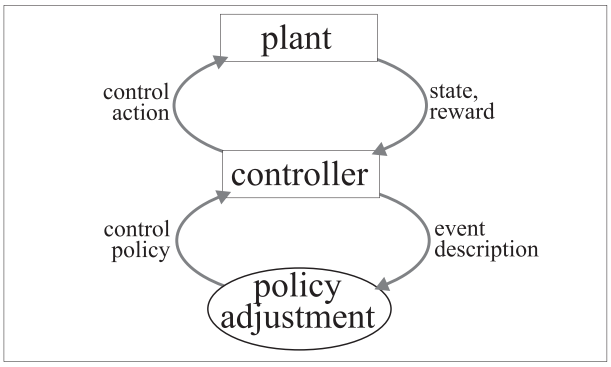

Figure 2.1: Reinforcement Learning with the use of a short-term-memory algorithm.

5. Repeat from Step 1, with the next plant's state.

Such algorithms do not employ any explicit "memory" of past events, at most its compact representation, e.g. the *eligibility trace* [44, 10]. The occurring information must be utilized immediately for a policy adjustment and forgotten.

The algorithms based on the alternative approach "remember" the occurring events and, simultaneously to the control process, build the control policy in some kind of computational process basing on the collected data. The generic method of this kind consists of the two loops different in their nature. The first loop explores plant's dynamics:

1. Observe the plant's state.
2. Suggest an action to perform.
3. Observe consequences of the action, that is the immediate reward and the next state.
4. Add the description of the event to a database. The description encompasses the state, action, reward, next state, and probably some additional information.
5. Repeat from Step 1, with next plant's state.

The second loop takes place simultaneously to the first one. It is a computational process that determines the policy on the basis of the data collected so far. The examples of such methods are Sutton's DYNA architecture [43], and, to a certain extend, the family of Adaptive Critic Designs [31].

For purposes of this dissertation we distinguish **short-term-memory** (Figure 2.1) and **long-term-memory** (Figure 2.2) methods of reinforcement learning, depending on whether they process occurred events consecutively or not [52]. While

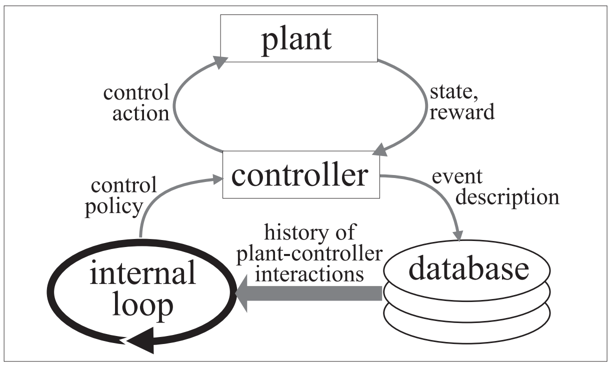

Figure 2.2: Reinforcement Learning with the use of a long-term-memory algorithm.

the former are theoretically based on stochastic approximation, the natural theoretical base for the later seems to be batch estimation.

Let us show the difference between short- and long-term-memory methods using a metaphor. Namely, let us analyze the following problem. We collect samples $X_1, X_2, \dots$ from some unknown scalar distribution. After each i-th sample, we are to provide an approximation m_i of the median of the distribution. One way is to employ stochastic approximation, namely

$$\begin{aligned} m_1 &= X_1 \\ m_i &= m_{i-1} + \beta_{i-1} \operatorname{sign}(X_i - m_{i-1}) \text{ for } i = 2, 3, 4, \dots \end{aligned}$$

where $\{\beta_i, i \in N\}$ is a decreasing sequence which satisfies the standard stochastic approximation conditions, i.e. $\sum_{i\geq 1} \beta_i = +\infty$ and $\sum_{i\geq 1} \beta_i^2 < +\infty$.

Another way is to employ a batch estimator, i.e. to take m_i equal to $\lceil i/2 \rceil$-th highest value among $X_1, \dots, X_i$. Obviously the second way provides better approximations. However, it requires remembering the entire history of sampling and is more computationally expensive.

Although short-term-memory RL methods prevail in the literature, this dissertation will focus mainly on long-term-memory ones. The reason for this is that contemporary computers are powerful enough to keep in memory the entire history of control. There is no justification for any waste of information.

2.3 The approach to RL inspired by biology

Very important ideas associated with reinforcement learning come from Andrew Barto, Richard Sutton, and their associates [44]. What seems to be the most inspiring for this group, are observations of adaptation in living organisms. Their interests concern mainly RL in discrete domains. Consequently, many of their concepts can not be directly, or can not be at all applied to problems tackled in this dissertation, i.e. to RL in continuous domain.

Many of the algorithms presented in the remaining part of this chapter are somehow based on two fundamental functions. The first one is the value function $V^\pi : \mathcal{X} \mapsto \mathbb{R}$, defined in (2.1). Note the following property of V^π:

$$V^\pi(x) = \mathcal{E}\left(r_{t+1} + \gamma V^\pi(x_{t+1}) \Big| x_t = x; \pi \right). \tag{2.2}$$

The *action-value function* $Q^\pi : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$ is typically defined as the expected value of future discounted rewards the controller may expect when starting from a state x, performing an action u, and following the policy π afterwards [50]:

$$Q^\pi(x, u) = \mathcal{E}\left(r_{t+1} + \gamma V^\pi(x_{t+1}) \Big| x_t = x, u_t = u \right). \tag{2.3}$$

2.3.1 The prototype of randomized Actor-Critics

We start our discussion with Barto's and Sutton's algorithm [4, 45], which seems to be the first one employing the methodology called later the reinforcement learning.

Suppose both $\mathcal{X}$ and $\mathcal{U}$ are finite. The method employs two functions, which both are adjusted during the algorithm's work. The first one, the *Actor* $\mu : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$ serves for action selection. Namely, the larger is the value $\mu(x, u)$, the higher is the probability of selection of u in state x. The generic form of the algorithm does not determine this mechanism. The probability of choosing u in x may be for instance equal to

$$P(u|x) = \frac{\exp(\mu(x, u))}{\sum_{u'} \exp(\mu(x, u'))}.$$

This action selection mechanism determines properly a control policy. Let us denote this policy by $\pi(\mu)$. The second function, the *Critic*, $V : \mathcal{X} \mapsto \mathbb{R}$ approximates the value function $V^{\pi(\mu)}$.

The algorithm is presented in Table 2.1. Its step 3 is the key one. Let us denote the *temporal difference*

$$d_t = r_{t+1} + \gamma V(x_{t+1}) - V(x_t).$$

If $d_t > 0$, the algorithm infers that the expected return associated with control u_t is larger than the expected return in state x_t. Consequently, $\mu(x_t, u_t)$ is increased. Conversely, if $d_t < 0$, the action u_t is worse than expected and $\mu(x_t, u_t)$ is decreased. Depending on the sign of d_t, the action u_t is made either more or less likely in the state x_t.

The correctness of this form of the randomized Actor-Critic method has not been proved so far. Moreover, implementation of this algorithm in the case of continuous action is questionable. However, the very idea of increasing the probability

```
0. Set $t := 1$. Initialize $\mu$ and $V$ randomly.
1. Determine the action $u_t$ on the basis of $\mu(x_t, \cdot)$.
2. Perform the action $u_t$, observe the next state $x_{t+1}$ and the reinforcement $r_{t+1}$.
3. Increase $\mu(x_t, u_t)$ by the value proportional to $r_{t+1} + \gamma V(x_{t+1}) - V(x_t)$.
4. Adjust $V(x_t)$ towards $r_{t+1} + \gamma V(x_{t+1})$.
5. Set $t := t + 1$ and repeat from Step 1.
```

Table 2.1: The prototype of randomized Actor-Critics.

of "good" actions and decreasing the probability of "bad" ones probably exists in every reinforcement learning algorithm, as well as it is present in the algorithms introduced in this dissertation.

Randomized Actor-Critic algorithms have evolved [57, 7, 45, 17]. The ideas stemming from biology and psychology reached a mathematically strict meaning. We present one of algorithms of this kind in detail in Chapter 4.

2.3.2 Q-Learning

The Q-Learning algorithm [49] employs only one function $Q : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$. Its objective is to approximate Q^{π^*} where π^* is the optimal policy.

Let $\mathcal{X}$ and $\mathcal{U}$ be finite. Table 2.2 presents the Q-Learning algorithm. It is convergent, provided certain regularity conditions are satisfied [50]. First, the action

```
0. Set $t := 1$. Initialize $Q$ randomly.
1. Determine the action $u_t$ on the basis of $Q(x_t, .)$.
2. Perform the action $u_t$, observe the next state $x_{t+1}$ and the reinforcement $r_{t+1}$.
3. Adjust $Q(x_t, u_t)$ towards $r_{t+1} + \gamma \max_{u'} Q(x_{t+1}, u')$.
4. Set $t := t + 1$ and repeat from Step 1.
```

Table 2.2: The Q-Learning algorithm.

selection in Step 1. should assure that each control in each state is performed infinitely many times. It is usually achieved by a random mechanism giving higher probabilities of selection for controls u associated with larger $Q(x_t, u)$. Second, the adjustment in Step 3. should be implemented as

$$Q(x_t, u_t) := Q(x_t, u_t) + \beta_t \left(r_{t+1} + \gamma \max_{u'} Q(x_{t+1}, u') - Q(x_t, u_t) \right)$$

for $\sum_{t \geq 1} \beta_t = \infty$ and $\sum_{t \geq 1} \beta_t^2 < \infty$. If these are true, then the algorithm makes Q converge to $Q^{\pi *}$ with probability one [50]. There is an incarnation of this algorithm for continuous $\mathcal{X}$ and $\mathcal{U}$ [22], yet its convergence has not been proved.

2.4 The approach inspired by dynamic programming

Adaptive Critic Designs (ACDs) constitute a family of tools for finding an approximate solution of the Dynamic Programming [6] problem. These tools are mainly based on, so-called, *backpropagation through time* [58, 62, 9]. Initially, ACDs were proposed for determining a control policy in a computational process based on a model of a controlled plant. However, possibility of adaptation of these methods for on-line learning control tasks has been also investigated [37].

Each ACD algorithm[1] assumes that an action to perform in a state x is an output of a parametric approximator $\widetilde{A}(x; w_A)$ parameterized by the weights' vector w_A. The structure of the approximator $\widetilde{A}$ is constant and thus, this control policy depends only on w_A. Let denote it by $\pi(w_A)$. Notice that each policy $\pi(w_A)$ is deterministic.

To simplify the below discussion of ACDs, we shall use the following properties. Let $\widetilde{f}(x; w)$ be an approximator parameterized by the vector w. Suppose, there are given x and the vector y whose dimension is equal to the dimension of the output of $\widetilde{f}$. We want to bring $\widetilde{f}(x; w)$ closer to y. Notice that

$$\frac{\mathrm{d}}{\mathrm{d}w} \left\| \widetilde{f}(x; w) - y \right\|^2 = 2 \frac{\mathrm{d}\widetilde{f}(x; w)}{\mathrm{d}w} (\widetilde{f}(x; w) - y).$$

To achieve the goal we may then add

$$\beta \frac{\mathrm{d}\widetilde{f}(x; w)}{\mathrm{d}w} (y - \widetilde{f}(x; w))$$

to the vector w for a small positive constant β. We shall call such a modification of w an *adjustment of $\widetilde{f}(x; w)$ towards y*. Similarly, we *adjust $\widetilde{f}(x; w)$ along Δy*.

[1]The ACDs literature has its standard notation and style which are different than these used in this dissertation. ACDs are presented here atypically enough.

Namely, we exploit the fact that

$$\frac{\mathrm{d}}{\mathrm{d}w}\left\|(\widetilde{f}(x;w^*)+\Delta y)-\widetilde{f}(x;w)\right\|^2 = 2\frac{\mathrm{d}\widetilde{f}(x;w)}{\mathrm{d}w}\Delta y.$$

(w^* is fixed and equal to the current value of w). To achieve the goal we add

$$\beta\frac{\mathrm{d}\widetilde{f}(x;w)}{\mathrm{d}w}\Delta y$$

to the weights' vector w for a small positive constant β.

In order to improve the policy $\pi(w_A)$, each ACD algorithm constructs $\frac{\mathrm{d}}{\mathrm{d}u}Q(x,u)$, an estimate of $\frac{\mathrm{d}}{\mathrm{d}u}Q^{\pi(w_A)}(x,u)|_{u=\widetilde{A}(x;w_A)}$, and modifies $\widetilde{A}(x;w_A)$ along this estimate. This way $\widetilde{A}$ is modified for its output to produce actions that give larger returns. Various algorithms from the family use different forms of this estimate, yet its usage is similar. Namely, in its core activity, each such method uses the estimate for a policy adjustment in one of the following ways:

- Increase $Q(x,\widetilde{A}(x;w_A))$ by modifying w_A along the direction of

$$\frac{\mathrm{d}}{\mathrm{d}w_A}Q(x,\widetilde{A}(x;w_A)) = \frac{\mathrm{d}\widetilde{A}(x;w_A)}{\mathrm{d}w_A}\frac{\mathrm{d}Q(x,u)}{\mathrm{d}u}\bigg|_{u=\widetilde{A}(x;w_A)}.$$

  This is intended to modify the policy such that the expected return the controller may expect is increased. See [31].

- Modify w_A to decrease a distance between $Q(x,\widetilde{A}(x;w_A))$ and some previously assumed numeric objective [37]. This objective should be as large as we hope the expected return should be in the state x.

Since the above possibilities are similar in their spirit and differ only in technical details, in the sequel we shall discuss only the first one.

2.4.1 Heuristic Dynamic Programming

Heuristic Dynamic Programming (HDP) [59, 60, 31] is a methodology of a generation of a control policy rather than a well defined algorithm. This methodology involves observing a working plant, building its model, and computing the policy. Table 2.3 presents probably the most popular version of HDP.

On the highest level, HDP alternates two activities, which differ a lot in their very nature. The first one, the *exploitation loop*, is an observation of the control process. The process may be real or simulated. In fact, the loop serves only for determining which parts of the state space are visited by the plant. The second

```
Initialize $w_A$ randomly.
Repeat until the policy $\pi(w_A)$ is satisfying:
    1. *The exploitation loop.* Set $i = 1$ and repeat $t$ times:
        1.1. Observe $x_i$ and generate the action $u_i = \widetilde{A}(x_i; w_A)$.
        1.2. Apply $u_i$. The plant moves to a next state.
        1.3. Set $i := i + 1$ and repeat from Step 1.1.
    2. *The internal loop.* Basing on the data gathered in the exploitation loop, perform an approximate policy iteration. Namely, repeat until convergence:
        2.1. Policy evaluation. Find
                 $w_V := \arg\min_w \frac{1}{t} \sum_{i=1}^{t} (v_i - \widetilde{V}(x_i; w))^2$
             where $v_i$ is recalculated after every step of the above minimization as
                 $v_i = R(x_i, \widetilde{A}(x_i; w_A)) + \gamma \widetilde{V}(S(x_i, \widetilde{A}(x_i; w_A)); w_V)$
        2.2. Policy improvement. Find
                 $w_A := \arg\max_w \frac{1}{t} \sum_{i=1}^{t} Q_{HDP}(x_i, \widetilde{A}(x_i; w), w_V)$
             where
                 $Q_{HDP}(x, u, w_V) = R(x, u) + \gamma \widetilde{V}(S(x_i, u); w_V)$
```

Table 2.3: Heuristic Dynamic Programming.

activity is a computational process which determines the policy on the basis of the model and information of the visited states. Let us discuss this process in detail.

The algorithm uses two functions that constitute a model of the plant's dynamics, namely the model of rewards

$$R(x,u) = \mathcal{E}(r_{t+1}|x_t = x, u_t = u) \tag{2.4}$$

and the model of next states (the state equation)

$$S(x,u) = \mathcal{E}(x_{t+1}|x_t = x, u_t = u). \tag{2.5}$$

Note that when these functions are initially known, the algorithm is not a reinforcement learning method any more. It does not learn to control from experience, but from a model provided by a human designer. The algorithm may, however, become a RL method when models R and S are built from the information gathered in the exploitation loop. In Appendix A we provide an analysis of the algorithm's behavior in this case. We point there reasons to expect that the version of the HDP algorithm that learns R and S on-line is likely to behave poorly. In Chapter 7 we provide an empirical argument for this claim.

HDP employs an auxiliary approximator $\widetilde{V}(x; w_V)$ parameterized by the weight vector w_V. The objective of $\widetilde{V}$ is to approximate the value function $V^{\pi(w_A)}$. The plant model (2.4) and (2.5) together with the action-value function definition (2.3) and the value function approximation serve to construct an approximation of the action-value function

$$Q_{_{HDP}}(x, u, w_V) = R(x,u) + \gamma \widetilde{V}(S(x,u); w_V).$$

Notice that values of $Q_{_{HDP}}$ are calculated on the basis of R, S, and $\widetilde{V}$ whenever it is necessary.

Step 2. of HDP performs approximate Policy Iteration. Step 2.1. approximates the value function $V^{\pi(w_A)}$ for the fixed policy $\pi(w_A)$ to enable approximation of $Q^{\pi(w_A)}$. Step 2.2. adjusts $\widetilde{A}(x; w_A)$ to maximize the averaged approximation of the action-value function. In this way Step 2.1. evaluates the policy and Step 2.2. optimizes the "first step along each trajectory", as it is usually done in Policy Iteration.

The above version of HDP is so far the first discussed long-term-memory algorithm (see Section 2.2) as it requires keeping in memory an entire history of a control process. The very idea of utilizing the history of control in some kind of approximate Policy Iteration will be used in many algorithms presented in this dissertation.

2.4.2 Action-Dependent HDP

Action-Dependent Heuristic Dynamic Programming [59, 31, 37] is basically a short-term-memory method of reinforcement learning. There are, however, also its long-term-memory versions [20]. The basic form of the algorithm is presented in Table 2.4.

Set $t := 1$. Initialize w_A and w_Q randomly.
Repeat until the policy $\pi(w_A)$ is satisfying:

1. $u_t = \widetilde{A}(x_t; w_A)$.
2. Apply the control stimulus u_t, observe the next state x_{t+1} and the reinforcement r_{t+1}.
3. *Policy improvement.* Adjust $\widetilde{A}(x_t; w_A)$ along $\frac{\mathrm{d}}{\mathrm{d}u_t}\widetilde{Q}(x_t, u_t; w_Q)$, namely
$$w_A := w_A + \beta_t^A \frac{\mathrm{d}}{\mathrm{d}w_A}\widetilde{A}(x_t; w_A)\frac{\mathrm{d}}{\mathrm{d}u_t}\widetilde{Q}(x_t, u_t; w_Q)$$
4. *Policy evaluation.* Adjust $\widetilde{Q}(x_t, u_t; w_Q)$ towards $r_{t+1}+\gamma\widetilde{Q}(x_{t+1}, u_{t+1}; w_Q)$, where $u_{t+1} = \widetilde{A}(x_{t+1}; w_A)$, namely
$$q_t := r_{t+1} + \gamma\widetilde{Q}(x_{t+1}, \widetilde{A}(x_{t+1}; w_A); w_Q)$$
$$w_Q := w_Q + \beta_t^Q \frac{\mathrm{d}\widetilde{Q}(x_t, u_t; w_Q)}{\mathrm{d}w_Q}\Big(q_t - \widetilde{Q}(x_t, u_t; w_Q)\Big)$$
5. Set $t := t + 1$ and repeat from Step 1.

Table 2.4: Action-Dependent HDP.

The algorithm employs the auxiliary approximator $\widetilde{Q}(x, u; w_Q)$ parameterized by the weight vector w_Q. Its objective is to approximate the action-value function $Q^{\pi(w_A)}$. The algorithm simply modifies $\widetilde{A}(x; w_A)$ along $\frac{\mathrm{d}}{\mathrm{d}u}\widetilde{Q}(x, u; w_Q)|_{u=\widetilde{A}(x;w_A)}$ and reapproximates $Q^{\pi(w_A)}$ for the changing policy $\pi(w_A)$.

2.4.3 Dual Heuristic Programming and its modifications

Dual Heuristic Programming (DHP) [59, 54, 31] is, in its spirit, very similar to HDP. Both algorithms maximize the averaged approximation of the Q function. In order to do so, they both use a gradient of the Q function approximation. In HDP, this gradient is calculated from the model and the approximation of the $V^{\pi(w_A)}$ function. In DHP a dedicated approximator is used for a direct calculation of this gradient. The basic form of this algorithm is presented in Table 2.5.

This algorithm employs the auxiliary approximator $\widetilde{\lambda}(x, u; w_\lambda)$ parameterized by the weight vector w_λ. Its objective is to approximate the gradient of the action-value function $\frac{\mathrm{d}}{\mathrm{d}u}Q^{\pi(w_A)}(x, u)$. The algorithm makes $\widetilde{A}(x; w_A)$ approximately ascend $Q^{\pi(w_A)}$ along $\widetilde{\lambda}$ and reapproximates $\frac{\mathrm{d}}{\mathrm{d}u}Q^{\pi(w_A)}$ for the changing policy $\pi(w_A)$.

In its core activity, the DHP algorithm maximizes an approximation of the $Q^{\pi(w_A)}$ function. Here, this approximation is defined as

$$Q_{DHP}(x, u, w_\lambda) = Q^{\pi(w_A)}(x, 0) + \int\limits_{[0,u]} \widetilde{\lambda}(x, u'; w_\lambda)\, \mathrm{d}u'. \tag{2.6}$$

$[0, u]$ is here understood as the multi-dimensional interval $[0, u^1] \times \cdots \times [0, u^{\dim \mathcal{U}}]$, where $u = [u^1, \ldots, u^{\dim \mathcal{U}}]$. Notice that one does not have to calculate the integral (2.6). This is the gradient $\frac{\mathrm{d}}{\mathrm{d}u}Q_{DHP}(x, u, w_\lambda)$ what is needed to maximize Q_{DHP}, whereas this gradient is equal to $\widetilde{\lambda}(x, u; w_\lambda)$.

HDP, ADHDP, and DHP do not constitute all Adaptive Critic Designs. There are several other modifications. Globalized Dual Heuristic Programming may be viewed as a combination of HDP and DHP. It adjusts $\widetilde{A}(x; w_A)$ along the vector that is a weighted average of the HDP and DHP gradients. Algorithms HDP, DHP, and GDHP have their versions modified in the way that the $\widetilde{V}$ approximator is replaced with $\widetilde{Q}$ that approximates $Q^{\pi(w_A)}(x, u)$. The entire family is discussed in a number of publications [59, 60, 23, 30, 31].

2.5 The approach based on agnostic optimization

Let us analyze an issue slightly more general than the RL problem. We want to choose the best policy from among a family Π parameterized by a vector w. Each policy $\pi(w) \in \Pi$ is characterized by a quality measure $\Phi(w)$. We may estimate $\Phi(w)$ by observing the use of the policy $\pi(w)$ for some time. For instance suppose we are to provide parameters for a PID controller. We may check each triple of parameters by observing a control process driven by the triple for some time. In this case it is natural to define $\Phi(w)$ as the negative mean-squared difference between the actual and the desired trajectory of the plant. We can only observe the control process for a while, so the true value of $\Phi(w)$ is unavailable to us, but only its estimate.

We may treat the generic problem as an optimization issue

$$\arg\max_w \Phi(w)$$

```
Initialize $w_A$ randomly.
Repeat until the policy $\pi(w_A)$ is satisfying:
    1. *The exploitation loop.* Set $i = 1$ and repeat $t$ times:
        1.1. Observe $x_i$ and generate the action $u_i = \widetilde{A}(x_i; w_A)$.
        1.2. Perform $u_i$ (the plant moves to the next state).
        1.3. Set $i := i + 1$ and repeat from Step (a).
    2. *The internal loop.* Basing on the data gathered in the exploitation loop, perform an approximate policy iteration. Namely, repeat until convergence:
        2.1. Policy evaluation. Find
             $$w_\lambda := \arg\min_w \frac{1}{t} \sum_{i=1}^{t} \|g_i - \widetilde{\lambda}(x_i, \widetilde{A}(x_i; w_A); w)\|^2$$
             where $g_i$ is recalculated after every step of the above minimization as
             $$g_i = \left.\frac{\mathrm{d}R(x_i, u)}{\mathrm{d}u}\right|_{u=\widetilde{A}(x_i;w_A)} + \gamma \left.\frac{\mathrm{d}S(x_i, u)}{\mathrm{d}u}\right|_{u=\widetilde{A}(x_i;w_A)} \left.\frac{\mathrm{d}\widetilde{V}(x; w_V)}{\mathrm{d}x}\right|_{x=S(x_i,\widetilde{A}(x_i;w_A))}$$
        2.2. Policy improvement. Find
             $$w_A := \arg\max_w \frac{1}{t} \sum_{i=1}^{t} Q_{DHP}(x_i, \widetilde{A}(x_i; w), w_\lambda) \qquad (2.7)$$
             for $Q_{DHP}$ defined in (2.6).
```

Table 2.5: Dual Heuristic Programming.

such that the only available means to solve it encomapass choosing w and drawing an unbiased estimator of $\Phi(w)$. The gradient $\nabla\Phi$ is unavailable because its calculations would have to be based on additional assumptions and these are given up here. We do not take advantage of the structure of the problem which is the reason we called this approach the “agnostic” one.

Genetic algorithms constitute a family of methods useful for solving problems defined in this generic way. Indeed, they have been used for the RL problem [26]. What they try to do is a kind of a global search of Π. There are also methods that are intended to perform a local search. One of these is the Simple Random Search algorithm [33]. Its idea is to draw and evaluate the parameters vector w in the vicinity of the best parameters found so far. It assumes that it is able to check the value of $\Phi(w)$ (instead of its unbiased estimator) within a single control trial for a chosen w.

Let us consider the relation between this approach and the one discussed in Section 2.3. The algorithm presented there were so constructed that they, roughly speaking, for each state, checked each control action and watched what happened. Here, an algorithm checks each possible policy and observes what happens. This “checking and watching what happens” is moved from the level of actions to the level of policies. On one hand, a direct search of the space of policies introduces a lot of redundancy: we do not take advantage of the fact that policies may be similar in the sense that they produce the same actions in the same states. On the other hand, exploitation of this redundancy is difficult; it forces us to use approximators of limited accuracy and we loose optimality. A direct search of the space of policies may be very slow, yet sometimes it is the only method that leads to satisfying results. We may always change parameters of the system and see what happens until the system’s behavior becomes satisfying.

Chapter 3

Smoothly exploring distributions

Reinforcement learning algorithms presented in the remaining of this dissertation share the same mechanism of action selection. Namely, at each state x an action u is drawn from a density $\varphi(\cdot\ ;\theta)$. The density is parameterized by the vector θ whose value is determined by a parametric approximator $\widetilde{\theta}(x; w_\theta)$ with input x and parameters w_θ.

For example, $\varphi(u;\theta)$ can be the normal density with mean θ and constant variance whereas $\widetilde{\theta}(x; w_\theta)$ can be a neural network with input x and weights w_θ. In this example, the output of the network determines a center of the distribution out of which an action is drawn.

The two following sections are devoted to an analysis of φ without references to its usage in reinforcement learning. Let $\mathcal{U} \subset \mathbb{R}^n$, Θ be a bounded and closed subset of $\mathbb{R}^m$, and $\varphi : \mathcal{U} \times \Theta \mapsto \mathbb{R}_+$ be a function which, for each fixed value θ of its second argument, is a density of a certain random vector Y in $\mathbb{R}^n$. Obviously, φ defines a family of distributions in $\mathcal{U}$. Let $f : \mathcal{U} \mapsto \mathbb{R}$ be an unknown function and denote by $\mathcal{E}_\theta f(Y)$ the expected value of $f(Y)$ for Y drawn from the distribution selected from φ by setting the parameter to θ. We look for

$$\arg\max_\theta \mathcal{E}_\theta f(Y)$$

assuming that we are just able to repeatedly draw Y and check $f(Y)$. Notice the simple fact that $f(Y)$ is an unbiased estimator of $\mathcal{E}_\theta f(Y)$.

$\mathcal{E}_\theta f(Y)$ interests us as a function of θ. Let denote this function by $(\mathsf{H}f)(\theta)$, i.e.

$$(\mathsf{H}f)(\theta) = \mathcal{E}_\theta f(Y) = \int\limits_{\mathcal{U}} f(z)\varphi(z;\theta)\,\mathrm{d}z \tag{3.1}$$

Formally speaking, H is an operator defined by the family of distributions φ. It maps each function $f : \mathcal{U} \mapsto \mathbb{R}$ to a certain function $h : \Theta \mapsto \mathbb{R}$ (see Figure 3.1). While we provide φ and H as system designers, f and $\mathsf{H}f$ are something we find in

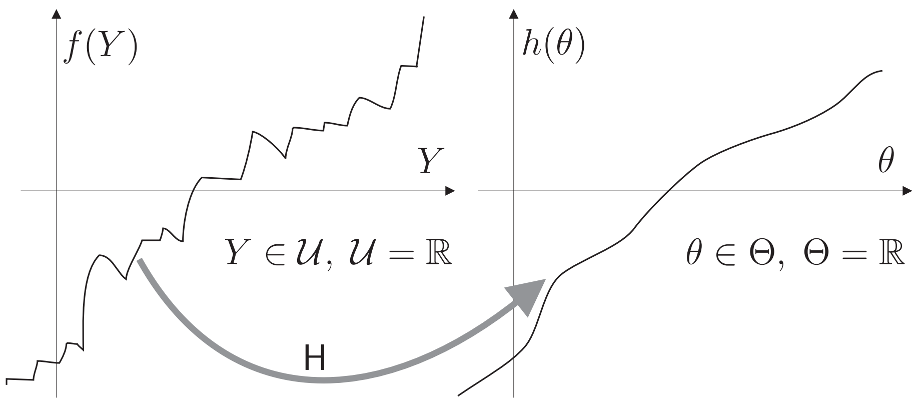
$f(Y)$
$h(\theta)$
Y
θ
$Y \in \mathcal{U},\ \mathcal{U} = \mathbb{R}$
$\theta \in \Theta,\ \Theta = \mathbb{R}$
H

Figure 3.1: An example of $f(Y)$ and $(\mathsf{H}f)(\theta)$. $\mathcal{U} = \Theta = \mathbb{R}$ and $\varphi(\cdot\ ;\theta)$ is the normal density with mean θ and small constant variance. In this case $(\mathsf{H}f)(\theta) = \int_{-\infty}^{\infty} f(z)\varphi(z;\theta)\,\mathrm{d}z$ may be understood as a result of local averaging of f.

the real world. In the remaining of the chapter we shall outline the properties of $\mathsf{H}f$ that enable the maximization of this function. The properties are based on the following assumptions concerning φ.

(A) $\varphi(z;\theta) > 0$ for $z \in \mathcal{U}, \theta \in \Theta$,

(B) for every $z \in \mathcal{U}$, the mapping $\theta \mapsto \varphi(z;\theta)$ is continuous and differentiable,

(C) Fisher Information $I(\theta) = \mathcal{E}_\theta\big(\nabla_\theta \ln\varphi(Y;\theta)\big)\big(\nabla_\theta \ln\varphi(Y;\theta)\big)^T$ is a bounded function of θ,

(D) the hessian $\nabla_\theta^2 \ln\varphi(z;\theta)$ is uniformly bounded.

The families of distributions satisfying the above conditions will be called **smoothly exploring**[1]. For example, the family of normal distributions with constant non-singular variance, parameterized by the expected value, satisfies the conditions. However, the family of normal distributions with constant mean, parameterized by non-zero variance, does not. It does not satisfy Condition (C).

In the discussion below the convention will be held that whenever Y and θ occur in the same formula, Y is drawn from the distribution with density $\varphi(\cdot\ ;\theta)$. All integrals in proofs of propositions are over $\mathcal{U}$.

[1]This name is given just for simplicity. Actually, it is nothing unusual for families of distributions considered in statistics to satisfy conditions (A), (B), (C), and (D).

3.1 Short-term-memory optimization

In this section we will prove that under certain, quite liberal, conditions $\mathsf{H}f$ is uniformly continuous. We show how to estimate its gradient and prove that the variance of the estimator is bounded. This, taken together, enables us to maximize $\mathsf{H}f$ by means of Robbins-Monroe procedure of stochastic approximation [18]. The content of this section is based on [57].

Before we shall begin with continuity, let us prove the following useful lemma.

Lemma 1. *If distributions of Y are smoothly exploring then there exists M such that for each $\theta_1, \theta_2 \in \Theta$ the inequality*

$$\int \big|\varphi(z;\theta_2) - \varphi(z;\theta_1)\big| \, \mathrm{d}z \le M\|\theta_2 - \theta_1\| \tag{3.2}$$

holds.

Proof: The property (C) of smoothly exploring distributions, together with Hölder inequality, implies that $\mathcal{E}_\theta\big\|\nabla_\theta \ln\varphi(Y;\theta)\big\|$ is bounded. Since $\nabla_\theta \ln\varphi(z;\theta) = \frac{\nabla_\theta\varphi(z;\theta)}{\varphi(z;\theta)}$, M_1 exists such that

$$M_1 = \sup_\theta \mathcal{E}_\theta \|\nabla_\theta \ln\varphi(Y;\theta)\| = \sup_\theta \int \|\nabla_\theta\varphi(z;\theta)\| \, \mathrm{d}z.$$

The expression in the integral

$$\int \big|\varphi(z;\theta_2) - \varphi(z;\theta_1)\big| \, \mathrm{d}z$$

can be bounded by a sum of k increments of φ at $\theta_1 + \Delta_i$ and $\theta_1 + \Delta_{i-1}$, $i = 1, \dots, k$, where $\Delta_i = \frac{i}{k}(\theta_2 - \theta_1)$ for some k. Consequently

$$\begin{aligned}
&\int \big|\varphi(z;\theta_2) - \varphi(z;\theta_1)\big| \, \mathrm{d}z \\
&\qquad \le \int \sum_{i=1}^k \Big|\varphi(z;\theta_1+\Delta_i) - \varphi(z;\theta_1+\Delta_{i-1})\Big| \, \mathrm{d}z.
\end{aligned}$$

The inequality takes place for each k, thus it also takes place in the limit, hence

$$\begin{aligned}
&\int \big|\varphi(z;\theta_2) - \varphi(z;\theta_1)\big| \, \mathrm{d}z \\
&\le \int \int_{[\theta_1,\theta_2]} \|\nabla_{\theta'}\varphi(z;\theta')\| \, \mathrm{d}\theta' \, \mathrm{d}z \\
&= \int_{[\theta_1,\theta_2]} \int \|\nabla_{\theta'}\varphi(z;\theta')\| \, \mathrm{d}z \, \mathrm{d}\theta' \\
&\le M_1\|\theta_2 - \theta_1\|
\end{aligned}$$

■
Now we are ready to prove the uniform continuity.

Proposition 1. *If*

(a) f is a bounded function $\mathbb{R}^n \mapsto \mathbb{R}$,

(b) distributions of Y are smoothly exploring,

then $\mathsf{H}f$ is uniformly continuous.

Proof: We will prove that a change in $\mathcal{E}_\theta f(Y)$ caused by an incremental change of θ is bounded linearly in $\Delta\theta$. Assumption (a) guarantees that

$$M_1 = \sup_z |f(z)|$$

exists. We have

$$\begin{aligned}
&|\mathcal{E}_{\theta+\Delta\theta} f(Y) - \mathcal{E}_\theta f(Y)| \\
&= \left| \int f(z)\varphi(z;\theta+\Delta\theta)\,\mathrm{d}z - \int f(z)\varphi(z;\theta)\,\mathrm{d}z \right| \\
&= \left| \int f(z)(\varphi(z;\theta+\Delta\theta) - \varphi(z;\theta))\,\mathrm{d}z \right| \\
&\le M_1 \int \big|\varphi(z;\theta+\Delta\theta) - \varphi(z;\theta)\big|\,\mathrm{d}z.
\end{aligned}$$

According to Lemma 1 there exists M such that the value of the last integral is smaller than $M\|\Delta\theta\|$. ■

The next proposition enables us to design an unbiased estimator of the gradient $\nabla(\mathsf{H}f)(\theta)$.

Proposition 2. *Under the assumptions of Proposition 1, the following relation holds*

$$\nabla(\mathsf{H}f)(\theta) = \mathcal{E}_\theta\Big(f(Y)\nabla_\theta \ln\varphi(Y;\theta)\Big). \tag{3.3}$$

Proof: We directly calculate the i-th coordinate of the gradient

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\theta_i}\mathcal{E}_\theta f(Y) &= \lim_{\delta\to 0}\frac{1}{\delta}\Big(\mathcal{E}_{\theta+\delta e_i} f(Y) - \mathcal{E}_\theta f(Y)\Big) \\
&= \lim_{\delta\to 0}\frac{1}{\delta}\Bigg(\int f(z)\varphi(z;\theta+\delta e_i)\,\mathrm{d}z \\
&\qquad - \int f(z)\varphi(z;\theta)\,\mathrm{d}z\Bigg) \\
&= \lim_{\delta\to 0}\frac{1}{\delta}\int\big(\varphi(z;\theta+\delta e_i) - \varphi(z;\theta)\big)f(z)\,\mathrm{d}z
\end{aligned}$$

where e_i is the versor of the i-th coordinate, i.e. the vector of zeros except the unit at the i-th position. The assumptions of Lebesgue dominated convergence theorem are satisfied, hence

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}\theta_i}\mathcal{E}_\theta f(Y) \\
&\quad= \int \lim_{\delta\to 0}\frac{1}{\delta}\big(\varphi(z;\theta+\delta e_i)-\varphi(z;\theta)\big)f(z)\,\mathrm{d}z \\
&\quad= \int \nabla_{\theta_i}\varphi(z;\theta)f(z)\,\mathrm{d}z \\
&\quad= \int \frac{\nabla_{\theta_i}\varphi(z;\theta)}{\varphi(z;\theta)}f(z)\varphi(z;\theta)\,\mathrm{d}z.
\end{aligned}$$

Since $\frac{\nabla_\theta\varphi(z;\theta)}{\varphi(z;\theta)} = \nabla_\theta \ln\varphi(z;\theta)$, we finally obtain (3.3). ■

Proposition above leads to the main result of this section:

Proposition 3 (Unbiased estimator). *Under the assumptions of Proposition 1, the random vector*

$$\big(f(Y)-c\big)\nabla_\theta \ln\varphi(Y;\theta) \tag{3.4}$$

is an unbiased estimator of $\nabla(\mathsf{H}f)(\theta)$*, regardless of* c.

Proof: According to Proposition 2

$$f(Y)\nabla_\theta \ln\varphi(Y;\theta)$$

is an unbiased estimator of $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathcal{E}_\theta f(Y)$. Obviously, it is also an unbiased estimator of

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\big(\mathcal{E}_\theta f(Y)-c\big) = \frac{\mathrm{d}}{\mathrm{d}\theta}\mathcal{E}_\theta\big(f(Y)-c\big).$$

We take $(f(Y)-c)\nabla_\theta \ln\varphi(Y;\theta)$ as an estimator of $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathcal{E}_\theta(f(Y)-c)$ and consequently as an estimator of $\frac{\mathrm{d}}{\mathrm{d}\theta}\mathcal{E}_\theta f(Y)$. ■

It is not clear how to choose the scalar c to obtain the best, in the sense of variance minimization, estimator of the form (3.4). Note yet that taking c equal to an approximation of $\mathcal{E}_\theta f(Y)$ decreases the absolute value of the differences $f(Y)-c$ and seems to constrain the variance. However, this might not be the optimal solution.

A gradient estimator, in order to be useful in Robbins-Monroe procedure of stochastic approximation, must be unbiased and its variance must be bounded. In the following proposition we prove the latter property.

Proposition 4 (Variance bound existence). *Under the assumptions of Proposition 1, variance of estimator* (3.4) *is bounded.*

```
0. Set $t := 1$. Initialize $\theta_1 \in \Theta$ randomly.
1. Draw:
       $Y_t \sim \varphi(\cdot\,; \theta_t)$
2. Calculate:
       $\bar\theta_{t+1} = \theta_t + \beta^\theta_t\big(f(Y_t) - c_t\big)\nabla_\theta \ln\varphi(Y_t;\theta)|_{\theta=\theta_t}$   (3.5)
       $\theta_{t+1} = \arg\min_{\theta\in\Theta} \|\theta - \bar\theta_{t+1}\|$   (3.6)
       $\bar c_{t+1} = c_t + \beta^c_t(f(Y_t) - c_t)$   (3.7)
       $c_{t+1} = \min\{\max\{-M, \bar c_{t+1}\}, M\}$   (3.8)
3. Set $t := t + 1$ and move to Step 1.
```

Table 3.1: The SHORT-MEM-MAX algorithm of maximization of $(\mathsf{H}f)(\theta)$.

Proof: Let M_1 be a bound of $\|f(z)\|$. We have

$$\begin{aligned}\mathcal{V}_\theta\big(f(Y) - c\big)\nabla_\theta \ln\varphi(Y;\theta) &\le \mathcal{E}_\theta\big(f(Y) - c\big)^2\big(\nabla_\theta \ln\varphi(Y;\theta)\big)\big(\nabla_\theta \ln\varphi(Y;\theta)\big)^T \\ &\le (M_1 + |c|)^2\mathcal{E}_\theta\big(\nabla_\theta \ln\varphi(Y;\theta)\big)\big(\nabla_\theta \ln\varphi(Y;\theta)\big)^T\end{aligned}$$

which is bounded due to Property (C) of smoothly exploring distributions. ■

This section was entirely devoted to develop tools for the algorithm presented in Table 3.1. The purpose of the algorithm is to make the sequence $\{\theta_t, t = 1, 2, \dots\}$ converge to a local maximum of $\mathsf{H}f$. In its core activity (3.5) it modifies θ along an estimator of $\nabla(\mathsf{H}f)$. The role of c_t is to approximate $(\mathsf{H}f)(\theta_t)$ and to be the reference point for the estimator used in (3.5). The sequences $\{\beta^c_t, t \in N\}$ and $\{\beta^\theta_t, t \in N\}$ should satisfy standard stochastic approximation conditions. In (3.8) c_t is bounded to the interval $[-M, M]$ for a certain positive M. In (3.6) the algorithm makes θ_t belong to Θ. The following proposition summarizes conditions of the algorithm's convergence.

Proposition 5 (SHORT-MEM-MAX convergence). *Let assumptions of Proposition 1 hold, the sequence $\{\beta^\theta_t, t = 1, 2, \dots\}$ satisfies the conditions $\sum_{t\ge1}\beta^\theta_t = \infty$ and $\sum_{t\ge1}(\beta^\theta)^2_t < \infty$, and M be some positive constant. The SHORT-MEM-MAX algorithm converges with probability one to a local maximum of $\mathsf{H}f$.*

The **proof** is a consequence of the fact that SHORT-MEM-MAX becomes a Robbins-Monroe procedure of stochastic approximation. The gradient estimator

used in (3.5) is unbiased (Proposition 3) and its variance is bounded (Proposition 4). ■
Notice that the SHORT-MEM-MAX algorithm (i) process each triple $\langle\theta_t, Y_t, f(Y_t)\rangle$ separately and (ii) each triple is analyzed only once. Hence, the algorithm needs a constant amount of memory.

3.2 Long-term-memory optimization

In the previous section we revealed how to maximize $\mathsf{H}f$ with the use of consecutive drawings from $\varphi(\cdot\ ;\theta)$. The algorithm introduced uses each drawing for a single adjustment of θ. Let us assume now that consecutive drawings are in a certain sense expensive, in contrary to computational power and memory, which are cheap. In this section we are looking for a method of $\mathsf{H}f$ maximization that assumes that after each drawing the entire sample is available for an exhaustive processing. The idea is as follows: We derive an estimator $(\widehat{\mathsf{H}f})(\theta)$ of $(\mathsf{H}f)(\theta)$ that is parameterized by θ and based on the entire sample. In each its step the algorithm maximizes this estimator with respect to θ. The maximization yields another value of θ which is then employed to generate a next drawing. In another step the sample that enables to estimate $(\mathsf{H}f)(\theta)$ is thus larger.

3.2.1 Two estimation problems

Suppose we are given the triple $\langle Y_0, \varphi_0(Y_0), f(Y_0)\rangle$ where Y_0 has been drawn from the density $\varphi_0(\cdot)$. We consider two problems. We want to estimate $\mathcal{E}_\theta f(Y)$ for a given θ and to estimate $\mathcal{E}_\theta(f(Y)-c)^2$ for a given θ and c.

The quantity

$$f(Y_0)\frac{\varphi(Y_0,\theta)}{\varphi_0(Y_0)} \tag{3.9}$$

is an unbiased estimator of $\mathcal{E}_\theta f(Y)$, since

$$\begin{aligned}\mathcal{E}_0\left(f(Y)\frac{\varphi(Y;\theta)}{\varphi_0(Y)}\right) &= \int f(z)\frac{\varphi(z;\theta)}{\varphi_0(z)}\varphi_0(z)\,\mathrm{d}z \\ &= \int f(z)\varphi(z;\theta)\,\mathrm{d}z \\ &= \mathcal{E}_\theta f(Y) \end{aligned} \tag{3.10}$$

where $\mathcal{E}_0$ is the expected value for the random vector Y drawn from the density φ_0. The estimator[2] (3.9) is unbiased, yet its variance can be large when the distributions

[2]This method of constructing estimators is known in statistics as *importance sampling* [34] and has already been used in reinforcement learning; see [28, 29, 36].

$\varphi(\cdot\,;\theta)$ and $\varphi_0(\cdot)$ differ too much. Bounding the variance is an issue of importance. A derivation similar to (3.10) proves that

$$c + (f(Y_0) - c)\frac{\varphi(Y_0, \theta)}{\varphi_0(Y_0)} \tag{3.11}$$

is also an unbiased estimator of $\mathcal{E}_\theta f(Y)$ for each constant (non-random) reference point c. The question arises, how to choose the reference point. It seems reasonable to take c equal to some preliminary assessment of $\mathcal{E}_\theta f(Y)$. If this assessment is correct, it bounds the absolute value of $f(Y_0) - c$ for large values of the fraction $\varphi(Y_0, \theta)/\varphi_0(Y_0)$ and consequently confines variance of (3.11). Another method of suppressing the variance is to replace the division of densities with

$$\rho(\varphi(Y_0;\theta), \varphi_0(Y_0)) = \min\left\{\frac{\varphi(Y_0, \theta)}{\varphi_0(Y_0)}, b\right\} \tag{3.12}$$

where b is a number greater than 1. The larger the number, the smaller the bias yet the larger variance. We obtain the estimator

$$c + (f(Y_0) - c)\rho(\varphi(Y_0;\theta), \varphi_0(Y_0)). \tag{3.13}$$

It is biased, yet it possesses a property that is very useful for our purposes. Namely, its bias vanishes when $\varphi(\cdot\,;\theta)$ approaches $\varphi_0(\cdot)$. We will show it in Proposition 8.

The second problem is to estimate $\mathcal{E}_\theta(f(Y) - c)^2$ for given θ and c. By applying an argument similar to (3.10), we obtain that

$$(f(Y_0) - c)^2\frac{\varphi(Y_0, \theta)}{\varphi_0(Y_0)} \tag{3.14}$$

is an estimator we look for, and it is unbiased. Another estimator

$$(f(Y_0) - c)^2\rho(\varphi(Y_0, \theta), \varphi_0(Y_0)) \tag{3.15}$$

has smaller variance. Once again, the estimator is biased and becomes unbiased for $\varphi(\cdot\,;\theta)$ approaching $\varphi_0(\cdot)$.

3.2.2 The LONG-MEM-MAX algorithm

Solutions of the two problems considered above lead to the LONG-MEM-MAX algorithm that implements the idea mentioned at the beginning of this section. The algorithm is outlined in Table 3.2. It uses each drawing to calculate an estimator of $(\mathsf{H}f)(\theta)$ parameterized by θ. An average of these constitute the estimator $\widehat{H}_t^k(\theta)$ that is maximized with respect to θ to obtain θ^{k+1} (3.17), (3.18). In order to construct $\widehat{H}_t^k(\theta)$, a reference point is required. As suggested above, the reference

```
0. Set $t := 1$. Initialize $\theta_1 \in \Theta$ randomly.
1. Draw:
       $Y_t \sim \varphi(\cdot\,;\theta_t)$
   and set $\varphi_t := \varphi(Y_t;\theta_t)$.
2. Set $k = 1$, $\theta^1 = \theta_t$, and $\theta_{t+1}$ as a limit $\theta^k$ of the following internal loop:
       $c^k := \arg\min_{c\in\mathbb{R}} \frac{1}{t}\sum_{i=1}^{t}(f(Y_i)-c)^2\rho(\varphi(Y_i;\theta^k),\varphi_i)$   (3.16)
       $\theta^{k+1} := \arg\max_{\theta\in\Theta} \widehat{H}_t^k(\theta)$   (3.17)
           where $\widehat{H}_t^k(\theta) = \frac{1}{t}\sum_{i=1}^{t} c^k + (f(Y_i)-c^k)\rho(\varphi(Y_i;\theta),\varphi_i)$   (3.18)
       $k := k+1$
3. Set $t := t+1$ and move to Step 1.
```

Table 3.2: The LONG-MEM-MAX algorithm of maximization of $(\mathsf{H}f)(\theta)$.

point is picked as an estimator of $(\mathsf{H}f)(\theta^k)$ that is as the value c^k that minimizes an estimator of $\mathcal{E}_{\theta^k}(f(Y)-c)^2$ (3.16). Because the reference point changes when the maximization is done, the estimation and maximization has to be repeated. Both processes are repeatedly executed until convergence.

It is worth remembering that there is a similarity between this procedure and the maximum likelihood estimation. In the former, we look for the parameter that maximizes the probability of generating the available data. Here, we look for the parameters most plausible to generate the data we would like to draw.

Implementation of LONG-MEM-MAX may seem difficult because of the infinite internal loop in its Step 2. The following property reduces the loop to a simple optimization issue.

Proposition 6 (LONG-MEM-MAX property). *Let assumptions of Proposition 1 hold. After each step of the LONG-MEM-MAX algorithm the equality*

$$\theta_{t+1} = \arg\max_{\theta} \widehat{H}_t(\theta) \tag{3.19}$$

holds where

$$\widehat{H}_t(\theta) = \frac{\sum_{i=1}^{t} f(Y_i)\rho(\varphi(Y_i;\theta),\varphi_i)}{\sum_{i=1}^{t}\rho(\varphi(Y_i;\theta),\varphi_i)}. \tag{3.20}$$

Proof: First note that the optimization (3.16) is a minimization of a quadratic function. Its solution is well defined as

$$c^k = \frac{\sum_{i=1}^t f(Y_i)\rho(\varphi(Y_i;\theta^k),\varphi_i)}{\sum_{i=1}^t \rho(\varphi(Y_i;\theta^k),\varphi_i)}.$$

We see that $c^k = \widehat{H}_t(\theta^k)$. The function optimized in 3.17 satisfies

$$\begin{aligned}
\widehat{H}_t^k(\theta) &= c^k + \frac{1}{t}\sum_{i=1}^t f(Y_i)\rho(\varphi(Y_i;\theta),\varphi_i) - c^k\frac{1}{t}\sum_{i=1}^t \rho(\varphi(Y_i;\theta),\varphi_i)\\
&= c^k + \left(\frac{\sum_{i=1}^t f(Y_i)\rho(\varphi(Y_i;\theta),\varphi_i)}{\sum_{i=1}^t \rho(\varphi(Y_i;\theta),\varphi_i)} - c^k\right)\frac{1}{t}\sum_{i=1}^t \rho(\varphi(Y_i;\theta),\varphi_i)\\
&= c^k + \left(\widehat{H}_t(\theta) - c^k\right)\frac{1}{t}\sum_{i=1}^t \rho(\varphi(Y_i;\theta),\varphi_i)\\
&= \widehat{H}_t(\theta^k) + \left(\widehat{H}_t(\theta) - \widehat{H}_t(\theta^k)\right) g_t(\theta) \qquad (3.21)
\end{aligned}$$

where

$$g_t(\theta) = \frac{1}{t}\sum_{i=1}^t \rho(\varphi(Y_i;\theta),\varphi_i).$$

Let us fix t. Since (i) Θ is bounded and closed, (ii) g_t is positive, (iii) g_t is continuous, there exist $m, M > 0$ such that for all $\theta \in \Theta$, $g_t(\theta) \in [m, M]$. Let θ^* be one of the vectors such that $\widehat{H}_t(\theta^*) = \max_\theta \widehat{H}_t(\theta)$. The fact that θ^{k+1} maximizes $\widehat{H}_t^k$ implies that

$$\widehat{H}_t^k(\theta^{k+1}) \geq \widehat{H}_t^k(\theta^*),$$

which, together with (3.21), gives

$$\begin{aligned}
\widehat{H}_t(\theta^k) + \left(\widehat{H}_t(\theta^{k+1}) - \widehat{H}_t(\theta^k)\right) g_t(\theta^{k+1}) &\geq \widehat{H}_t(\theta^k) + \left(\widehat{H}_t(\theta^*) - \widehat{H}_t(\theta^k)\right) g_t(\theta^*)\\
\left(\widehat{H}_t(\theta^{k+1}) - \widehat{H}_t(\theta^k)\right) &\geq \left(\widehat{H}_t(\theta^*) - \widehat{H}_t(\theta^k)\right)\frac{g_t(\theta^*)}{g_t(\theta^{k+1})}\\
\widehat{H}_t(\theta^*) - \left(\widehat{H}_t(\theta^{k+1}) - \widehat{H}_t(\theta^k)\right) &\leq \widehat{H}_t(\theta^*) - \left(\widehat{H}_t(\theta^*) - \widehat{H}_t(\theta^k)\right)\frac{g_t(\theta^*)}{g_t(\theta^{k+1})}\\
\widehat{H}_t(\theta^*) - \widehat{H}_t(\theta^{k+1}) &\leq \left(\widehat{H}_t(\theta^*) - \widehat{H}_t(\theta^k)\right)\left(1 - \frac{g_t(\theta^*)}{g_t(\theta^{k+1})}\right)\\
\widehat{H}_t(\theta^*) - \widehat{H}_t(\theta^{k+1}) &\leq \left(\widehat{H}_t(\theta^*) - \widehat{H}_t(\theta^k)\right)\left(1 - \frac{m}{M}\right).
\end{aligned}$$

Consequently, the difference $[\max_\theta \widehat{H}_t(\theta) - \widehat{H}_t(\theta^k)]$ decreases exponentially with growing k. ■

3.2.3 Limit properties of estimators

Even though we do not prove convergence of the LONG-MEM-MAX algorithm, we provide some idea about its limit behavior. At each t-th moment estimators of $(\mathsf{H}f)(\theta)$ for each θ are available to the algorithm. Because of usage of the ρ function in their construction, these estimators are in general biased. However, the closer is θ to vectors $\theta_i, i \in \{1, \dots, t\}$ available in the database, the smaller is the bias of $(\widehat{\mathsf{H}f})(\theta)$. At t-th moment the algorithm draws with the use of the parameter θ_t that is "suspected" of being in the maximum of $\mathsf{H}f$. The drawing decreases biases of $(\widehat{\mathsf{H}f})(\theta)$ for all θ that are close to θ_t. Furthermore, when the algorithm draws repeatedly in some "suspected" area, the bias of estimates of $\mathsf{H}f$ in this area decreases and these estimates converge to appropriate values.

The bias of estimators is a consequence of the usage of the ρ function in their construction. In the sequel we shall assume that ρ is defined as (3.12). Suppose the sample is available that has been drawn with the use of the parameter θ_0. We want to use this drawing to compute an estimation for a given θ. The following proposition determines bounds for the probability (in the distribution $\varphi(\cdot;\theta)$) that the ρ function would have reached its upper limit (which implies the bias). Actually we know the drawn value and we may calculate the value of ρ to determine whether it reaches its upper bound. However, we want to determine how frequent such an event is.

Proposition 7. *Let φ be a family of smoothly exploring distributions, ρ be defined in* (3.12)*. There exists M such that for each $\theta_0, \theta \in \Theta$ the inequality*

$$P_\theta(\rho(\varphi(Y;\theta), \varphi(Y;\theta_0)) = b) \le M\|\theta - \theta_0\|^2 \tag{3.22}$$

holds for $b > 1$.

Proof: We have

$$\begin{aligned}
&\rho(\varphi(Y;\theta), \varphi(Y;\theta_0)) = b > 1 \\
&\Leftrightarrow \varphi(Y;\theta) \ge b\varphi(Y;\theta_0) \\
&\Leftrightarrow \frac{\varphi(Y;\theta)}{\varphi(Y;\theta_0)} \ge b \\
&\Leftrightarrow \ln \frac{\varphi(Y;\theta)}{\varphi(Y;\theta_0)} \ge \ln b \\
&\Leftrightarrow (\ln b)^{-1}\big(\ln \varphi(Y;\theta) - \ln(Y;\theta_0)\big) \ge 1.
\end{aligned}$$

Consequently

$$\begin{aligned}
&P_\theta(\rho(\varphi(Y;\theta),\varphi(Y;\theta_0)) = b)\\
&= P_\theta\left((\ln b)^{-1}\big(\ln\varphi(Y;\theta) - \ln(Y;\theta_0)\big) \geq 1\right)\\
&= \int \left[(\ln b)^{-1}\big(\ln\varphi(z;\theta) - \ln\varphi(z;\theta_0)\big) \geq 1\right] \varphi(z;\theta)\,\mathrm{d}z\\
&= \int \left[(\ln b)^{-2}\big(\ln\varphi(z;\theta) - \ln\varphi(z;\theta_0)\big)^2 \geq 1\right] \varphi(z;\theta)\,\mathrm{d}z\\
&\leq \int (\ln b)^{-2}\big(\ln\varphi(z;\theta) - \ln\varphi(z;\theta_0)\big)^2 \varphi(z;\theta)\,\mathrm{d}z
\end{aligned}$$

where $[\cdot]$ has the conventional meaning. Obviously $[x \geq 1] \leq x$. Because of Property (D) of smoothly exploring distributions there exists $M_1 : (\forall z,\theta)\|\nabla^2_\theta \ln(z;\theta)\| < M_1$ and we have

$$\begin{aligned}
&P_\theta(\rho(\varphi(Y;\theta),\varphi(Y;\theta_0)) = b)\\
&\leq (\ln b)^{-2}\int \big(\|\nabla_\theta \ln\varphi(z;\theta)\|\|\theta_0 - \theta\| + M_1\|\theta_0-\theta\|^2\big)^2 \varphi(z;\theta)\,\mathrm{d}z\\
&\leq \|\theta_0-\theta\|^2(\ln b)^{-2}\int \|\nabla_\theta \ln\varphi(z;\theta)\|^2\varphi(z;\theta)\,\mathrm{d}z \qquad (3.23)\\
&\quad + \|\theta_0-\theta\|^3 2M_1(\ln b)^{-2}\int \|\nabla_\theta \ln\varphi(z;\theta)\|\varphi(z;\theta)\,\mathrm{d}z \qquad (3.24)\\
&\quad + \|\theta_0-\theta\|^4 M_1^2(\ln b)^{-2}\int \varphi(z;\theta)\,\mathrm{d}z. \qquad (3.25)
\end{aligned}$$

Thank to Property (C) of smoothly exploring distributions the integrals (3.23) and (3.24) are bounded. The integral (3.25) is equal to 1. From inequality (3.23) we may only conclude, that the required probability is no greater than $M\|\theta_0-\theta\|^2$ for $\|\theta_0-\theta\|$ small enough. However, this probability is obviously bounded also by 1 and this way bounded by $M\|\theta_0-\theta\|^2$ for all θ_0,θ, and a certain M. ■

Let denote

$$\widehat{h}(\theta, Y, \theta_0, c) = c + \big(f(Y) - c\big)\rho\big(\varphi(Y;\theta),\varphi(Y;\theta_0)\big) \qquad (3.26)$$

where ρ is defined in (3.12). LONG-MEM-MAX averages $\widehat{h}$-s to estimate $(\mathsf{H}f)(\theta)$ for a given θ. The following property of $\widehat{h}$ is the key one. It states that when we draw from $\varphi(\cdot;\theta_0)$, $\widehat{h}(\theta,Y,\theta_0,c)$ becomes the estimator of $(\mathsf{H}f)(\theta)$ whose bias is bounded by $M\|\theta-\theta_0\|^2$ for a certain constant M.

Proposition 8. *Let assumptions of Proposition 7 hold, f, c, and ρ be bounded. There exist M_1 and M_2 such that for each $\theta_0,\theta \in \Theta$ the inequalities*

$$\begin{aligned}
&a) \quad \mathcal{V}_{\theta_0}\widehat{h}(\theta,Y,\theta_0,c) \leq M_1\\
&b) \quad \left|\mathcal{E}_{\theta_0}\widehat{h}(\theta,Y,\theta_0,c) - (\mathsf{H}f)(\theta)\right| \leq M_2\|\theta_0-\theta\|^2
\end{aligned}$$

hold.

Proof: Let M_3 be the upper bound of $|f|$ and $|c|$ and $b > 1$ be the upper bound of ρ. We have
a)

$$\begin{aligned}
&\mathcal{V}_{\theta_0}\widehat{h}(\theta, Y, \theta_0, c) \\
&\le \mathcal{E}_{\theta_0}\big(\widehat{h}(\theta, Y, \theta_0, c)\big)^2 \\
&= \mathcal{E}_{\theta_0}\big(c + (f(Y) - c)\rho(\varphi(Y;\theta), \varphi(Y;\theta_0))\big)^2 \\
&\le \mathcal{E}_{\theta_0}\big(|c| + 2M_3 b\big)^2 \\
&\le (M_3)^2(2b+1)^2
\end{aligned}$$

We thus obtain $M_1 = (M_3)^2(2b+1)^2$.
b) We split $\mathcal{E}_{\theta_0}\widehat{h}$ and $\mathcal{E}_\theta f$ into pairs of integrals. For the first element of each pair ρ does not reach its upper bound and for the second one it does. The first elements cancel each other out and we derive a bound for the second ones.

$$\begin{aligned}
&\Big|\mathcal{E}_{\theta_0}\widehat{h}(\theta, Y, \theta_0, c) - \mathcal{E}_\theta f(Y)\Big| \\
&= |\mathcal{E}_{\theta_0}\left(c + (f(Y) - c)\rho(\varphi(Y;\theta), \varphi(Y;\theta_0))\right) - \mathcal{E}_\theta f(Y)| \\
&= \Bigg| \int\limits_{z:\varphi(z;\theta)/\varphi(z;\theta_0)<b} \left(c + (f(z) - c)\frac{\varphi(z;\theta)}{\varphi(z;\theta_0)}\right)\varphi(z;\theta_0)\,\mathrm{d}z \\
&\quad + \int\limits_{z:\varphi(z;\theta)/\varphi(z;\theta_0)\ge b} \big(c + (f(z) - c)b\big)\varphi(z;\theta_0)\,\mathrm{d}z \\
&\quad - \int f(z)\varphi(z;\theta)\,\mathrm{d}z\Bigg| \\
&= \Bigg| \int\limits_{z:\varphi(z;\theta)/\varphi(z;\theta_0)\ge b} \Big(\big(c + (f(z) - c)b\big)\varphi(z;\theta_0) - f(z)\varphi(z;\theta)\Big)\,\mathrm{d}z\Bigg| \\
&\le \Bigg| \int\limits_{z:b\varphi(z;\theta_0)\varphi(z;\theta)^{-1}\le 1} \Big(\big(c/b + (f(z) - c)\big)\frac{b\varphi(z;\theta_0)}{\varphi(z;\theta)} - f(z)\Big)\varphi(z;\theta)\,\mathrm{d}z\Bigg| \\
&\le \Bigg| \int\limits_{z:\varphi(z;\theta)/\varphi(z;\theta_0)\ge b} \Big(\big(M_3/b + 2M_3\big) + M_3\Big)\varphi(z;\theta)\,\mathrm{d}z\Bigg| \\
&\le 4M_3 P_\theta(\rho(\varphi(Y;\theta), \varphi(Y;\theta_0)) = b)
\end{aligned}$$

According to Proposition (7), the probability above is bounded by $M_4\|\theta - \theta_0\|^2$ for certain M_4. Consequently, we have $M_2 = 4M_3M_4$ which completes the proof. ■

Another proposition determines what happens when we draw Y with the use of θ_0 and use $\nabla_\theta\widehat{h}(\theta, Y, \theta_0, c)$ as an estimator of $\nabla(\mathsf{H}f)(\theta)$. It states that the variance of this estimator is bounded and its bias is limited by $M\|\theta - \theta_0\|$ for a certain M.

Proposition 9. *Let assumptions of Proposition 7 hold, f, c, and ρ be bounded. There exist M_1 and M_2 such that for each $\theta_0, \theta \in \Theta$ the inequalities*

$$\begin{aligned} &a) \quad \mathcal{V}_{\theta_0} \nabla_\theta \widehat{h}(\theta, Y, \theta_0, c) \le M_1 \\ &b) \quad \left\| \mathcal{E}_{\theta_0} \nabla_\theta \widehat{h}(\theta, Y, \theta_0, c) - \nabla(\mathsf{H}f)(\theta) \right\| \le M_2 \|\theta_0 - \theta\| \end{aligned}$$

hold.

Proof: Let M_3 be the upper bound of $|f|$ and $|c|$ and $b > 1$ be the upper bound of ρ. (3.26) implies that

$$\nabla_\theta \widehat{h}(\theta, Y, \theta_0, c) = (f(Y) - c) \frac{\mathrm{d}\rho(\varphi(Y;\theta), \varphi(Y;\theta_0))}{\mathrm{d}\theta}.$$

a) We utilize the fact that for z such that makes ρ reach its upper bound, $\nabla_\theta \widehat{h}(\theta, z, \theta_0, c) = 0$. We have

$$\begin{aligned} &\mathcal{V}_{\theta_0} \nabla_\theta \widehat{h}(\theta, Y, \theta_0, c) \\ &\le \mathcal{E}_{\theta_0} \left\| \frac{\mathrm{d}\widehat{h}(\theta, Y, \theta_0, c)}{\mathrm{d}\theta} \right\|^2 \\ &= \mathcal{E}_{\theta_0} \left\| (f(Y) - c) \frac{\mathrm{d}\rho(\varphi(Y;\theta), \varphi(Y;\theta_0))}{\mathrm{d}\theta} \right\|^2 \\ &\le (2M_3)^2 \int\limits_{z: \frac{\varphi(z;\theta)}{\varphi(z;\theta_0)} < b} \left\| \frac{\mathrm{d}\varphi(z;\theta)}{\mathrm{d}\theta} \frac{1}{\varphi(z;\theta_0)} \right\|^2 \varphi(z;\theta_0)\,\mathrm{d}z \\ &= (2M_3)^2 \int\limits_{z: \frac{1}{\varphi(z;\theta_0)} < \frac{b}{\varphi(z;\theta)}} \left\| \frac{\mathrm{d}\varphi(z;\theta)}{\mathrm{d}\theta} \right\|^2 \frac{1}{\varphi(z;\theta_0)}\,\mathrm{d}z \\ &\le (2M_3)^2 \int\limits_{z: \frac{1}{\varphi(z;\theta_0)} < \frac{b}{\varphi(z;\theta)}} \left\| \frac{\mathrm{d}\varphi(z;\theta)}{\mathrm{d}\theta} \frac{1}{\varphi(z;\theta)} \right\|^2 b\varphi(z;\theta)\,\mathrm{d}z \\ &\le (2M_3)^2 b \int \|\nabla_\theta \ln \varphi(z;\theta)\|^2 \varphi(z;\theta)\,\mathrm{d}z. \end{aligned}$$

The last integral is bounded by a certain M_4 thank to Property (C) of smoothly exploring distributions. We obtain $M_1 = (2M_3)^2 b M_4$.

Notice that the above inequalities along with Hölder inequality imply also that there exists M_5 such that

$$\mathcal{E}_{\theta_0} \left\| \frac{\mathrm{d}\widehat{h}(\theta, Y, \theta_0, c)}{\mathrm{d}\theta} \right\| \le M_5. \tag{3.27}$$

We will utilize the above property later on.
b) Let denote

$$\begin{aligned} x &= \left\| \mathcal{E}_{\theta_0} \nabla_\theta \widehat{h}(\theta, Y, \theta_0, c) - \nabla(\mathsf{H}f)(\theta) \right\| \\ &= \left\| \mathcal{E}_{\theta_0} \left((f(Y) - c) \frac{\mathrm{d}\rho(\varphi(Y;\theta), \varphi(Y;\theta_0))}{\mathrm{d}\theta} \right) - \frac{\mathrm{d}(\mathsf{H}f)(\theta)}{\mathrm{d}\theta} \right\|. \end{aligned}$$

Thank to the definition of the ρ function and Proposition 3 we have

$$\begin{aligned} x = \Bigg| \int\limits_{z:\varphi(z;\theta)/\varphi(z;\theta_0)<b} & \left((f(z) - c) \frac{\mathrm{d}\varphi(z;\theta)}{\mathrm{d}\theta} \frac{1}{\varphi(z;\theta_0)} \right) \varphi(z;\theta_0)\,\mathrm{d}z \\ & - \int \left((f(z) - c) \frac{\mathrm{d}\varphi(z;\theta)}{\mathrm{d}\theta} \frac{1}{\varphi(z;\theta)} \right) \varphi(z;\theta)\,\mathrm{d}z \Bigg| \end{aligned}$$

and consequently

$$\begin{aligned} x &= \left| \int\limits_{z:\varphi(z;\theta)/\varphi(z;\theta_0)\geq b} \left((f(z) - c) \frac{\mathrm{d}\varphi(z;\theta)}{\mathrm{d}\theta} \frac{1}{\varphi(z;\theta_0)} \right) \varphi(z;\theta_0)\,\mathrm{d}z \right| \\ &\leq \int\limits_{z:\varphi(z;\theta)/\varphi(z;\theta_0)\geq b} 2M_3 \left\| \frac{\mathrm{d}\varphi(z;\theta)}{\mathrm{d}\theta} \right\| \mathrm{d}z \end{aligned}$$

Because of Property (D) of smoothly exploring distributions there exists M_5 such that $(\forall z, \theta) \|\nabla^2_\theta \ln(z;\theta)\| < M_5$ and we have

$$\begin{aligned} & \varphi(z;\theta)/\varphi(z;\theta_0) \geq b \\ & \qquad \Leftrightarrow \ln\varphi(z;\theta) - \ln\varphi(z;\theta_0) \geq \ln b \\ & \qquad \Rightarrow \|\theta - \theta_0\| \|\nabla_\theta \ln\varphi(z;\theta)\| + \|\theta - \theta_0\|^2 M_5 \geq \ln b \\ & \qquad \Leftrightarrow (\ln b)^{-1} \left(\|\theta - \theta_0\| \|\nabla_\theta \ln\varphi(z;\theta)\| + \|\theta - \theta_0\|^2 M_5 \right) \geq 1. \end{aligned}$$

The last above equivalence is a result of the fact that $b > 1$ and thus $\ln b > 0$. We employ below the above implication and the fact that $[x \geq 1] \leq x$. We obtain

$$\begin{aligned} x &\leq 2M_3 \int\limits_{z:\varphi(z;\theta)/\varphi(z;\theta_0)\geq b} \left\| \frac{\mathrm{d}\varphi(z;\theta)}{\mathrm{d}\theta} \right\| \mathrm{d}z \\ &= 2M_3 \int \left\| \frac{\mathrm{d}\varphi(z;\theta)}{\mathrm{d}\theta} \right\| [\varphi(z;\theta)/\varphi(z;\theta_0) \geq b]\,\mathrm{d}z \\ &\leq 2M_3 \int \left\| \frac{\mathrm{d}\varphi(z;\theta)}{\mathrm{d}\theta} \right\| \left[(\ln b)^{-1} \left(\|\theta - \theta_0\| \|\nabla_\theta \ln\varphi(z;\theta)\| + \|\theta - \theta_0\|^2 M_5 \right) \geq 1 \right] \mathrm{d}z \\ &\leq 2M_3 (\ln b)^{-1} \|\theta - \theta_0\| \int \|\nabla_\theta \ln\varphi(z;\theta)\|^2 \varphi(z;\theta)\,\mathrm{d}z \qquad (3.28) \\ &\qquad + 2M_3 (\ln b)^{-1} \|\theta - \theta_0\|^2 M_5 \int \|\nabla_\theta \ln\varphi(z;\theta)\| \varphi(z;\theta)\,\mathrm{d}z. \qquad (3.29) \end{aligned}$$

Thank to Property (C) of smoothly exploring distributions the integrals in (3.28) and (3.29) are bounded. Because of uniform continuity of $\mathsf{H}f$ (Proposition 1), $\nabla(\mathsf{H}f)$ is bounded. However, (3.27) implies that $\mathcal{E}_{\theta_0}\widehat{h}$ is also bounded. Consequently, the required bias is uniformly bounded which completes the proof of condition (b). ∎

The above propositions suggest the following behavior of LONG-MEM-MAX. Whenever the algorithm draws with the use of a parameter θ_t, it improves estimates of $(\mathsf{H}f)(\theta)$ and $\nabla(\mathsf{H}f)(\theta)$ for all θ close to θ_t. As suggested by Proposition 9, the closer θ is to θ_t the larger the improvement is. According to Proposition 6, at each step t the algorithm calculates θ_{t+1} that maximizes $\widehat{H}_t(\cdot)$, an estimation of $(\mathsf{H}f)(\cdot)$ calculated on the basis of the data gathered up to the moment t. Obviously $\nabla\widehat{H}_t(\theta)|_{\theta=\theta_{t+1}} = 0$. When Y_{t+1} is drawn with the use of θ_{t+1}, the gradient $\nabla\widehat{H}_{t+1}(\theta)|_{\theta=\theta_{t+1}}$ is likely to reflect $\nabla(\mathsf{H}f)(\theta)|_{\theta=\theta_{t+1}}$ better than $\nabla\widehat{H}_t(\theta)|_{\theta=\theta_{t+1}}$. This way the algorithm follows a vector that approaches $\nabla(\mathsf{H}f)$ to make the sequence $\{\theta_t\}$ approach a local maximum of $\mathsf{H}f$.

At this point we consider the convergence of LONG-MEM-MAX hypothetical. However, it is based on the estimators whose properties are favorable. Namely, their variances are bounded and their biases vanish when they are generated by converging distributions. Furthermore, we shall see that a reinforcement learning algorithm that is based on LONG-MEM-MAX works very well.

3.2.4 Behavior of LONG-MEM-MAX, distant exploration tendency

In this section we consider several aspects of the behavior of LONG-MEM-MAX when the algorithm is far from convergence. The discussion is based mainly on Proposition 6. First of all, for $t = 1$ the algorithm maximizes

$$\widehat{H}_1(\theta) \equiv f(Y_1).$$

It is not surprising that when we have as few as a single sample, we have no information that enables to distinguish "good" θ from "bad" ones.

When there are more samples yet not very many, the algorithm may still behave in a "strange" way. Let us analyze a simple example of this behavior. Let φ be a family of scalar normal distributions $N(\theta; \sigma^2)$ (3.31) parameterized by the expected value θ. For $t = 2$ the algorithm calculates θ_3 as the value that maximizes

$$\widehat{H}_2(\theta) = \frac{f(Y_1)\rho(\varphi(Y_1;\theta),\varphi_1) + f(Y_2)\rho(\varphi(Y_2;\theta),\varphi_2)}{\rho(\varphi(Y_1;\theta),\varphi_1) + \rho(\varphi(Y_2;\theta),\varphi_2)}.$$

Notice that (a) $\widehat{H}_2(\theta)$ is a weighted average of $f(Y_1)$ and $f(Y_2)$, (b) both weights belong to the interval $(0, 1)$, (c) for $|\theta|$ large enough, the ratio of weighs of $f(Y_1)$

and $f(Y_2)$ is equal to the ratio of densities

$$\begin{aligned}\frac{\rho(\varphi(Y_1;\theta),\varphi_1)}{\rho(\varphi(Y_2;\theta),\varphi_2)} &= \frac{\varphi(Y_1;\theta)\varphi_2}{\varphi(Y_2;\theta)\varphi_1} \\ &= \frac{\exp(-0.5\sigma^{-2}(Y_1-\theta)^2)\varphi_2}{\exp(-0.5\sigma^{-2}(Y_2-\theta)^2)\varphi_1} \\ &= \exp(-0.5\sigma^{-2}((Y_1^2-Y_2^2)-(Y_1-Y_2)\theta))\frac{\varphi_2}{\varphi_1} \\ &= \exp(0.5\sigma^{-2}\theta(Y_1-Y_2))\exp(0.5\sigma^{-2}(Y_2^2-Y_1^2))\frac{\varphi_2}{\varphi_1}. \qquad (3.30)\end{aligned}$$

For instance suppose $Y_1 > Y_2$ and $f(Y_1) > f(Y_2)$. In this case (3.30) implies that the larger θ is, the larger relative weight of $f(Y_1)$ is and thus the larger $\widehat{H}_2(\theta)$ is. Hence, $\widehat{H}_2(\theta)$ is optimized for $\theta = +\infty$.

This illustration may be generalized as follows. Let us consider a set of parameters $S_t = \{\theta_i, i = 1\dots t\}$ that have been employed to generate data up to the moment t. The maximization of $\widehat{H}_t(\theta)$ is likely to lead to θ that gives relatively large $\varphi(Y_i;\theta)$ for i-s such that the values of $f(Y_i)$ are large. If the largest $f(Y_i)$ is associated with θ_i that belongs to the convex hull of S_t, the optimization of $\widehat{H}_t(\theta)$ may lead to θ that is relatively close to θ_i yet as far as possible from the entire set S_t, and this could be infinite θ. In general, if Θ is not bounded, it may happen that $\widehat{H}_t$ is optimized for its infinite argument. Let us call this phenomenon the *distant exploration tendency.* It forces us to bound Θ.

Because of the distant exploration tendency, LONG-MEM-MAX is likely to pick the values of θ from borders of Θ in early stages of its work. At this time consecutive θ_t are far from one another. Afterwards, for t large enough, the tendency vanishes because the choice of θ_{t+1} from outside the convex hull of the set $\{\theta_i, i = 1\dots t\}$ would be equivalent to pick it outside Θ which is impossible. From this moment consecutive θ_t-s are picked from the narrowing area of hypothetical presence of a local maximum of $\mathsf{H}f$. The more narrow the area is the better the estimates of $\mathsf{H}f$ and $\nabla(\mathsf{H}f)$ within this area are.

3.3 Examples of φ

In the two previous sections φ has been treated as an abstract family of probability distributions. We will now discuss the issue of the selection of φ for a control system. Throughout the remaining of the dissertation we will assume that in each state x, a control action is drawn from the density $\varphi(\cdot\,;\theta)$, where the parameter θ is determined by a certain approximator $\widetilde{\theta}$ on the basis of x.

Generally speaking, the choice of the family of densities φ that governs the control selection should depend on what kind of randomization is acceptable, and what kind of exploration seems fruitful for a particular reinforcement learning problem. We present a number of examples illustrating the possible choices.

3.3.1 Normal distribution

A control selection mechanism may be designed as follows. $\mathcal{U} = \Theta = \mathbb{R}^m$ and an action in each state is a sum of a $\widetilde{\theta}$ approximator's output and a zero-mean normal noise. The noise is necessary for optimization of a control policy. In this case φ is a family of normal distributions $N(\theta, C)$ of a constant covariance matrix equal to C, parameterized by the expected value θ:

$$\varphi(u;\theta) = \frac{1}{\sqrt{(2\pi)^m |C|}} \exp\left(-\frac{1}{2}(u-\theta)^T C^{-1}(u-\theta)\right) \tag{3.31}$$

The C parameter determines an amount of randomization in control. If the randomization is large, it results in a deterioration of quality of control. However, the randomization is introduced in order to guarantee an exploration of a set of possible actions. The larger is the randomization, the more efficient is the exploration.

3.3.2 Log-normal distribution

Suppose that the actions are real positive numbers. Suppose also, that varying the actions within some *relative* distance from their optimal values does not affect the performance in a crucial way. Let σ (equal to, say, 0.05) be the required relative accuracy.

In this case we may use $\Theta = \mathbb{R}$ and a family of log-normal distributions

$$\varphi(u;\theta) = \frac{1}{\sigma a\sqrt{2\pi}} \exp\left(-\frac{(\ln u - \theta)^2}{2\sigma^2}\right).$$

To sample according to this density, one may simply take $u = \exp Y$ where Y is drawn from the normal distribution $N(\theta, \sigma^2)$ with the density (3.31).

Extending the discussion above to the case of multidimensional actions and θ is relatively easy.

3.3.3 Finite set of controls

So far the case of discrete actions has not been discussed in this work. The only thing that changes is that φ should be understood as a probability rather than a density.

There is a finite set of actions $\mathcal{U} = \{u^1, u^2, \dots, u^n\}$. We propose the following parameterization. Let θ be a n-element vector and

$$\varphi(u_i;\theta) = \frac{\exp(\theta_i)}{\sum_{j=1}^{n}\exp(\theta_j)}, \quad i = 1,\dots,n$$
$$\theta_i \in (-M, M), \quad i = 1,\dots,n$$

for some positive M. M is the parameter that determines "minimal exploration". The smaller it is, the larger is minimal probability of each action.

An obvious way to parameterize the distributions in discussed $\mathcal{U}$ is to proceed as follows. Let θ be a $n-1$-element vector and we have

$$\varphi(u^i;\theta) = \theta_i, \quad i = 1,\dots,n-1$$
$$\varphi(u^n;\theta) = 1 - \sum_{j=1}^{n-1}\theta_j$$

The drawback of this intuitive parameterization, is that it imposes an asymmetry between u^n and the rest of actions.

3.3.4 Ordered set of controls

Sometimes the controls are discrete, but there is a natural order among them. The following example reveals how to exploit this order.

Let $\mathcal{U} = \{n, n+1, \dots, N\}$ for $n < N$. We define a family of distributions parameterized by a single real value $\theta \in \Theta = [n, N]$ in such a way that it assigns higher probabilities to integers closer to θ. Namely,

$$\varphi(u;\theta) = \frac{\exp\left(-(u-\theta)^2/\sigma^2\right)}{\sum_{j=n}^{N}\exp\left(-(j-\theta)^2/\sigma^2\right)}$$

where σ is a parameter that determines the amount of exploration. Notice that even if the difference $N-n$ is large, for $\sigma \cong 1$ the sum above has very few components greater than the numeric error.

Usually when the action space is defined as above, we may expect that actions that are close to each other bring similar (in a certain sense) consequences. It may be true that a local search of $\mathcal{U}$ induced by the above definition of φ is a better option than independent checking of each action which is induced by the formulation of φ presented in the previous point.

Chapter 4

Short-term-memory Actor-Critic algorithm

In this chapter we discuss the short-term-memory Actor-Critic algorithm presented in [16] in the language that we will use to introduce new methods. The idea of the algorithm has been presented in [4], the main tools for its theoretical understanding were provided in [57]. It obtained its modern form in [16]. Convergence of this type of methods has been analyzed in [45, 17].

4.1 Actor

In Actor-Critic methods [45, 17] Actor is responsible for generation of control actions. Its behavior is a subject of optimization. Usually, Actor is characterized by an action density function $g(\cdot\,; x, w)$ parameterized by a state and some vector $w \in \mathbb{R}^{n_w}$ (e.g., the vector of weights of a neural network). For a known state, this density is employed to generate the action, and the vector w is optimized.

In this dissertation we will use a different representation of the action density we found more useful in derivations. Let $\varphi(\cdot\,;\theta)$ be the action density function parametrized by the vector $\theta \in \Theta \subset \mathbb{R}^m$. Values of θ are determined by a parametric approximator $\widetilde{\theta}(x; w_\theta)$ whose input is a state and parameters are equal to w_θ. We assume that φ satisfies the regularity conditions presented in the previous chapter. The action density depends here on the state indirectly. However, every action density g can be formally represented in this way, just by taking $\widetilde{\theta}(x; w)^T = [x^T w^T]$ and $\varphi(\cdot\,;\theta) = g(\cdot\,;\theta_1, \theta_2)$ where $\theta^T = [\theta_1^T\ \theta_2^T]$.

For example, $\varphi(\cdot\,;\theta)$ may be chosen as the normal density with mean θ and constant variance, whereas $\widetilde{\theta}(x; w_\theta)$ can be represented by a neural network whose input is a state. The network's output determines a center of the action distribution for a given state.

For a fixed φ and a given function $\widetilde{\theta}$, the discussed action selection mechanism forms a policy that depends only on w_θ. We denote this policy by $\pi(w_\theta)$ and assume that for each $\pi(w_\theta)$, the sequence of states $\{x_t\}$ forms a Markov chain which has a steady distribution $\eta^{\pi(w_\theta)}$. The objective of Actor's adaptation is to find the policy $\pi(w_\theta)$ that maximizes the expected sum of future rewards in each state. Formally, we want to maximize the value function (2.1) averaged by the steady state distribution, namely

$$\Phi(w_\theta) = \int\limits_{x\in\mathcal{X}} V^{\pi(w_\theta)}(x)\,\mathrm{d}\eta^{\pi(w_\theta)}(x).$$

The Φ function should be maximized with respect to Actor's parameters w_θ.

4.2 Critic

In general, Actor-Critic algorithms employ some estimators of $\nabla\Phi(w_\theta)$ to maximize $\Phi(w_\theta)$. In order to construct such estimators, an approximator $\widetilde{V}(x; w_V)$ of the value function $V^{\pi(w_\theta)}(x)$ of the current policy $\pi(w_\theta)$ is employed. The approximator (e.g., a neural network) is parameterized by the weight vector w_V which in the policy improvement process should minimize the mean-square error

$$\Psi(w_V, w_\theta) = \int\limits_{x\in\mathcal{X}} \left(V^{\pi(w_\theta)}(x) - \widetilde{V}(x; w_V)\right)^2 \mathrm{d}\eta^{\pi(w_\theta)}(x)$$

In order to explain how $\widetilde{V}$ is used to construct the estimator of $\nabla\Phi$, we need to define two additional functions. The *action-value function* $Q^\pi : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$, mentioned already in (2.3), is typically defined as the expected value of future discounted rewards of the controller starting from a state x, performing an action u, and following a policy π afterwards [50], namely

$$Q^\pi(x, u) = \mathcal{E}\left(r_{t+1} + \gamma V^\pi(x_{t+1})\Big| x_t = x, u_t = u\right). \tag{4.1}$$

We are interested in parameters that govern an action selection rather than in particular actions. Let us define the *pre-action-value function* $U^\pi : \mathcal{X} \times \Theta \mapsto \mathbb{R}$ as the expected value of future discounted rewards the controller may expect starting from a state x, performing an action drawn from the distribution characterized by a parameter θ, and following a policy π afterwards [51], namely

$$\begin{aligned} U^\pi(x, \theta) &= \mathcal{E}\left(r_{t+1} + \gamma V^\pi(x_{t+1})\Big| x_t = x; u_t \sim \varphi(\cdot\,;\theta)\right) \\ &= \mathcal{E}_\theta Q^\pi(x, Y) \end{aligned} \tag{4.2}$$

where by $u \sim \varphi$ we mean that u has the distribution φ, and $\mathcal{E}_\theta$ denotes the expected value calculated for a random vector Y drawn from the distribution $\varphi(\cdot\,;\theta)$. Summing up, the value function defines the expected return when the plant starts from

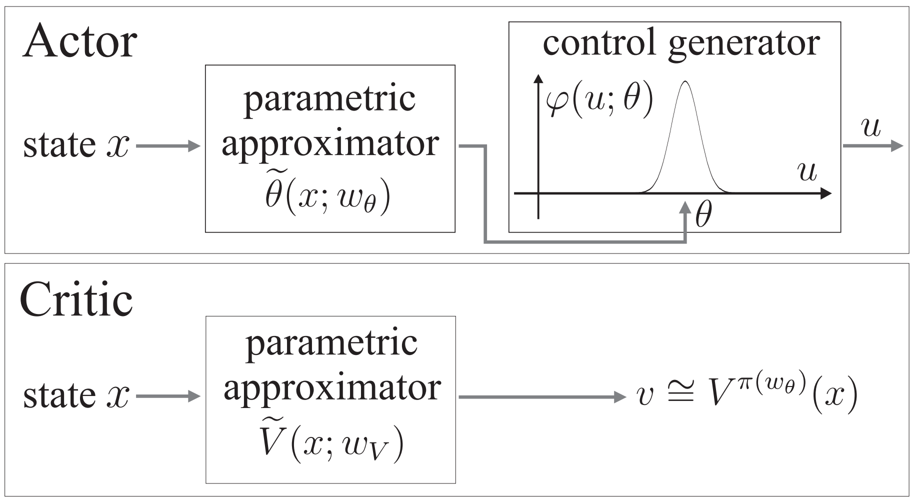
Actor
state x
parametric approximator $\widetilde{\theta}(x; w_\theta)$
control generator
$\varphi(u;\theta)$
u
θ
u
Critic
state x
parametric approximator $\widetilde{V}(x; w_V)$
$v \cong V^{\pi(w_\theta)}(x)$

Figure 4.1: Both components of the Actor-Critic framework.

a given state. The pre-action-value function defines the expected return when the plant starts from a given state and the first control action is distributed with the use of a given parameter. Finally, the action-value function defines the expected return when the plant starts from a given state and the first control action to apply is also given.

Note that, by definition

$$V^{\pi(w_\theta)}(x) = U^{\pi(w_\theta)}\left(x, \widetilde{\theta}(x; w_\theta)\right)$$

because when the policy $\pi(w_\theta)$ is in use, the distribution of the first action in the state x is defined by the parameter θ equal to $\widetilde{\theta}(x; w_\theta)$.

4.3 Actor-Critic interactions

Figure 4.1 depicts both elements of the Actor-Critic framework. Each algorithm based on this framework can be represented as a pair of combined optimization problems. The first one consists in searching for a vector w_θ that maximizes the function

$$\Phi(w_\theta) = \int\limits_{x \in \mathcal{X}} U^{\pi(w_\theta)}\left(x, \widetilde{\theta}(x; w_\theta)\right) \, \mathrm{d}\eta^{\pi(w_\theta)}(x). \tag{4.3}$$

The maximization is executed with the use of the approximation $\widetilde{V}$ of the value function $V^{\pi(w_\theta)}$. This approximation is obtained from the solution of the second optimization problem which is the minimization of

$$\Psi(w_V, w_\theta) = \int\limits_{x\in\mathcal{X}} \Big(U^{\pi(w_\theta)}\big(x, \widetilde{\theta}(x; w_\theta)\big) - \widetilde{V}(x; w_V)\Big)^2 \, \mathrm{d}\eta^{\pi(w_\theta)}(x) \tag{4.4}$$

with respect to w_V for a fixed w_θ.

Algorithms based on the Actor-Critic framework try to find (usually approximate) solutions of the above problems. Although the objectives are similar, the means are very diverse.

Since in the remaining of the dissertation we always consider the policy $\pi(w_\theta)$, we will simplify the notation by writing V^{w_θ}, U^{w_θ}, Q^{w_θ}, and η^{w_θ} in place of $V^{\pi(w_\theta)}$, $U^{\pi(w_\theta)}$, $Q^{\pi(w_\theta)}$, and $\eta^{\pi(w_\theta)}$.

4.4 The RAWC algorithm

The algorithm presented in this chapter does not have a widely spread name. Even though there exist people better authorized to name this method, for conciseness we shall call it Random Actor With Critic (RAWC), as it employs Actor and Critic parts and the policy that the algorithm optimizes is randomized.

The algorithm consists of the following loop.

1. Draw the control u_t
$$u_t \sim \varphi(\cdot\,; \widetilde{\theta}(x_t; w_\theta))$$
2. Perform the action u_t, observe the next state x_{t+1} and the reinforcement r_{t+1}.
3. *Actor adaptation.* Adjust Actor's parameter w_θ along an estimator of a vector that increases Φ. The estimator is calculated on the basis of x_t, u_t, r_{t+1}, and x_{t+1}.
4. *Critic adaptation.* Adjust Critic's parameter w_V along an estimator of a vector that decreases Ψ. The estimator is calculated on the basis of x_t, u_t, r_{t+1}, and x_{t+1}.
5. Set $t := t + 1$ and repeat from Step 1.

In the next two sections we shall derive the estimators mentioned in Points 3 and 4. The derivation will be bases on the properties presented in Section 3.1. To establish the relations between the generic terms of Chapter 3 and our optimization problems, we denote

$$q_t = r_{t+1} + \gamma\widetilde{V}(x_{t+1}; w_V) \tag{4.5}$$

and use the following "dictionary":

1. Y_0 translates into u_t, the drawing,
2. $\varphi(\cdot\,;\theta) \leftrightarrow \varphi(\cdot\,;\widetilde{\theta}(x_t; w_\theta))$, the density used for the drawing,
3. $f(Y_0) \leftrightarrow Q^{w_\theta}(x_t, u_t)$, the expected return; $Q^{w_\theta}(x_t, u_t)$ is estimated by q_t,
4. $(\mathsf{H}f)(\theta) \leftrightarrow U^{w_\theta}(x_t, \widetilde{\theta}(x_t; w_\theta))$, the value to be estimated and maximized,
5. $c \leftrightarrow \widetilde{V}(x_t; w_V)$, the reference point.

4.4.1 Actor adaptation

The idea of a control policy adaptation in RAWC is as follows. At each step t, the control u_t is drawn from $\varphi(\cdot\,;\widetilde{\theta}(x_t; w_\theta))$. The return $q_t = r_{t+1} + \gamma\widetilde{V}(x_{t+1}; w_V)$ is compared with an approximation $\widetilde{V}(x_t; w_V)$ of the expected return. If the applied control u_t turns out to be “good”, i.e. if $q_t > \widetilde{V}(x_t; w_V)$, then $\widetilde{\theta}(x_t; w_\theta)$ is modified to make u_t more likely in state x_t. In the opposite case, $\widetilde{\theta}(x_t; w_\theta)$ is modified to make it less likely.

This intuition translates to maximization of future rewards in the state x_t in the following way. The return expected in the state x_t is equal to $U^{w_\theta}(x_t, \widetilde{\theta}(x_t; w_\theta))$. In order to maximize the expected return $\widetilde{\theta}(x_t; w_\theta)$ is modified along an estimator of $\nabla_\theta U^{w_\theta}(x_t, \theta)|_{\theta=\widetilde{\theta}(x_t;w_\theta)}$.

Let us recall the idea of Policy Iteration. It states that optimization of a control policy can be performed by a sequence modifications of the policy that improve the first action along each trajectory. To implement this idea, let us fix $\pi = \pi(w_\theta)$ and consider a gradient that improves the expected return in a state. Let us define a vector that averages such improvements over all states, namely

$$(T\Phi)(w_\theta) = \int\limits_{x\in\mathcal{X}} \frac{\mathrm{d}}{\mathrm{d}w_\theta} U^\pi\left(x, \widetilde{\theta}(x; w_\theta)\right) \mathrm{d}\eta^\pi(x). \tag{4.6}$$

Properties of Policy Iteration along with (4.3) suggest that

$$(T\Phi)(w_\theta)^T \nabla\Phi(w_\theta) > 0. \tag{4.7}$$

Consequently, (4.7) enables us to maximize Φ with the use of estimators of $(T\Phi)(w_\theta)$.

An estimator of (4.6) is constructed as follows. The current state x_t can be understood as drawn from the stationary distribution $\eta^{w_\theta}(\,\cdot\,)$. The problem of estimation of (4.6) is thus reduced to the problem of estimation of

$$\frac{\mathrm{d}}{\mathrm{d}w_\theta} U^\pi\big(x_t, \widetilde{\theta}(x_t; w_\theta)\big). \tag{4.8}$$

The above gradient is estimated with

$$\frac{\mathrm{d}\widetilde{\theta}(x_t;w_\theta)}{\mathrm{d}w_\theta}\widehat{g}_t$$

where $\widehat{g}_t$, in turn, is an estimator of

$$\left.\frac{\mathrm{d}U^{w_\theta}(x_t,\theta)}{\mathrm{d}\theta}\right|_{\theta=\widetilde{\theta}(x_t;w_\theta)}$$

To construct $\widehat{g}_t$, we utilize the properties discussed in Sec. 3.1. Namely, we treat q_t as an estimate of the return associated with the sample u_t. Consider the expected return as a function of the parameter that determines the sample distribution. The gradient of this function may be estimated in a way similar to (3.4), namely

$$\widehat{g}_t = (q_t - \widetilde{V}(x_t;w_V))\left.\frac{\mathrm{d}\ln\varphi(u_t;\theta)}{\mathrm{d}\theta}\right|_{\theta=\widetilde{\theta}(x_t;w_\theta)} \tag{4.9}$$

where $\widetilde{V}(x_t;w_V)$ is a non-random assessment of q_t, taken as a reference point.

Summing up, we estimate $(T\Phi)(w_\theta)$ with

$$(q_t - \widetilde{V}(x_t;w_V))\frac{\mathrm{d}\widetilde{\theta}(x_t;w_\theta)}{\mathrm{d}w_\theta}\left.\frac{\mathrm{d}\ln\varphi(u_t;\theta)}{\mathrm{d}\theta}\right|_{\theta=\widetilde{\theta}(x_t;w_\theta)} \tag{4.10}$$

which leads us to Actor's adjustment scheme proposed in RAWC, namely

$$\begin{aligned} w_\theta := w_\theta + \beta_t^\theta(r_{t+1} + \gamma\widetilde{V}(x_{t+1};w_V) - \widetilde{V}(x_t;w_V))\times \\ \times\frac{\mathrm{d}\widetilde{\theta}(x_t;w_\theta)}{\mathrm{d}w_\theta}\left.\frac{\mathrm{d}\ln\varphi(u_t;\theta)}{\mathrm{d}\theta}\right|_{\theta=\widetilde{\theta}(x_t;w_\theta)}. \end{aligned} \tag{4.11}$$

For algorithm's convergence the step parameters β_t^θ should satisfy the standard stochastic approximation requirements, namely $\sum_{t\geq 1}\beta_t^\theta = \infty$ and $\sum_{t\geq 1}(\beta_t^\theta)^2 < \infty$.

4.4.2 Critic adaptation

In order to derive Critic's adaptation scheme we apply a similar argument as for Actor. This time our objective is to minimize $\Psi(w_V, w_\theta)$ with respect to w_V. We define

$$(T\Psi)(w_V,w_\theta) = \int\limits_{x\in\mathcal{X}} \frac{\mathrm{d}}{\mathrm{d}w_V}\Big(U^{w_\theta}\big(x_t,\widetilde{\theta}(x_t;w_\theta)\big) - \widetilde{V}(x_t;w_V)\Big)^2\,\mathrm{d}\eta^{w_\theta}(x). \tag{4.12}$$

Next, we treat the current state x_t as drawn from the stationary distribution η^{w_θ} and estimate $(T\Psi)$ with an estimator of

$$\frac{\mathrm{d}}{\mathrm{d}w_V}\Big(U^{w_\theta}\big(x_t,\widetilde{\theta}(x_t;w_\theta)\big) - \widetilde{V}(x_t;w_V)\Big)^2. \tag{4.13}$$

We estimate the above gradient with

$$-2(q_t - \widetilde{V}(x_t; w))\frac{\mathrm{d}\widetilde{V}(x_t; w_V)}{\mathrm{d}w_V}$$

where q_t is in turn an estimator of $U^{w_\theta}\big(x_t, \widetilde{\theta}(x; w_\theta)\big)$, that is the expected return in state x_t.

This leads to the standard [44] Critic adjustment scheme

$$\begin{aligned} w_V := w_V + \beta_t^V (r_{t+1} + \gamma\widetilde{V}(x_{t+1}; w_V) - \widetilde{V}(x_t; w))\times \\ \times \frac{\mathrm{d}\widetilde{V}(x_t; w_V)}{\mathrm{d}w_V}. \end{aligned} \tag{4.14}$$

The step parameters β_t^V must satisfy the same requirements as β_t^θ.

0. Set $t := 1$. Initialize w_θ and w_V randomly.
1. Draw the action u_t
$$u_t \sim \varphi(\cdot\,; \widetilde{\theta}(x_t; w_\theta))$$
2. Perform the action u_t, observe the next state x_{t+1} and the reinforcement r_{t+1}.
3. Calculate the temporal difference as
$$d_t = r_{t+1} + \gamma\widetilde{V}(x_{t+1}; w_V) - \widetilde{V}(x_t; w_V)$$
4. Adjust Actor:
$$w_\theta := w_\theta + \beta_t^\theta d_t \frac{\mathrm{d}\widetilde{\theta}(x_t; w_\theta)}{\mathrm{d}w_\theta}\frac{\mathrm{d}\ln\varphi(u_t; \theta)}{\mathrm{d}\theta}\bigg|_{\theta=\widetilde{\theta}(x_t; w_\theta)} \tag{4.15}$$
5. Adjust Critic:
$$w_V := w_V + \beta_t^V d_t \frac{\mathrm{d}\widetilde{V}(x_t; w_V)}{\mathrm{d}w_V} \tag{4.16}$$
6. Set $t := t + 1$ and repeat from Step 1.

Table 4.1: The RAWC algorithm.

4.5 Implementational issues

Table 4.1 shows the exact formulation of the RAWC algorithm. The question may arise how to easily implement adjustments (4.15) and (4.16). In the case $\widetilde{\theta}$ is a feedforward neural network, the vector

$$d_t \frac{\mathrm{d}\widetilde{\theta}(x_t; w_\theta)}{\mathrm{d}w_\theta} \frac{\mathrm{d}\ln\varphi(u_t;\theta)}{\mathrm{d}\theta}\bigg|_{\theta=\widetilde{\theta}(x_t;w_\theta)} \tag{4.17}$$

may be computed such that the vector

$$d_t \frac{\mathrm{d}\ln\varphi(u_t;\theta)}{\mathrm{d}\theta}\bigg|_{\theta=\widetilde{\theta}(x_t;w_\theta)} \tag{4.18}$$

is *backpropagated through* the network. In case φ is the normal density $N(\theta, C)$ (3.31), the vector (4.18) is equal to

$$d_t C^{-1}(u_t - \widetilde{\theta}(x_t; w_\theta)).$$

If Critic is implemented by a neural network, the gradient used in (4.16) may be simply calculated by backpropagating d_t through the $\widetilde{V}$ network.

4.6 RAWC with λ-estimators of future rewards

Kimura extends in [16] the algorithm discussed above by replacing q_i with an estimator of future rewards whose bias is smaller than that of q_i. For purposes of this dissertation, the algorithm he obtains is called RAWC(λ).

Let us denote

$$q_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \widetilde{V}(x_{t+n}; w_V).$$

RAWC employs $q_t = q_t^{(1)}$ as an estimator of $Q^{\pi(w_\theta)}(x_t, u_t)$. Since $\widetilde{V}$ is an approximation of $V^{\pi(w_\theta)}$ of only limited precision, this estimator is biased. By definition

$$Q^{\pi(w_\theta)}(x, u) = \mathcal{E}(q_t^{(\infty)} | x_t = x, u_t = u, \pi(w_\theta)),$$

and hence $q_t^{(\infty)}$ is an unbiased estimator of $Q^{\pi(w_\theta)}(x_t, u_t)$. Furthermore, the average (over all states) bias of $q_t^{(n)}$ is smaller than the average bias of $q_t^{(m)}$ for $n > m$. To see this, notice that for $n > 1$

$$q_t^{(n)} = r_{t+1} + \gamma q_{t+1}^{(n-1)}. \tag{4.19}$$

Consequently, the bias of $q_t^{(n)}$ may be expressed as a product of γ and the expected, at the moment t, bias of $q_{t+1}^{(n-1)}$. Because $\gamma < 1$, the average bias of $q_t^{(n)}$ decreases as n grows.

It is particularly convenient to use the following λ-*estimator of future rewards* that a weighted average of $q_t^{(n)}$ for all n-s and $\lambda \in [0,1]$:

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)} \tag{4.20}$$

The larger λ, the larger the weight of $q_t^{(n)}$ for large n, the smaller bias but the larger variance. By varying λ we balance the bias and the variance of q_t^λ. Thank to (4.19) we have

$$\begin{aligned} q_t^\lambda &= (1-\lambda)(r_{t+1} + \gamma \widetilde{V}(x_{t+1}; w_V)) + (1-\lambda)\sum_{n=2}^{\infty} \lambda^{n-1}(r_{t+1} + \gamma q_{t+1}^{(n-1)}) \\ &= (1-\lambda)(r_{t+1} + \gamma \widetilde{V}(x_{t+1}; w_V)) + \lambda r_{t+1} + \lambda\gamma q_{t+1}^\lambda \\ &= r_{t+1} + \gamma\left(\lambda q_{t+1}^\lambda + (1-\lambda)\widetilde{V}(x_{t+1}; w_V)\right). \end{aligned} \tag{4.21}$$

The derivation above leads to the following comparison between q_t^λ and q_t. Namely, q_t uses $\widetilde{V}(x_{t+1}; w_V)$ as an assessment of the return gained after step t. Instead, q_t^λ uses a weighted average of $\widetilde{V}(x_{t+1}; w_V)$ and q_{t+1}^λ.

To construct an algorithm similar to RAWC based on q_t^λ as an estimate of future rewards, we must start from (4.21) and notice that

$$q_t^\lambda - \widetilde{V}(x_t; w_V) = r_{t+1} + \gamma \widetilde{V}(x_{t+1}; w_V) - \widetilde{V}(x_t; w_V) + \gamma\lambda(q_{t+1}^\lambda - \widetilde{V}(x_{t+1}; w_V))$$

which leads easily to the equation

$$q_t^\lambda - \widetilde{V}(x_t; w_V) = \sum_{i=0}^{\infty} (\lambda\gamma)^i \left(r_{t+1+i} + \gamma \widetilde{V}(x_{t+1+i}; w_V) - \widetilde{V}(x_{t+i}; w_V)\right).$$

Next, we replace the adjustments (4.11) and (4.14) with

$$\begin{aligned} w_\theta := w_\theta &+ \beta_t^\theta (q_t^\lambda - \widetilde{V}(x_t; w_V)) \times \\ &\times \frac{\mathrm{d}\widetilde{\theta}(x_t; w_\theta)}{\mathrm{d}w_\theta} \frac{\mathrm{d}\ln\varphi(u_t; \theta)}{\mathrm{d}\theta}\bigg|_{\theta = \widetilde{\theta}(x_t; w_\theta)} \end{aligned} \tag{4.22}$$

and

$$\begin{aligned} w_V := w_V &+ \beta_t^V (q_t^\lambda - \widetilde{V}(x_t; w_V)) \times \\ &\times \frac{\mathrm{d}\widetilde{V}(x_t; w_V)}{\mathrm{d}w_V}. \end{aligned} \tag{4.23}$$

This can not be done directly because in order to calculate q_t^λ one should know the entire history of states and rewards that take place after moment t. This history could be infinite. The relation

$$q_t^\lambda - \widetilde{V}(x_t; w_V) = \sum_{i \ge 0} (\gamma\lambda)^i d_{t+i} \tag{4.24}$$

enables to implement (4.22) and (4.23) incrementally. The infinite sequence of adjustments of the form (4.22) produces the following sum

$$\begin{aligned}
&\sum_{t\geq 1}\beta_t^\theta(q_t^\lambda - \widetilde{V}(x_t;w_V))\frac{\mathrm{d}\widetilde{\theta}(x_t;w_\theta)}{\mathrm{d}w_\theta}\frac{\mathrm{d}\ln\varphi(u_t;\theta)}{\mathrm{d}\theta}\bigg|_{\theta=\widetilde{\theta}(x_t;w_\theta)}\\
&=\sum_{t\geq 1}\beta_t^\theta\left(\sum_{i\geq 0}(\gamma\lambda)^i d_{t+i}\right)\frac{\mathrm{d}\widetilde{\theta}(x_t;w_\theta)}{\mathrm{d}w_\theta}\frac{\mathrm{d}\ln\varphi(u_t;\theta)}{\mathrm{d}\theta}\bigg|_{\theta=\widetilde{\theta}(x_t;w_\theta)}\\
&\cong\sum_{t\geq 1}\beta_t^\theta d_t\sum_{i=0}^{t-1}(\gamma\lambda)^i\frac{\mathrm{d}\widetilde{\theta}(x_{t-i};w_\theta)}{\mathrm{d}w_\theta}\frac{\mathrm{d}\ln\varphi(u_{t-i};\theta)}{\mathrm{d}\theta}\bigg|_{\theta=\widetilde{\theta}(x_{t-i};w_\theta)}
\end{aligned}$$

The last of the above equalities is a result of (4.24). It becomes strict for constant β_t^θ, yet because β_t^θ changes very slowly, it is "almost" strict. Similarly, the infinite sequence of adjustments of the form (4.23) produces

$$\sum_{t\geq 1}\beta_t^V(q_t^\lambda - \widetilde{V}(x_t;w_V))\frac{\mathrm{d}\widetilde{V}(x_t;w_V)}{\mathrm{d}w_V}\cong\sum_{t\geq 1}\beta_t^V d_t\sum_{i=0}^{t-1}(\gamma\lambda)^i\frac{\mathrm{d}\widetilde{V}(x_{t-i};w_V)}{\mathrm{d}w_V}$$

The RAWC(λ) algorithm presented in Table 4.2 uses the above approximate equalities to implement the sequence of adjustments of the form (4.22) and (4.23). The algorithm uses auxiliary vectors m_θ and m_V. Their dimensions are equal to the dimensions of w_θ and w_V respectively. Their values at moment t are

$$\begin{aligned}
m_\theta &= \sum_{i=0}^{t-1}(\gamma\lambda)^i\frac{\mathrm{d}\widetilde{\theta}(x_{t-i};w_\theta)}{\mathrm{d}w_\theta}\frac{\mathrm{d}\ln\varphi(u_{t-i};\theta)}{\mathrm{d}\theta}\bigg|_{\theta=\widetilde{\theta}(x_{t-i};w_\theta)}\\
m_V &= \sum_{i=0}^{t-1}(\gamma\lambda)^i\frac{\mathrm{d}\widetilde{V}(x_{t-i};w_V)}{\mathrm{d}w_V}.
\end{aligned}$$

Employing λ-estimators of future rewards is a classical technique of enhancing algorithms in the field of Reinforcement Learning. We use it in the design of methods proposed in Chapter 6. The experimental study presented in Chapter 7 confirms usefulness of this technique.

0. Set $t := 1$, $m_\theta = 0$, and $m_V = 0$. Initialize w_θ and w_V randomly.
1. Draw the action u_t
$$u_t \sim \varphi(\cdot\,; \widetilde{\theta}(x_t; w_\theta))$$
2. Perform the action u_t, observe the next state x_{t+1} and the reinforcement r_{t+1}.
3. Calculate the temporal difference as
$$d_t = r_{t+1} + \gamma \widetilde{V}(x_{t+1}; w_V) - \widetilde{V}(x_t; w_V)$$
4. Adjust Actor:
$$\begin{aligned} m_\theta &:= (\gamma\lambda) m_\theta + \frac{\mathrm{d}\widetilde{\theta}(x_t; w_\theta)}{\mathrm{d}w_\theta} \frac{\mathrm{d}\ln \varphi(u_t; \theta)}{\mathrm{d}\theta}\bigg|_{\theta = \widetilde{\theta}(x_t; w_\theta)} \\ w_\theta &:= w_\theta + \beta_t^\theta d_t m_\theta \end{aligned} \tag{4.25}$$
5. Adjust Critic:
$$\begin{aligned} m_V &:= (\gamma\lambda) m_V + \frac{\mathrm{d}\widetilde{V}(x_t; w_V)}{\mathrm{d}w_V} \\ w_V &:= w_V + \beta_t^V d_t m_V \end{aligned} \tag{4.26}$$
6. Set $t := t + 1$ and repeat from Step 1.

Table 4.2: The RAWC(λ) algorithm.

Chapter 5

Long-term-memory Actor-Critic algorithm

In this chapter we present a long-term-memory Actor-Critic algorithm, the main achievement of the present dissertation. The algorithm was introduced in reference [53]. It is ideologically based on LONG-MEM-MAX presented in Section 3.2. It puts information on occurring control steps in a database and forms a policy in an intensive computation process that goes simultaneously with the control process.

5.1 Structure

The algorithm presented here employs the same Actor and Critic parts as RAWC and RAWC(λ). $\varphi(\cdot\,;\theta)$ is an action density function, where θ is determined by an approximator $\widetilde{\theta}(x; w_\theta)$ parametrized by w_θ. φ and $\widetilde{\theta}$ together define a policy $\pi(w_\theta)$. Another function $\widetilde{V}(x; w_V)$ parametrized by w_V approximates the value function $V^{w_\theta}(x)$. The algorithm consists of two activities performed simultaneously:

1. The exploration of the plant's dynamics by performing actions based on the current policy $\pi(w_\theta)$. Information on consecutive control steps is put into a database.
2. The approximation of Policy Iteration, which implements both elements of Actor-Critic schemes:
   (a) *Policy evaluation* or *Critic training.* Adjustment of w_V to minimize an estimate $\widehat{\Psi}_t(w_V, w_\theta)$ of $\Psi(w_V, w_\theta)$ based on all events up to the current step t.
   (b) *Policy optimization* or *Actor training.* Adjustment of w_θ to maximize an estimate $\widehat{\Phi}_t(w_\theta, w_V)$ of $\Phi(w_\theta)$ based on all events up to the current step t.

The policy employed in Step 1. is the one modified in Step 2. which should be understood as a computationally intensive process basing on an entire history of plant-controller interactions. In order to define this process, suppose that the history is given, namely the states visited $\{x_i, i = 1, \dots, t\}$, the instantaneous rewards received $\{r_{i+1}, i = 1, \dots, t\}$, and the control actions that have been performed, namely $\{u_i, i = 1, \dots, t\}$. The actions were generated by previous incarnations of the policy (which is constantly optimized). They were drawn from densities $\varphi(\cdot\,; \theta_i)$ and the set $\{\varphi_i, i = 1, \dots, t\}$ is given where $\varphi_i = \varphi(u_i; \theta_i)$.

In order to derive $\widehat{\Phi}_t(w_\theta, w_V)$, we treat all the previous states x_i as drawn from $\eta^{w_\theta}(\cdot)$ and estimate the integral (4.3) by the average value

$$\widehat{\Phi}_t(w_\theta, w_V) = \frac{1}{t}\sum_{i=1}^{t} \widehat{U}_i(w_\theta, w_V). \tag{5.1}$$

Here $\widehat{U}_i(w_\theta, w_V)$ is an estimator of $U^{w_\theta}(x_i, \widetilde{\theta}(x_i; w_\theta))$. $\widehat{U}_i$ is constructed with the use of $\widetilde{V}$, and hence both $\widehat{U}_i$ and $\widehat{\Phi}_t$ depend on w_V.

The estimator $\widehat{\Psi}_t$ is built similarly, namely the integral (4.4) is replaced with the average value

$$\widehat{\Psi}_t(w_V, w_\theta) = \frac{1}{t}\sum_{i=1}^{t} \widehat{e_i^2}(w_\theta, w_V) \tag{5.2}$$

where $\widehat{e_i^2}(w_\theta, w_V)$ estimates the squared difference

$$\Big(U^{w_\theta}\big(x_i, \widetilde{\theta}(x_i; w_\theta)\big) - \widetilde{V}(x_i; w_V)\Big)^2.$$

In the next two sections we derive estimators $\widehat{U}_i(w_\theta, w_V)$ and $\widehat{e_i^2}(w_\theta, w_V)$. We heavily utilize the properties discussed in Section 3.2. In order to establish relations between the generic terms of Chapter 3 and our optimization problem, we denote

$$q_i(w_V) = r_{i+1} + \gamma\widetilde{V}(x_{i+1}; w_V) \tag{5.3}$$

and apply the "dictionary" (see the end of Section 4.4 for comparison):

1. Y_0 translates into u_i, the drawing,
2. $\varphi_0(\cdot) \leftrightarrow \varphi(\cdot\,; \widetilde{\theta}(x_i; w_\theta))$, the density used for the drawing,
3. $f(Y_0) \leftrightarrow Q^{w_\theta}(x_i, u_i)$, the expected return; $Q^{w_\theta}(x_i, u_i)$ is estimated by $q_i(w_V)$,
4. $(\mathsf{H}f)(\theta) \leftrightarrow U^{w_\theta}(x_i, \widetilde{\theta}(x_i; w_\theta))$, the value to be estimated and maximized,
5. $c \leftrightarrow \widetilde{V}(x_i; w_V)$, the reference point.

We will also respect the following notational convention. Whenever a certain value is a function of data, it will be denoted by subscript i or t. Accordingly, $q_i(w_V)$ (5.3) is a function of data as well as it is a function of w_V which is mentioned explicitly.

5.2 Policy evaluation

In order to construct $\widehat{\Psi}_i$ (5.2), we specify $\widehat{e_i^2}$ in the form of the *importance sampling* estimator discussed in Section 3.2. In terms of this section, for the drawing u_i, the density φ_i that generated the drawing, and the return $q_i(w_V) = r_{i+1} + \gamma\widetilde{V}(x_{i+1}; w_V)$ are available. We also know the distribution that would currently generate drawings. This distribution is defined by the parameter $\widetilde{\theta}(x_i; w_\theta)$. What we want is to estimate the expected value of the squared difference between the return and the reference point $\widetilde{V}(x_i; w_V)$ for the current distribution. Implementing (3.15), we introduce

$$\widehat{e_i^2}(w_\theta, w_V) = \Big(q_i(w_V) - \widetilde{V}(x_i; w_V)\Big)^2 \rho\big(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i\big) \tag{5.4}$$

and the estimator

$$\widehat{\Psi}_t(w_V, w_\theta) = \frac{1}{t}\sum_{i=1}^{t} \Big(q_i(w_V) - \widetilde{V}(x_i; w_V)\Big)^2 \rho\big(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i\big). \tag{5.5}$$

In the first approach we may say that we replace minimization of $\Psi(w_V, w_\theta)$, which cannot be done directly, with minimization of $\widehat{\Psi}_t(w_V, w_\theta)$ with respect to w_V. This is yet slightly more complicated. We want to manipulate w_V to bring the approximation $\widetilde{V}(x_i; w_V)$ closer to $q_i(w_V)$ for each i. Both these functions depend on w_V but the latter is here understood as a constant estimator of $Q^{w_\theta}(x_i, u_i)$. Modifications of w_V that adjust $\widetilde{V}(x_i; w_V)$ induce also casual changes of $q_i(w_V)$. We assume that the changes of q_i are much smaller than the changes of $\widetilde{V}$.[1] We proceed as follows: We assign

$$w_V' := w_V$$

and minimize

$$\frac{1}{t}\sum_{i=1}^{t} \Big(q_i(w_V') - \widetilde{V}(x_i; w_V)\Big)^2 \rho\big(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i\big). \tag{5.6}$$

with respect to w_V. We obtain another value of w_V that allows us to calculate another values of $q_i(w_V)$, i.e. better estimators of $Q^{w_\theta}(x_i, u_i)$. We repeat this process until convergence of w_V. Finally, we obtain w_V that satisfies

$$w_V = \arg\min_w \frac{1}{t}\sum_{i=1}^{t} \Big(q_i(w_V) - \widetilde{V}(x_i; w)\Big)^2 \rho\big(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i\big). \tag{5.7}$$

[1]Almost each RL algorithm (e.g. each one referred to in this dissertation) faces the problem of adjusting an approximator of the value function by means of the same approximator. The problem is difficult and probably that is why proofs concerning such procedures require that approximators are linear in their parameters. See [47].

Let us call (5.7) the *quasi-minimum condition* of $\widehat{\Psi}_t$ and the process of finding of such w_V the *quasi-minimization* of $\widehat{\Psi}_t$. Notice that similar quasi-minimization is executed by HDP discussed in Point 2.4.1. Usually we use a gradient to minimize a function. Here the minimization of $\widehat{\Psi}_t$ is replaced by the quasi-minimization, and the gradient $\nabla_{w_V}\widehat{\Psi}_t(w_V, w_\theta)$ can not be in use but rather the gradient of (5.6) on w_V. Let this vector be denoted by

$$(T\widehat{\Psi}_t)(w_V, w_\theta) = \frac{1}{t}\sum_{i=1}^{t}(-2)\frac{\mathrm{d}\widetilde{V}(x_i; w_V)}{\mathrm{d}w_V}\Big(q_i(w_V) - \widetilde{V}(x_i; w_V)\Big)\rho\big(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i\big). \tag{5.8}$$

5.3 Policy optimization

To specify $\widehat{\Phi}_t$ (5.1), we need to introduce $\widehat{U}_i$, an estimator of $U^{w_\theta}(x_i, \widetilde{\theta}(x_i; w_\theta))$. In terms of Section 3.2, considering the drawing u_i we know the density φ_i that generated u_i, and the return $q_i(w_V) = r_{i+1} + \gamma\widetilde{V}(x_{i+1}; w_V)$. We want to estimate the expected return as a function of the parameter $\widetilde{\theta}(x_i; w_\theta)$ that defines the current distribution of the drawings. To maximize the expected return we shall optimize this parameter. We employ here the estimator based on (3.13), taking $\widetilde{V}(x_i; w_V)$ as the reference point, namely

$$\widehat{U}_i(w_\theta, w_V) = \widetilde{V}(x_i; w_V) + \Big(q_i(w_V) - \widetilde{V}(x_i; w_V)\Big)\rho\big(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i\big). \tag{5.9}$$

We thus obtain

$$\widehat{\Phi}_t(w_\theta, w_V) = \frac{1}{t}\sum_{i=1}^{t}\widetilde{V}(x_i;w_V) + \Big(q_i(w_V) - \widetilde{V}(x_i; w_V)\Big)\rho\big(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i\big). \tag{5.10}$$

Let us recall the *temporal difference*

$$d_i(w_V) = r_{i+1} + \gamma\widetilde{V}(x_{i+1}; w_V) - \widetilde{V}(x_i; w_V)$$

and rewrite (5.10) as

$$\widehat{\Phi}_t(w_\theta, w_V) = \frac{1}{t}\sum_{i=1}^{t} d_i(w_V)\rho\big(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i\big) + \frac{1}{t}\sum_{i=1}^{t}\widetilde{V}(x_i; w_V). \tag{5.11}$$

Notice that the second sum of the above does not depend on w_θ.

Therefore, instead of a maximization of $\Phi(w_\theta)$, which cannot be done directly, its estimate $\widehat{\Phi}_t(w_\theta, w_V)$ is maximized with respect to w_θ and hence the policy $\pi(w_\theta)$ is approximately optimized. In maximization, $\widetilde{\theta}(x_i; w_\theta)$ should be kept in some

The exploration loop:

0. Set $t := 1$. Initialize w_θ and w_V randomly.
1. Draw the control action u_t
$$u_t \sim \varphi(\cdot\,; \widetilde{\theta}(x_t; w_\theta))$$
where the parameter vector w_θ is calculated in the internal loop.
2. Perform control u_t, and observe the next state x_{t+1} and the reinforcement r_{t+1}.
3. Add the five $\langle x_t, u_t, r_{t+1}, x_{t+1}, \varphi_t \rangle$ to a database where $\varphi_t := \varphi(u_t; \widetilde{\theta}(x_t; w_\theta))$.
4. Set $t := t + 1$ and repeat from Step 1.

The internal loop:

1. Policy evaluation. Adjust w_V for $\widetilde{V}(x_i; w_V)$ to approximate $V^{w_\theta}(x_i)$ for all $i \in \{1, \dots, t\}$, i.e. to quasi-minimize $\widehat{\Psi}_t(w_V, w_\theta)$ (5.5).
2. Policy improvement. Adjust w_θ to maximize the estimator of $U^\pi(x_i, \widetilde{\theta}(x_i; w_\theta))$ for all $i \in \{1, \dots, t\}$ and the fixed $\pi = \pi(w_\theta)$, i.e. to maximize $\widehat{\Phi}_t(w_\theta, w_V)$ (5.11).

Table 5.1: The IRAWC algorithm.

bounded area, otherwise the solution might not have a physical sense. This is consequence of the fact that the algorithm inherits the distant exploration tendency (see Point 3.2.4) from LONG-MEM-MAX.

The maximization of (5.11) with respect to w_θ optimizes the first step along each trajectory. It changes the policy $\pi(w_\theta)$ so its value function V^{w_θ} must be approximated again. The minimization of (5.5) and the maximization of (5.11) are mutually dependent optimization tasks. The optimization procedures are thus related to those applied in classical Policy Iteration.

The discussion above leads to the algorithm of reinforcement learning based on batch estimation. We call it Intensive Random Actor With Critic (IRAWC) to stress that it is much more computationally intensive than the traditional Actor-Critic algorithm which performs a single weights adjustment per each control step. IRAWC uses two parametric approximators: $\widetilde{\theta}$ forms the randomized control policy $\pi(w_\theta)$ and $\widetilde{V}$ to approximate the value function V^{w_θ}. The algorithm is presented in

Table 5.1. Note that it is not necessary to perform a full optimization in either of two steps of the internal loop.

5.4 Limit behavior, comparison to RAWC

In this section we informally compare limit behavior of IRAWC to limit behavior of RAWC. Suppose IRAWC makes sequences of its parameters $\{w_\theta^t, t \in N\}$ and $\{w_V^t, t \in N\}$ converge to certain limits w_θ^* and w_V^*. Suppose this is the consequence of the fact that the functions $\widehat{\Phi}_t$ and $\widehat{\Psi}_t$ that IRAWC optimizes converge uniformly to their limits Φ^* and Ψ^*, respectively. The convergence implies that

$$\begin{aligned}
\Phi^*(w_\theta, w_V^*) &= \lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} \widetilde{V}(x_i; w_V) + \Big(r_{i+1} + \gamma \widetilde{V}(x_{i+1}; w_V^*) - \widetilde{V}(x_i; w_V^*)\Big) \times \\
&\qquad \times \rho(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i) \\
\Psi^*(w_V, w_\theta^*) &= \lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} \Big(r_{i+1} + \gamma \widetilde{V}(x_{i+1}; w_V^*) - \widetilde{V}(x_i; w_V)\Big)^2 \times \\
&\qquad \times \rho(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta^*)), \varphi_i)
\end{aligned}$$

for any w_θ and w_V. Because the convergence is uniform and in the limit the appropriate functions are optimized, their gradients vanish and we have

$$\begin{aligned}
\nabla_{w_\theta} \Phi^*(w_\theta, w_V^*)|_{w_\theta = w_\theta^*} &= 0 \\
\nabla_{w_V} \Psi^*(w_V, w_\theta^*)|_{w_V = w_V^*} &= 0.
\end{aligned}$$

which is equivalent to the equations

$$\begin{aligned}
0 = \lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} &\Big(r_{i+1} + \gamma \widetilde{V}(x_{i+1}; w_V^*) - \widetilde{V}(x_i; w_V^*)\Big) \nabla_\theta \ln \varphi(u_i; \theta)|_{\theta = \widetilde{\theta}(x_i; w_\theta^*)} \times \qquad (5.12) \\
&\times \frac{\varphi(u_i; \widetilde{\theta}(x_i; w_\theta^*))}{\varphi(u_i; \widetilde{\theta}(x_i; w_\theta^i))} \left[\frac{\varphi(u_i; \widetilde{\theta}(x_i; w_\theta^*))}{\varphi(u_i; \widetilde{\theta}(x_i; w_\theta^i))} < b\right] \\
0 = \lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} &\frac{\mathrm{d}\widetilde{V}(x_i; w_V^*)}{\mathrm{d}w_V^*} \Big(r_{i+1} + \gamma \widetilde{V}(x_{i+1}; w_V^*) - \widetilde{V}(x_i; w_V^*)\Big) \times \qquad (5.13) \\
&\times \min\left\{\frac{\varphi(u_i; \widetilde{\theta}(x_i; w_\theta^*))}{\varphi(u_i; \widetilde{\theta}(x_i; w_\theta^i))}, b\right\}.
\end{aligned}$$

On the other hand the convergence of RAWC (see Table 4.1) takes place for w_θ and w_θ such that the expected improvement of the these parameters does not take place.

That is for w_θ^+ and w_V^+ such that the equalities

$$0 = \mathcal{E}\left(\left(r_{t+1} + \gamma\widetilde{V}(x_{t+1}; w_V^+) - \widetilde{V}(x_t; w_V^+)\right)\nabla_\theta \ln\varphi(u_t;\theta)|_{\theta=\widetilde{\theta}(x_t;w_\theta^+)}\Big|C^+\right) \tag{5.14}$$

$$0 = \mathcal{E}\left(\frac{\mathrm{d}\widetilde{V}(x_t; w_V^+)}{\mathrm{d}w_V^+}\left(r_{t+1} + \gamma\widetilde{V}(x_{t+1}; w_V^+) - \widetilde{V}(x_t; w_V^+)\right)\Big|C^+\right) \tag{5.15}$$

take place. x_t, u_t, r_{t+1}, x_{t+1} are here random variables. Condition C^+ states that x_t is drawn from the steady state distribution $\eta^{w_\theta^+}$ and the control u_t is drawn from $\varphi(\cdot\,; \widetilde{\theta}(x_t; w_\theta^+))$. x_{t+1} and r_{t+1} constitute a response of the plant to the pair $\langle x_t, u_t\rangle$.

Notice the similarities between the formulae (5.12) and (5.14) and between (5.13) and (5.15). Obviously the infinite averages are replaced with expected values. The main differences lie in the fact that (5.12) and (5.13) contain quotients of densities that do not appear in (5.14) and (5.15). They "compensate" the fact that consecutive controls u_i have been drawn from distributions different than $\varphi(\cdot\,; \widetilde{\theta}(x_i; w_\theta^*))$. Propositions 8 and 9 of Chapter 3 state that such compensation mechanism gives estimators whose biases vanish when the distribution that generated the sample approaches the one analyzed.

Another difference between pairs (5.12), (5.13) and (5.14), (5.15) lies in the fact that the states x_i that appear in (5.12) and (5.13) have not been drawn from the limit stationary distribution. However, it is proved [45, 17] that in the linear case the convergence of the policy implies the convergence of the stationary state distribution. Summing up, there are reasons to believe that IRAWC converges to the same limit points that RAWC does.

5.5 The algorithm at work

Let us now analyze the behavior of the algorithm in early stages of its work, before its limit properties are reached. We shall consider a special case, such that φ is a family of normal distributions (3.31). This case seems to be the most interesting from the practical point of view. We shall also generalize some of the conclusions.

Consider $\widehat{\Phi}_t$ (5.11) as a function of w_θ.

$$\widehat{\Phi}_t(w_\theta, w_V) = \frac{1}{t}\sum_{i=1}^{t} d_i(w_V)\rho\big(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i\big) + \frac{1}{t}\sum_{i=1}^{t}\widetilde{V}(x_i; w_V).$$

It consists of a certain constant element and a weighted sum of $d_i(w_V)$ where w_θ determines weights. The maximization of this function in the algorithm's internal loop has a very simple interpretation. It maximizes $\varphi(u_i; \widetilde{\theta}(x_i; w_\theta))$ for $d_i(w_V) > 0$ and minimizes this value for $d_i(w_V) < 0$. In the case of the normal density, that means that for $d_i(w_V) > 0$ the maximization brings the value of $\widetilde{\theta}(x_i; w_\theta)$ closer to

u_i. In opposite case $\widetilde{\theta}(x_i; w_\theta)$ is brought as far as possible from u_i. Because it is likely that $d_i(w_V) < 0$ for some i, the values of $\widetilde{\theta}(x_i; w_\theta)$ must be kept within a bounded area. Otherwise the algorithm may find out that in order to minimize the probability of action u_i in state x_i, the value $\widetilde{\theta}(x_i; w_\theta)$ should be infinite.

In fact, we may say that in very early stage of its work, when t is no larger than the dimension of w_V, the mechanism that the algorithm is based on is not valid. Notice that $\widehat{\Phi}_t(w_\theta, w_V)$ (5.11) is constructed on the basis of

$$\widetilde{V}(x_i; w_V) + \Big(q_i(w_V) - \widetilde{V}(x_i; w_V)\Big)\rho\big(\varphi(u_i; \theta), \varphi_i\big) \tag{5.16}$$

where $\widetilde{V}(x_i; w_V)$ is the reference point. According to Point 3.2.1, the estimator of this type has favorable properties when its reference point is independent from the drawing (from u_i that determines $q_i(w_V)$ in this case). However, the Critic training consists in minimizing

$$\Big(q_i(w_V) - \widetilde{V}(x_i; w_V)\Big)^2 \rho\big(\varphi(u_i; \theta), \varphi_i\big)$$

with respect to $\widetilde{V}(x_i, w_V)$ for all i. This makes $\widetilde{V}(x_i; w_V)$ dependent on u_i. Fortunately, $\widetilde{V}(x_i; w_V)$ is determined on the basis of all the data, hence as t grows, its dependence on u_i for this particular i vanishes and the estimator (5.16) reaches its favourable properties. Yet when t is no larger than the dimension of w_V, the value of $\widetilde{V}(x_i, w_V)$ may be considered dependent on u_i entirely.

The computational process conducted in the internal loop of IRAWC may be understood as a mechanism that enables to distinguish good actions from bad ones in all regions of the state space. However, the distinction is based on a comparison of various actions. Before the action space is not covered with "tried" points for different state regions, the algorithm is likely to generate the actions not similar to any control stimuli tried so far. This yields a good exploration, yet in cases when the action space is large and of many dimensions, its thorough exploration could be less beneficial than a local steepest ascent search.

Let us now analyze the role of the ρ function (3.12) used in the construction of estimators. Replacing a quotient of densities by ρ is a source of bias of the estimators $\widehat{U}_i$ (5.9) and $\widehat{e_i^2}$ (5.4). However, ρ has two favorable properties. First, the bias that ρ produces vanishes when the distribution generating the appropriate action converges to the action's distribution implied by the current policy. Hence, if the policy converges to the right limit, the bias converge to zero. Second, ρ is limited when the distance between distribution that generated the action and the one implied by the current policy increases. To see a usefulness of this property, suppose we want to infer about a certain policy with the use of observations of a control process driven by another, distant policy. Let us denote ρ with upper limit equal to b by ρ_b. If we use ρ_∞, (3.13) and (3.15) return to their unbiased versions (3.11) and (3.14). However, for an action unlikely for the policy that generated the action (small φ_i) and typical for the current policy (large $\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)))$, the value of ρ_∞ (the quotient of the two densities) is large. This value in turn becomes a weight of the associated observation in $\widehat{\Psi}_t$ (5.5) and $\widehat{\Phi}_t$ (5.11). This weight of such a particular observation is as large as the value of ρ_∞ even if the current policy would have generated new, more representative data. It seems wise to artificially decrease the weight of such accidental data to increase comparatively weight of "fresh" data coming from the current policy observation. The problem of an appropriate specification of ρ certainly requires further investigation.

5.6 Implementation of the internal loop

Implementation of IRAWC may seem to be quite complicated. It must encompass two mutually dependent optimization processes performed in real time of a working plant. Furthermore, the complexity of these processes increases in time. Experiments presented in Chapter 7 show that the use of common optimization techniques yields a satisfying behavior of IRAWC. However, there are some approaches we may particularly recommend.

Note that the optimized functions $\widehat{\Psi}_t$ and $\widehat{\Phi}_t$ have the form of certain sums. This enables to optimize them by processing their elements separately, i.e. to optimize both functions by means of stochastic approximation. Furthermore, the number of elements of the sums increases with the exploration time and may finally reach a very large value.

One may choose between batch optimization techniques and methods that process the sums element by element. Taking into consideration that the number of elements is growing, the second possibility seems to be better. Our experiments confirm this intuition: The first-order batch methods behave poorly, the second-order batch methods perform substantially better, but both are outperformed by a first-order incremental algorithm which is used in simulations shown in Chapter 7.

If $\widetilde{V}$ and $\widetilde{\theta}$ are implemented by neural networks, the problem of increasing set of data is particularly difficult. Additionally, at the beginning of the learning process when there is little data collected it is recommended to slow the optimization down. Otherwise, the networks may get stuck in local minima.

According to our experience, we may recommend stochastic approximation for optimizations of $\widehat{\Psi}_t$ and $\widehat{\Phi}_t$ implemented in the following loop:

1. Draw a random i from the uniform discrete distribution on $\{1, \dots, t\}$.

2. Adjust w_V along the i-th component of $(T\widehat{\Psi}_t)$.

3. Adjust w_θ along the gradient of the i-th component of $\widehat{\Phi}_t$ on w_θ.

The consecutive i-s may form random permutations of all numbers in $\{1, \dots, t\}$. The algorithm we use in the experimental study (Chapter 7) optimizes $\widehat{\Psi}_t$ and $\widehat{\Phi}_t$ in the way described above. In its internal loop, steps 1, 2, 3 are repeated n times, following every step of the exploration loop.

Chapter 6

Extensions

In the present chapter we propose several enhancements of the long-term-memory algorithm, IRAWC, presented in the previous chapter. First, we replace an estimator that the algorithm is based on with another one whose bias is smaller. This modification is driven by the same argument we employed to extend RAWC to RAWC(λ) in Chapter 4. Large memory requirements of IRAWC in some cases put this algorithm at a disadvantage. We cope with this problem and propose an algorithm that lies exactly between RAWC and IRAWC. This method rests on limited memory capacity.

In the Reinforcement Learning problem we deal in this dissertation, state observations are taken and control stimuli are applied in discrete time. However, in present control applications, with very fast computer controllers, control processes can naturally be modeled in continuous time. The last section of this chapter is devoted to speculations about extension of the concept of IRAWC to continuous time problems.

The algorithms presented below employ the same Actor and Critic parts as RAWC and IRAWC. $\varphi(\cdot\ ;\theta)$ is an action density function, where θ is determined by an approximator $\widetilde{\theta}(x; w_\theta)$ parametrized by w_θ. φ and $\widetilde{\theta}$ together define a policy $\pi(w_\theta)$. Another function $\widetilde{V}(x; w_V)$ parametrized by w_V approximates the value function $V^{\pi(w_\theta)}(x)$.

6.1 IRAWC with λ-estimators of future rewards

In the IRAWC algorithm we maximize an estimator $U^\pi(x_i; \widetilde{\theta}(x_i; w_\theta))$ of the expected value of a sum of future discounted rewards for each state x_i visited so far. This estimator is in general biased, because it is based on an approximation of the value function V^π instead of the value function itself. In Section 4.6 the RAWC algorithm is extended to RAWC(λ) by replacing an estimator of $Q^\pi(x_t; u_t)$ with one whose bias is smaller. In this section we extend IRAWC to IRAWC(λ) in a similar manner. The resulting IRAWC(λ) algorithm performs exactly the same activities as IRAWC does, except it calculates appropriate estimators differently.

We need to introduce two statistics that estimate $U^{w_\theta}(x_i, \widetilde{\theta}(x_i; w_\theta))$ as well as $\big(V^{w_V}(x_i) - \widetilde{V}(x_i; w_V)\big)^2$ whose biases are smaller than the biases of $\widehat{U}_i(w_\theta, w_V)$ (5.9) and $\widehat{e^2_i}(w_\theta, w_V)$ (5.4) respectively. In the previous chapter we defined these estimators with the use of the importance sampling technique presented in Chapter 3, namely

$$\begin{aligned}\widehat{U}_i(w_\theta, w_V) =& \widetilde{V}(x_i; w_V) + \rho(\varphi(u_i; \widetilde{\theta}(u_i; w_\theta)), \varphi_i) \times \\ & \times \left(q_i - \widetilde{V}(x_i; w_V)\right) \\ \widehat{e^2_i}(w_\theta, w_V) =& \rho(\varphi(u_i; \widetilde{\theta}(u_i; w_\theta)), \varphi_i) \times \\ & \times \left(q_i - \widetilde{V}(x_i; w_V)\right)^2\end{aligned}$$

where q_i was a certain estimator of the return $Q^{w_\theta}(x_i, u_i)$ associated with the control stimulus u_i applied in the state x_i. Because we could not use the unbiased estimator

$$\widehat{q_i} = r_{i+1} + \gamma V^{w_\theta}(x_{i+1}),$$

we used the biased one

$$q_i = r_{i+1} + \gamma \widetilde{V}(x_{i+1}; w_V).$$

The expected value of a sum of future discounted rewards in the following state, i.e. $V^{w_\theta}(x_{i+1})$, is generally different than its approximation $\widetilde{V}(x_{i+1}; w_V)$. The estimator q_i is thus biased. In (4.21) we saw how to cope with such a bias problem. Namely, we calculated an estimator that combined the approximation $\widetilde{V}$ with the rewards that actually had been received. We shall follow this idea, yet with certain modifications. Let $\{\lambda_i, i \in N\}$ be a sequence of numbers from the interval $[0, \lambda_{\max}]$. Let us define the desired estimators as follows:

$$\begin{aligned}\widehat{U}^*_i(w_\theta, w_V) =& \widetilde{V}(x_i; w_V) + \rho(\varphi(u_i; \widetilde{\theta}(u_i; w_\theta)), \varphi_i) \times \qquad (6.1) \\ & \times \left(q^*_i - \widetilde{V}(x_i; w_V)\right) \\ \widehat{e^{2*}_i}(w_\theta, w_V) =& \rho(\varphi(u_i; \widetilde{\theta}(u_i; w_\theta)), \varphi_i) \times \qquad (6.2) \\ & \times \left(q^*_i - \widetilde{V}(x_i; w_V)\right)^2\end{aligned}$$

where

$$q^*_i = r_{i+1} + \gamma\big(\lambda_i \widehat{U}^*_{i+1}(w_\theta, w_V) + (1 - \lambda_i)\widetilde{V}(x_{i+1}; w_V)\big). \qquad (6.3)$$

Here we do not rest entirely on the approximation $\widetilde{V}$ of the value function in the state x_{i+1}, but combine the approximation with the estimator $\widehat{U}^*_{i+1}$ of the value function calculated for the following state. The smaller λ_i, the more we rely on the approximation. The larger λ_i, the more we rely on received rewards. In comparison to what was done in Section 4.6, we simply resign from constancy of the parameter λ. This resignation is driven by a need of bounding variances of certain estimators as the following argument shows. Notice that according to the idea of IRAWC, we need $\widehat{U}_i^*(w_\theta, w_V)$ and $\widehat{e_i^{2*}}(w_\theta, w_V)$, and we need maximize or minimize them with respect to w_θ and w_V. What really interest us for optimization purposes are gradients:

$$\begin{aligned} g_i^U &= \frac{\mathrm{d}}{\mathrm{d}w_\theta}\widehat{U}_i^*(w_\theta, w_V) && (6.4)\\ &= \frac{\mathrm{d}\rho(\varphi(u_i;\widetilde{\theta}(u_i;w_\theta)),\varphi_i)}{\mathrm{d}w_\theta}\left(q_i^* - \widetilde{V}(x_i;w_V)\right) \\ g_i^e &= \frac{\mathrm{d}}{\mathrm{d}w_V}\widehat{e_i^{2*}}(w_\theta, w_V) && (6.5)\\ &= -2\frac{\mathrm{d}\widetilde{V}(x_i;w_V)}{\mathrm{d}w_V}\rho(\varphi(u_i;\widetilde{\theta}(u_i;w_\theta)),\varphi_i)\left(q_i^* - \widetilde{V}(x_i;w_V)\right). \end{aligned}$$

g_i^U and g_i^e are results of a certain sequence of drawings (of control stimuli). Variances of these statistics should be bounded. Otherwise, g_i^U and g_i^e would be useless for optimization purposes. We will restrict their variances by defining appropriate values of λ_i. In the remaining part of this section we analyze how to determine λ_i.

To analyze variances of g_i^U and g_i^e, suppose both

$$\frac{\mathrm{d}\rho(\varphi(u_i;\widetilde{\theta}(u_i;w_\theta)),\varphi_i)}{\mathrm{d}w_\theta}$$

and

$$\frac{\mathrm{d}\widetilde{V}(x_i;w_V)}{\mathrm{d}w_V}$$

are bounded. This assumption is satisfied when $\widetilde{\theta}$ and $\widetilde{V}$ are implemented with the use of feedforward neural networks and the parameters w_θ and w_V are bounded. In this case, bounded variance of

$$g_i = \rho(\varphi(u_i;\widetilde{\theta}(u_i;w_\theta)),\varphi_i)\left(q_i^* - \widetilde{V}(x_i;w_V)\right).$$

is obviously enough for variances of g_i^U and g_i^e to be bounded. We will thus confine variance of g_i. Let denote

$$\begin{aligned} \rho_i &= \rho(\varphi(u_i;\widetilde{\theta}(u_i;w_\theta)),\varphi_i) \\ d_i &= r_{i+1} + \gamma\widetilde{V}(x_{i+1};w_V) - \widetilde{V}(x_i;w_V). \end{aligned}$$

From (6.1) and (6.3) we have

$$\begin{aligned}
g_i &= \widehat{U}_i^*(w_\theta, w_V) - \widetilde{V}(x_i; w_V) \\
&= \rho_i \cdot \Big(r_{i+1} + \gamma \widetilde{V}(x_{i+1}; w_V) - \widetilde{V}(x_i; w_V) + \gamma\lambda_i \big(U_{i+1}^*(w_\theta, w_V) - \widetilde{V}(x_{i+1}; w_V) \big) \Big) \\
&= \rho_i d_i + \rho_i \gamma \lambda_i \big(U_{i+1}^*(w_\theta, w_V) - \widetilde{V}(x_{i+1}; w_V) \big) \\
&= \sum_{k=0}^{\infty} \rho_i \left(\prod_{j=0}^{k-1} \gamma \lambda_{i+j} \rho_{i+j+1} \right) d_{i+k}
\end{aligned}$$

In the above formula γ is constant, λ_i-s are constant values that we are going to define. Both ρ_{i+j}-s and d_{i+k}-s are random in the sense that their values are results of drawing. Suppose for a while that ρ_{i+j}-s are constant, and d_i-s are random variables whose variances are bounded by σ^2. Obviously

$$\mathcal{V} g_i \le \Upsilon_i^2 \sigma^2$$

where

$$\Upsilon_i = \sum_{k=0}^{\infty} \rho_i \left(\prod_{j=0}^{k-1} \gamma \lambda_{i+j} \rho_{i+j+1} \right).$$

We are going to determine λ_i on the basis of ρ_{i+j} for $j \ge 0$ so to keep Υ_i smaller than a certain $\Upsilon_{\max}$. In this case

$$\mathcal{V} g_i \le \Upsilon_{\max}^2 \sigma^2,$$

even if we allow ρ_i to be random.

Now it becomes clear why we gave up constancy of λ. Even if consecutive ρ-s are artificially bounded, their infinite product may grow to infinity. If g_i is calculated as multiplications of random variables and such products, its variance may be large and it must be confined by some auxiliary trick.

In order to derive λ_i, a recursion that combines Υ_i, λ_i, and Υ_{i+1} is needed

$$\begin{aligned}
\Upsilon_i =& \sum_{k=0}^{\infty} \rho_i \left(\prod_{j=0}^{k-1} \gamma \lambda_{i+j} \rho_{i+j+1} \right) \\
=& \rho_i + \sum_{k=1}^{\infty} \rho_i \left(\prod_{j=0}^{k-1} \gamma \lambda_{i+j} \rho_{i+j+1} \right) \\
=& \rho_i + \rho_i \sum_{k=1}^{\infty} \gamma \lambda_i \rho_{i+1} \left(\prod_{j=1}^{k-1} \gamma \lambda_{i+j} \rho_{i+j+1} \right) \\
=& \rho_i + \rho_i \gamma \lambda_i \sum_{k=1}^{\infty} \rho_{i+1} \left(\prod_{j=0}^{k-2} \gamma \lambda_{i+1+j} \rho_{i+1+j+1} \right) \\
=& \rho_i (1 + \gamma \lambda_i \Upsilon_{i+1}) \qquad (6.6)
\end{aligned}$$

We want Υ_i to be no greater than $\Upsilon_{\max}$. We also want λ_i to be no greater than a value $\lambda_{\max}$ which may be identified with the constant λ parameterizing the RAWC(λ) algorithm. Simple transformations of (6.6) give

$$\lambda_i = \min\{\lambda_{\max}, (\Upsilon_{\max}/\rho_i - 1)/(\gamma\Upsilon_{i+1})\}. \tag{6.7}$$

Finally, we obtain a recursive definition of the required estimators $\widehat{U}_i^*(w_\theta, w_V)$ and $\widehat{e_i^{2*}}(w_\theta, w_V)$. The recursion leads from larger i-s to smaller ones. It is applicable to all i such that both x_i and its successor x_{i+1} are known. However, the two step ahead follower x_{i+2} may be unknown. This happens when a reinforcement learning experiment has finished after step $i+1$.

$$\rho_i = \rho(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i) \tag{6.8}$$

$$\begin{cases} \text{if } x_{i+1} \text{ is a terminal state:} \\ \Upsilon_i = 0 \\ q_i^* = r_{i+1} \\ \text{otherwise if the successor of } x_{i+1} \text{ is unknown:} \\ \Upsilon_i = 0 \\ q_i^* = r_{i+1} + \gamma\widetilde{V}(x_{i+1}; w_V) \\ \text{otherwise:} \\ \lambda_i = \begin{cases} \lambda_{\max} & \text{if } \Upsilon_{i+1} = 0 \\ \min\{\lambda_{\max}, (\Upsilon_{\max}/\rho_i - 1)/(\gamma\Upsilon_{i+1})\} & \text{if } \Upsilon_{i+1} \neq 0 \end{cases} \\ \Upsilon_i = \rho_i(1 + \lambda_i\gamma\Upsilon_{i+1}) \\ q_i^* = r_{i+1} + \gamma\big(\lambda_i\widehat{U}_{i+1}^*(w_\theta, w_V) + (1-\lambda_i)\widetilde{V}(x_{i+1}, w_V)\big) \end{cases} \tag{6.9}$$

$$\widehat{U}_i^*(w_\theta, w_V) = \widetilde{V}(x_i; w_V) + \rho(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i)\left(q_i^* - \widetilde{V}(x_i; w_V)\right) \tag{6.10}$$

$$\widehat{e_i^{2*}}(w_\theta, w_V) = \rho(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i)\left(q_i^* - \widetilde{V}(x_i; w_V)\right)^2 \tag{6.11}$$

The IRAWC(λ) algorithm may be formulated almost exactly as IRAWC (Table 5.1), yet instead of $\widehat{\Psi}_t$ and $\widehat{\Phi}_t$, it optimizes $\widehat{\Psi}_t^*$ and $\widehat{\Phi}_t^*$ defined as

$$\widehat{\Psi}_t^*(w_V, w_\theta) = \frac{1}{t}\sum_{i=1}^{t}\left(q_i^* - \widetilde{V}(x_t; w_V)\right)^2\rho(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i) \tag{6.12}$$

$$\widehat{\Phi}_t^*(w_\theta, w_V) = \frac{1}{t}\sum_{i=1}^{t}\widetilde{V}(x_i; w_V) + (q_i^* - \widetilde{V}(x_i; w_V))\rho(\varphi(u_i; \widetilde{\theta}(x_i; w_\theta)), \varphi_i) \tag{6.13}$$

where q_i^* is calculated by means of the recursion (6.8) – (6.11). One could optimize the above functions in the batch mode. This would require a recalculation of the recursion after each step of the optimization.

6.2 Sequential implementation of IRAWC(λ)

Suppose, in order to optimize Ψ^*_t and Φ^*_t we want to use the pattern mode instead of the batch one. We simply draw i from the set $\{1,\dots,t\}$ and optimize these functions along i-th components of (6.12) and (6.13). How to do this avoiding recalculation of the entire database just to compute a single q^*_i? In fact, we will not compute q^*_i directly. Instead, we will calculate a random variable whose expected value is equal to q^*_i.

Let ξ be a random variable from the uniform distribution $U(0,1)$. Suppose it is drawn independently whenever it appears in a formula. Let also ξ be the only random variable in the below expected value and $[\cdot]$ have the conventional meaning. We have

$$\begin{aligned}\mathcal{E}\Big(r_{i+1}&+\gamma\widetilde{V}(x_{i+1};w_V)+[\xi<\lambda_i]\gamma(\widehat{U}^*_{i+1}(w_\theta,w_V)-\widetilde{V}(x_{i+1};w_V))\Big)\\&=r_{i+1}+\gamma\widetilde{V}(x_{i+1};w_V)+\lambda_i\gamma(\widehat{U}^*_{i+1}(w_\theta,w_V)-\widetilde{V}(x_{i+1};w_V))\\&=q^*_i.\end{aligned}$$

We obtained q^*_i (6.3). Let us compute $\widehat{q}^*_i$ with the use of the following recurrent procedure:

$$\widehat{q}^*_i:=\begin{cases}r_{i+1} & \text{if } x_{i+1} \text{ is terminal,}\\ r_{i+1}+\gamma\widetilde{V}(x_{i+1};w_V) & \text{if } x_{i+2} \text{ is unknown or } \xi\ge\lambda_i,\\ r_{i+1}+\gamma\widehat{U}^*_{i+1}(w_\theta,w_V) & \text{otherwise,}\end{cases}\tag{6.14}$$

where $\widehat{U}^*_{i+1}(w_\theta,w_V)$ is recalculated by means of (6.10) and (6.14). Instead of using q^*_i which requires recalculating of the entire database, we may hence use $\widehat{q}^*_i$ whose expected value is equal to q^*_i. The calculation of $\widehat{q}^*_i$ requires several steps of the above recursion. The number of steps is random yet its expected value is no larger than $1+1/\lambda_{\max}$ if only $\lambda_i\le\lambda_{\max}$ for each i.

We now see how the pattern mode of optimization of (6.12) and (6.13) can be implemented. Within each step, an index i is drawn from $\{1,\dots,t\}$, $\widehat{q}^*_i$ is calculated on the basis of (6.10) and (6.14), and i-th components of $\widehat{\Psi}^*_t$ and $\widehat{\Phi}^*_t$ are optimized along w_V or w_θ. What we need now is a method of computing values of λ_i. We will set the values of consecutive λ_i-s to bound

$$\begin{aligned}\Upsilon^K_i&=\mathcal{E}\left(\sum_{k=0}^{K+1}\rho_i\left(\prod_{j=0}^{k-1}[\xi_{i+j}<\lambda_{i+j}]\gamma\rho_{i+j+1}\right)\right)\\&=\sum_{k=0}^{K+1}\rho_i\left(\prod_{j=0}^{k-1}\gamma\lambda_{i+j}\rho_{i+j+1}\right)\end{aligned}$$

for each $K = 0, 1, \dots$ where every term is understood as constant, except ξ_{i+j}, which now becomes an element of a sequence of independent random variables drawn from $U(0, 1)$. We will calculate consecutive λ_{i+K} so that they will not be greater than $\lambda_{\max}$ and Υ_i^K will not be greater than $\Upsilon_{\max}$. Let us express Υ_i^K as

$$\Upsilon_i^K = \sum_{k=0}^{K} a_i^k \lambda_{i+k}$$

In order to find consecutive a_i^K and λ_{i+K}, let us define:

$$c_i^K = \rho_i \prod_{j=0}^{K-1} \gamma \lambda_{i+j} \rho_{i+j+1}$$
$$b_i^K = \sum_{k=0}^{K} c_i^k$$

We can see that

$$a_i^K = c_i^K \gamma \rho_{i+K+1}$$
$$\Upsilon_i^K = \sum_{k=0}^{K} a_i^k \lambda_{i+k} = b_i^K + a_i^K \lambda_{i+K}$$

We want λ_{i+K} not to be greater than $\lambda_{\max}$ nor Υ_i^K to be greater than $\Upsilon_{\max}$. We thus obtain:

$$\lambda_{i+K} = \min\{\lambda_{\max}, (\Upsilon_{\max} - b_i^K)/a_i^K\} \qquad (6.15)$$

Notice that once we pick i to calculate q_i^*, we compute λ_{i+k} for $k \geq 0$. The sequence $\{\lambda_j, j = i, i+1, \dots\}$ we obtain is generally different than the one we would obtain if we picked different i. Furthermore, this sequence depends on w_θ, w_V and the history of plant-controller interaction that took place after step i. What we have denoted here by λ_{i+k} should have been denoted rather by $\lambda_{i,k}(w_\theta, w_V)$. However, it would have made this, technical enough, discussion even more complicated.

If we calculate λ_{i+k} on the basis of the above procedure, we obtain $\Upsilon_i^K \leq \Upsilon_i^{K+1}$ for $K \geq 0$. Furthermore, if $\Upsilon_i^K < \Upsilon_{\max}$ for a certain K, then $\lambda_{i+k} = \lambda_{\max}$ for all $k \leq K$. Conversely, if $\Upsilon_i^K = \Upsilon_{\max}$, then $\lambda_{i+k} = 0$ for all $k > K$. The sequence $\langle \lambda_i, \lambda_{i+1}, \dots \rangle$ is thus equal to $\langle \lambda_{\max}, \dots, \lambda_{\max}, \lambda_{i+K}, 0, \dots \rangle$, where K is the smallest number such that $\Upsilon_i^K = \Upsilon_{\max}$.

What is the cost of an execution of the recursion defined by (6.10) and (6.14)? Its first iteration always takes place. A consecutive iteration always takes place with the probability λ_i. Since $\lambda_i \leq \lambda_{\max}$ for each i, the expected number of iterations is no greater than 1 plus the expected value of the geometrical distribution with parameter equal to $\lambda_{\max}$. Hence this number is no greater than $1/(1 - \lambda_{\max})$. That means that computational expense associated with this form of IRAWC(λ) is no more than $1/(1-\lambda_{\max})$ times larger than the cost of an equivalent implementation of IRAWC.

In Chapter 7 we report application of the sequential version of the IRAWC(λ) algorithm. The batch version of this method presented in the previous section requires very large computational power and we do not recommend it.

6.3 Between RAWC and IRAWC. Semi-Intensive algorithms

RAWC and RAWC(λ) algorithms discussed in Chapter 4 utilize each control step to calculate gradients and use them to adjust the quality measures Φ and Ψ. Long-term-memory versions of these algorithms discussed in the previous and present chapters construct estimators of Φ and Ψ parameterized by Actor's and Critic's parameters and execute a full optimization of these estimators. Objectives are thus similar but means are different.

Usefulness of long-term-memory RL methods is based on the assumption that the entire history of a plant-controller interactions is possible to keep in a computer memory. If we want the control optimization process not to last more than several hours, this assumption seems to be reasonable. However, it is unlikely that a trial and error method of control optimization would be used more than several hours. We either can guarantee that the control system learning will be completed in a short time and then computer memory limitations do not matter or the method would not be applied at all.

Let us consider control optimization problems that impose some strong limitations on memory capacity and computational power. It seems that we still can take an advantage of long-term-memory Actor-Critic approach. The idea is as follows: instead of keeping the entire history of plant-controller interaction, let the algorithm keep in memory only n most resent events. These n events are enough to calculate estimators of Φ and Ψ. Because n is constant, variances of these estimators have no opportunity to converge to zero and their full optimization would lead to poor performance. However, gradients of these estimators reflect gradients of Φ and Ψ much better than the gradients computed by RAWC and RAWC(λ).

The above idea leads us to Semi–Intensive Random Actor With Critic (SIRAWC, Table 6.1). It is similar to the IRAWC algorithm presented in Table 5.1, except for the following differences:

```
0. Set $t := 1$. Initialize $w_\theta$ and $w_V$ randomly.
1. Draw the control action $u_t$
       $u_t \sim \varphi(\cdot\,; \widetilde{\theta}(x_t; w_\theta))$
   where the parameter vector $w_\theta$ is calculated in the internal loop.
2. Perform control $u_t$, and observe the next state $x_{t+1}$ and the reinforcement $r_{t+1}$.
3. Add the five $\langle x_t, u_t, r_{t+1}, x_{t+1}, \varphi_t \rangle$ to the database where $\varphi_t := \varphi(u_t; \widetilde{\theta}(x_t; w_\theta))$
   and remove from the database all events up to the step $t - n$.
4. (a) Policy evaluation. Adjust $w_V$ by performing limited number of steps of
       quasi-minimization of $\widehat{\Psi}^n_t(w_V, w_\theta)$.
   (b) Policy improvement. Adjust $w_\theta$ by performing limited number of steps
       of maximization of $\widehat{\Phi}^n_t(w_\theta, w_V)$.
5. Set $t := t + 1$ and repeat from Step 1.
```

Table 6.1: The SIRAWC algorithm.

1. Instead of $\widehat{\Psi}_t$ and $\widehat{\Phi}_t$ it optimizes

$$\widehat{\Phi}^n_t(w_\theta, w_V) = \frac{1}{n} \sum_{i=t-n+1}^{t} \widehat{U}_i(w_\theta, w_V)$$
$$\widehat{\Psi}^n_t(w_V, w_\theta) = \frac{1}{n} \sum_{i=t-n+1}^{t} \widehat{e}^2_i(w_\theta, w_V)$$

2. A number of optimization steps performed between consecutive control steps must be limited.

Note that the RAWC algorithm may be viewed as a special case of SIRAWC such that $n = 1$ and it performs a single, gradient-driven weights' adjustment at each step. The similarity between these methods is deeper. Even though SIRAWC is structurally like IRAWC, its hypothetical convergence is a result of a stochastic ascent along a gradient estimator, like in RAWC rather than in IRAWC. Thank to more economical utilization of acquired data, the estimator may have smaller variance and SIRAWC may converge faster.

In a similar manner the IRAWC(λ) algorithm can be modified to SIRAWC(λ). It only requires defining of the estimators:

$$\widehat{\Phi}_t^{n*}(w_\theta, w_V) = \frac{1}{n} \sum_{i=t-n+1}^{t} \widehat{U}_i^*(w_\theta, w_V)$$

$$\widehat{\Psi}_t^{n*}(w_V, w_\theta) = \frac{1}{n} \sum_{i=t-n+1}^{t} \widehat{e_i^{2*}}(w_\theta, w_V)$$

6.4 Continuous time

Reinforcement learning in continuous time domain is a rarely discussed subject. In [13] a continuous-time RL algorithm is described that builds a plant's model on-line and in the same time adjusts the policy by means of the mechanism very similar to HDP (Paragraph 2.4.1). The optimized policy is deterministic and hence this algorithm suffers from problems we discuss in Appendix A. In [33] the Simple Policy Search algorithm (see Section 2.5) is discussed that does not really care whether the optimized policy operates in a discrete or continuous time space.

If we want to apply randomized Actor-Critic algorithms to real-time control we may choose from between two possibilities.

- We may assume that control is piecewise constant and apply consecutive u_t directly as a control stimuli. Obviously whenever a digital controller is employed, time must be discredited and consequently control is piecewise constant. However, very important difficulties emerge when discretization becomes very fine. Namely, each consecutive control stimulus impacts the overall control quality very mildly. Roughly speaking, a good control policy can not be spoilt by mismatching a control signal for a very short time. If an action does not make any difference, it is impossible to learn which is good and which is lousy. As a result, this approach is limited to problems whose solution may be piecewise constant control with pieces long enough.

- For many plants, elaborate, nonlinear controllers of many parameters are required. However, within a short period of control, when the state of the plant does not change drastically, a linear controller of few parameters is enough. Yet the parameters must be changed following long time state changes. We thus introduce hierarchy into control policy. A reinforcement learning algorithm operates on the higher level and its actions u_t become parameters for the lower level. On the lower level, a fixed function maps a state and the parameter to a control stimulus.

Note that when we replace the first design with the second one, the dimensionality of the action space $\mathcal{U}$ increases substantially. For instance, suppose that our plant is controlled by 3 stimuli. Within the first approach, the dimensionality of $\mathcal{U}$ is equal to 3. Within the second design it depends on how we design the low-level controller, but probably each controller for such a problem we can think of would require much more than 3 parameters. E.g. a combination of 3 independent PID controllers would require 9 parameters. We devote the remaining of this section to introduce a certain modification of the second above design.

Let us consider continuous time τ and a plant whose state $x(\tau) \in \mathcal{X} = \mathbb{R}^{n_\mathcal{X}}$ evolves according to the equation

$$\frac{\mathrm{d}}{\mathrm{d}\tau} x(\tau) = f(x(\tau), a(\tau)) \tag{6.16}$$

where f is a certain unknown function and $a(\tau) \in \mathcal{A} = \mathbb{R}^{n_\mathcal{A}}$ is a control stimulus applied by a controller. We introduce hierarchy into control policy. $a(\tau)$ becomes a low-level stimulus computed on the basis of a state and a parameter determined on the higher control level. The parameter is a piece-wise constant function of time. It changes in discrete moments $\{\tau_t, t = 1, 2, \dots\}$ such that $\tau_{t+1} - \tau_t \equiv \Delta$ for a certain $\Delta \in \mathbb{R}_+$. We have

$$a(\tau) = C(x(\tau); u_t) \tag{6.17}$$

for $\tau \in [\tau_t, \tau_{t+1})$ where C is a known function defined by a system designer and u_t is a value of the parameter. The task of a reinforcement learning algorithm is to learn to determine u_t on the basis of an observed state $x_t = x(\tau_t)$. The plant generates a stream of reinforcements

$$r(\tau) = R(x(\tau), a(\tau))$$

where $R : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$ is a certain function. For each t, the average reinforcement within the interval $[\tau_t, \tau_{t+1})$

$$r_{t+1} = \Delta^{-1} \int_{\tau_t}^{\tau_{t+1}} r(\tau)\, \mathrm{d}\tau.$$

is available to the RL algorithm. Let us denote by X_t a trajectory of state within the time interval $[\tau_t, \tau_{t+1})$

$$X_t = \{x(\tau), \tau \in [\tau_t, \tau_{t+1})\}$$

and by A_t a trajectory of low-level controls within this period, namely

$$A_t = \{a(\tau), \tau \in [\tau_t, \tau_{t+1})\}.$$

Suppose we apply to this problem the original version of IRAWC. The database is filled with quintets

$$\langle x_i, u_i, r_{i+1}, x_{i+1}, \varphi_i \rangle. \tag{6.18}$$

The algorithm increases (or decreases) the density $\varphi(u_i; \widetilde{\theta}(x_i; w_\theta))$ depending on the future reinforcements that u_i leads to. Notice that from the point of view of the algorithm, the fact that its actions u_t becomes a parameter for the low-level controller does not matter. It does make sense, because the future reinforcements are indirectly determined by u_i. However, they are directly determined by the pair of trajectories $\langle X_i, A_i \rangle$. Hence, this is $\langle X_i, A_i \rangle$ that could be made more (or less) likely following the state x_i.

Does this distinction matter? The quintet (6.18) impacts the estimator $\widehat{\Phi}_t(w_\theta, w_V)$ proportionally to the density $\varphi(u_i; \widetilde{\theta}(x_i; w_\theta))$ for the current value of w_θ. Suppose the dimensionality of $\mathcal{U}$ is much larger than the dimensionality of $\mathcal{A}$. That is a reasonable assumption. It may be the case that the action u_i is hardly possible for a current $\widetilde{\theta}(x_i; w_\theta)$, yet the pair $\langle X_i, A_i \rangle$ may actually be generated by a certain u that is likely enough yet very distant from u_i.

The idea of increasing or decreasing the likelihood of the pair $\langle X_i, A_i \rangle$ can not be implemented directly because one can not define a density (or a probability) for an infinite-dimensional entity like $\langle X_i, A_i \rangle$. However, we can introduce a certain finite-dimensional representation of this entity. Suppose there exists a certain function L that maps each trajectory A to a point in a certain space $\mathcal{L}$ whose dimensionality is finite, namely

$$L : \{[0, \Delta) \mapsto \mathcal{A}\} \mapsto \mathcal{L}.$$

Suppose there exists also a mapping

$$L^\dagger : \mathcal{L} \mapsto \{[0, \Delta) \mapsto \mathcal{A}\}$$

such that an application of the low-level control A_i and $L^\dagger(L(A_i))$ after the state x_i produces almost the same r_{i+1} and X_i.

How can we define such L and $L^\dagger$? For instance, we may employ the sampling technique used in many areas of engineering. Namely, we define

$$L(A_t)^T = [a(\tau_t + \delta_1)^T, \dots, a(\tau_t + \delta_n)^T] \tag{6.19}$$

where $\{\delta_i, i = 1, \dots, n\}$ is a set of real numbers spread evenly over the interval $[0, \Delta)$. Intuitively, if we sample a function $a : [0, \Delta) \mapsto \mathbb{R}^m$ densely enough, the samples enable to recreate the function. Nyquist theorem defines sufficient conditions for this intuition to become true.

Let us call $L(A)$ a *label* of the trajectory A. Suppose we fix a trajectory of states X and a parameter θ. We know the density $\varphi(\cdot\,; \theta)$ that generates u and the function

C that produces a trajectory A of actions on the basis of X and u. We may define a distribution of labels that this mechanism may produce for this X and θ. Let us denote this distribution by a density function $\varphi^L(L; X, \theta)$ parameterized by the trajectory X of states and θ. In (6.16) we assumed that state dynamics depends on control deterministically. We also assumed that the label $L(A)$ is enough to recreate the control trajectory A. Hence, changes of the density $\varphi^L(L(A); X, \theta)$ caused by manipulating θ induce proportional changes of a likelihood of the pair of trajectories $\langle A, X\rangle$.

A label $L(A)$ may be thought of as a compact representation of the actions A. Let $H(X)$ be a corresponding compact representation of the states X that is enough to compute $\varphi^L(L; X, \theta)$. E.g. if $L(A)$ is equal to selected elements along the control trajectory A (6.19), then $H(X)$ may be equal to corresponding selected elements of the state trajectory X:

$$H(X_t)^T = [x(\tau_t + \delta_1)^T, \dots, x(\tau_t + \delta_n)^T]. \tag{6.20}$$

In order to calculate the probability that for a given state trajectory X our control generation mechanism would generate a given label L (6.19), we need not the entire X but only $H(X)$ (6.20). We will thus write $\varphi^L(L; X, \theta) = \varphi^L(L; H(X), \theta)$.

The algorithm we propose in this section is similar to IRAWC in the sense that it stores in a memory an entire history of a plant-controller interactions. However, instead of remembering a state x_t, an action u_t, and a density φ_t, it stores

$$\begin{aligned} H_t &= H(X_t) \\ L_t &= L(A_t) \\ \varphi_t^L &= \varphi^L(L_t; H_t, \widetilde{\theta}(x_t; w_\theta)) \end{aligned}$$

respectively. In its internal loop, it increases the density $\varphi^L(L_i; H_i, \widetilde{\theta}(x_i; w_\theta))$ of labels associated with control trajectories that lead to large values of the future rewards

$$q_i = r_{i+1} + \gamma \widetilde{V}(x_{i+1}; w_V)$$

and decreases a density of labels that lead to small values of q_i. Instead of optimization of $\widehat{\Phi}_t$ and $\widehat{\Psi}_t$ as IRAWC does, the algorithm optimizes

$$\begin{aligned} \widehat{\Phi}_t^L(w_\theta, w_V) = \frac{1}{t}\sum_{i=1}^t & \widetilde{V}(x_i; w_V) + (q_i - \widetilde{V}(x_i; w_V)) \times \\ & \times \rho(\varphi^L(L_i; H_i, \widetilde{\theta}(x_i; w_\theta)), \varphi_i^L) \\ \widehat{\Psi}_t^L(w_V, w_\theta) = \frac{1}{t}\sum_{i=1}^t & \big(q_i - \widetilde{V}(x_i; w_V)\big)^2 \times \\ & \times \rho(\varphi^L(L_i; H_i, \widetilde{\theta}(x_i; w_\theta)), \varphi_i^L). \end{aligned}$$

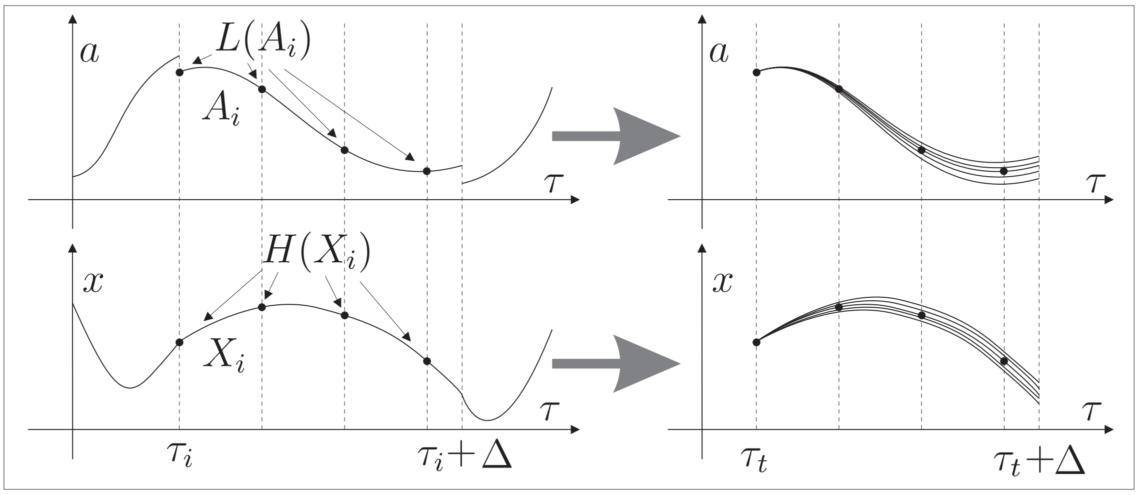

Figure 6.1: The idea of ILRAWC. $L(A_i)$ is a certain approximate representation of A_i. Let for a certain u_t and a given $H = H(X_i)$ Actor generates $L = L(A_i)$. That means that starting from $x_t = x_i$ it would generate certain X_t and A_t *similar* to X_i and A_i, respectively.

Let us call the resulting algorithm Intensive Labelled RAWC or ILRAWC. Figure 6.1 presents its idea. The algorithm increases (or decreases) the probability of $L(A_i)$ given $H(X_i)$. Consequently, it increases (or decreases) the probability of a pair of bunches of trajectories "similar" to X_i and A_i. The denser the labelling is, the "thicker" the bunches are. The experiments presented in Chapter 7 suggest that, in fact, the labelling does not have to be very dense.

ILRAWC can easily be combined with all the ideas presented earlier in this chapter, namely

- Replacing q_i with q_i^* (6.9) yields Labelled Intensive RAWC(λ) or ILRAWC(λ).
- Confining the database to constant number of the most recent events gives Semi-Intensive Labelled RAWC or SILRAWC.
- Combining ILRAWC with both above modifications yields Semi-Intensive Labelled RAWC or SILRAWC(λ).

Chapter 7

Experimental study

In the present dissertation we propose a family of long-term-memory Reinforcement Learning methods. The subject of this chapter is to verify their behavior empirically. Are these methods feasible? Are they faster or slower than their short-term-memory equivalents? Do they obtain better or worse policies? In this chapter we report experiments that shad light on these questions. Obviously empirical results do not suffice to identify all advantages and disadvantages of algorithms, however, they can provide some clues.

We apply all the previously discussed algorithms to three non-trivial control problems. Each of them could be solved by means of existing RL methods. However, the amount of time the existing methods would need, is so large that it is questionable whether anyone could seriously consider their application for solving these problems.

The experiments reported in this chapter are simulated. They imitate a situation where control of an unknown plant is to be optimized with the use of trials and errors in real time. We accept the fact that before control is optimized, it is inappropriate — a controller has not yet learned to behave properly. However, the period of inappropriate control is costly (the plant may be damaged) and we are interested in its minimization.

7.1 Algorithms' implementations

In this study we compare experimentally behavior of various algorithms. The objectivity of a comparison is sometimes difficult to guarantee, since one could always set parameters that improve performance of one algorithm and deteriorate performance of the other one. We thus formulated several rules concerning our experiments.

- Whenever a neural network is employed by an algorithm, it is implemented as a two layer perceptron with sigmoidal (arctan) activation functions in the hidden layer and the linear output layer. Each neuron of both layers has a constant input (bias). The initial weights of the hidden layer are drawn randomly from the normal distribution $N(0, 1)$ (d is the network input dimension) and the initial weights of the output layer are set to zero. Neural network's inputs were normalized by dividing them by their assumed (provided initially by an experimenter) standard deviations.

- Whenever two algorithms use parameters that correspond to one another, they have the same values in both algorithms applied to the same problem. It usually yields good behavior of both methods but there are several exceptions and one of them is regular. Step parameters used in long-term-memory methods are always 3 times smaller than corresponding step parameters in their short-term-memory equivalents.

- The optimal value of a parameter of an algorithm usually depends on the problem that the algorithm is applied to. However, experimental studies (including our own) suggest that in the case of some parameters there are their values that work well in almost each application. We do not optimize values of such parameters but use ones suggested by the studies. We always apply $\lambda = 0.5$, $\lambda_{\max} = 0.5$, $\Upsilon_{\max} = 5$, and the upper bound of ρ equal to 5.

Usually results in the field of Reinforcement Learning are presented in the form of learning curves (average reinforcement vs. control trial number). They are to imply that an algorithm optimizes control of a plant as fast as the curve shows. Usually however, the learning process depicted by the curve is a result of a number of preceding experiments that aim at human-hand-made parameters' selection. Even though it is a common practice, it is a little dishonest to say that an algorithm learns something in, say, thousand trials when its parameters have been optimized in tens of thousands of trials. We follow this practice reluctantly leaving methods of automatic optimization of such free parameters for future work.

In the below experiments three groups of parameters are optimized manually: (i) the ones concerning structures of approximators, i.e. numbers of neurons in neural networks, (ii) the ones concerning control randomization, and (iii) step parameters. Values of all the others are standard or chosen as "reasonable" and not optimized.

7.1.1 RAWC and RAWC(λ)

Implementations of RAWC and RAWC(λ) follow exactly their original formulations (Tables 4.1 and 4.2 respectively). The parameters of RAWC that need to be specified are gathered in Table 7.1. M^θ is to be specified only if Actor is implemented by a neural network. One must also set the value of the discount parameter γ but this is

φ	smoothly-exploring distribution
M^θ	number of hidden neurons of $\widetilde{\theta}$
M^V	number of hidden neurons of $\widetilde{V}$
β_t^θ	step parameter of $\widetilde{\theta}$
β_t^V	step parameter of $\widetilde{V}$

Table 7.1: Parameters of the RAWC algorithm's implementation.

associated with a plant rather than with a learning control method. For each plant we discuss in this study we set parameters for RAWC(λ) equal to corresponding parameters of RAWC. Additionally, for RAWC(λ) one must specify the λ parameter and we always set $\lambda = 0.5$.

7.1.2 IRAWC

An implementation of RAWC requires several design decisions. First, one has to specify the ρ function (3.12). In all experiments we use the same

$$\rho(d, d_0) = \min\left\{\frac{d}{d_0}, 5\right\}.$$

More important, one has to define a process that optimizes $\widehat{\Phi}_t$ and $\widehat{\Psi}_t$ and assure keeping of $\widetilde{\theta}(x_i; w_\theta)$ within a bounded area (see remarks on page 59). Our implementations will be the simplest possible. Namely they will optimize $\widehat{\Phi}_t$ and $\widehat{\Psi}_t$ with the use of stochastic approximation and constant step parameters (see remarks on page 64). Let g_i^Φ be a gradient of i-th component of $\widehat{\Phi}_t$ (5.11) on $\widetilde{\theta}(x_i; w_\theta)$. That is

$$g_i^\Phi = \left.\frac{\mathrm{d}\rho(\varphi(u_i; \theta), \varphi_i)}{\mathrm{d}\theta}\right|_{\theta = \widetilde{\theta}(x_i; w_\theta)}.$$

Pure optimization of $\widehat{\Phi}_t$ by means of stochastic approximation would imply modification of $\widetilde{\theta}(x_i; w_\theta)$ along g_i^Φ for randomly chosen i-s. However, we must employ an additional trick in order to keep $\widetilde{\theta}(x_i; w_\theta)$ within a bounded area. We assume that this bounded area is a cube $[\theta_1^m, \theta_1^M] \times \cdots \times [\theta_{\dim\Theta}^m, \theta_{\dim\Theta}^M]$ and simply replace g_i^Φ with $g_i = G(g_i^\Phi, \widetilde{\theta}(x_i; w_\theta))$, where

$$G_j(g, \theta) = \begin{cases} g_j & \text{if } \theta_j \in [\theta_j^m, \theta_j^M] \\ \max\{g_j, c\} & \text{if } \theta_j < \theta_j^m \\ \min\{g_j, -c\} & \text{if } \theta_j > \theta_j^M \end{cases}, \tag{7.1}$$

subscript j denotes j-th element of an appropriate vector and c is some positive constant. Table 7.2 describes a generic implementation of the IRAWC algorithm used in this study.

```
0. Set $t := 1$ and $i := 1$. Initialize $w_\theta$ and $w_V$ randomly.
1. Draw the control action $u_t$:
       $u_t \sim \varphi(\cdot; \widetilde{\theta}(x_t; w_\theta))$
2. Perform the control action $u_t$, and observe the next state $x_{t+1}$ and the reinforcement $r_{t+1}$.
3. Add the five $\langle x_t, u_t, r_{t+1}, x_{t+1}, \varphi_t \rangle$ to the database $D$ where $\varphi_t := \varphi(u_t; \widetilde{\theta}(x_t; w_\theta))$.
4. (The internal loop) Repeat $\min\{n_1 \# D, n_\infty\}$ times:
   (a) Calculate $d_i = r_{t_i+1} - \widetilde{V}(x_{t_i}; w_V)$ if $x_{t_i+1}$ was not acceptable and $d_i = r_{t_i+1} + \gamma \widetilde{V}(x_{t_i+1}; w_V) - \widetilde{V}(x_{t_i}; w_V)$ otherwise.
   (b) Adjust the approximation of the value function:
       $w_V := w_V + \beta_i^V d_i \rho(\varphi(u_{t_i}; \widetilde{\theta}(x_{t_i}; w_\theta)), \varphi_{t_i}) \times \times \frac{\mathrm{d}\widetilde{V}(x_{t_i}; w_V)}{\mathrm{d}w_V}$
   (c) Adjust the policy:
       $g_i = G\left( \frac{\mathrm{d}\rho(\varphi(u_{t_i}; \widetilde{\theta}(x_{t_i}; w_\theta)), \varphi_{t_i})}{\mathrm{d}\widetilde{\theta}(x_{t_i}; w_\theta)}, \widetilde{\theta}(x_{t_i}; w_\theta) \right)$
       $w_\theta := w_\theta + \beta_i^\theta d_i \frac{\mathrm{d}\widetilde{\theta}(x_{t_i}; w_\theta)}{\mathrm{d}w_\theta} g_i$
   (d) Set $i := i + 1$.
5. Set $t := t + 1$ and repeat from Step 1.
```

Table 7.2: The implementation of the IRAWC algorithm. $\#D$ denotes the number of elements (fives) in the database D.

φ	smoothly-exploring distribution
M^θ	number of hidden neurons of $\widetilde{\theta}$
M^V	number of hidden neurons of $\widetilde{V}$
β_i^θ	step parameter of $\widetilde{\theta}$
β_i^V	step parameter of $\widetilde{V}$
n_∞	max. absolute intensity of the internal loop
n_1	max. intensity of the internal loop relative to the current cardinality $\#D$ of the dataset D.

Table 7.3: Parameters of IRAWC implementation.

The sequence $\{t_i, i \in N\}$ is a concatenation of random permutations of time indexes available in the dataset D (this technique, called "reshuffling", is a "better" form of simple drawing a random sample from the data).

Table 7.3 gathers all the parameters of the implementation of IRAWC. We use the largest n_∞ value still enabling for our simulations (PC with Athlon$^{\text{TM}}$1400 MHz) to be carried in real time of a plant. While the plant is really simulated and n_∞ of any size may be set, we want to check IRAWC's behavior when run on an ordinary PC and confronted with a real plant.

7.1.3 IRAWC(λ), and SIRAWC, SIRAWC(λ)

Implementation of IRAWC(λ) requires a slight modification of the algorithm presented in Table 7.2. Namely, d_i must be calculated as

$$d_i = \widehat{q}^*_{t_i} - \widetilde{V}(x_{t_i}; w_V).$$

where $\widehat{q}^*_{t_i}$ is calculated with use of the recursion discussed in Section 6.2. Additional parameters are needed and they are $\lambda_{\max}$ (in our experiments always equal to 0.5) and $\Upsilon_{\max}$ (always equal to 5).

The implementation of the SIRAWC algorithm almost exactly follows Table 7.2 except it keeps in memory only n_2 most recent events. In the implementation of SIRAWC(λ) there is nothing surprising, it is similar to the implementation of IRAWC(λ) yet only n_2 most recent events are kept in the database.

7.2 Cart-Pole Swing-Up

The Cart-Pole Swing-Up problem is an issue of optimal control of a robotic manipulator, which has two degrees of freedom. One of them is controlled and the other is free. The manipulator consists of a cart moving along a track and a pole hanging freely from the cart. Control signal defines force applied to the cart. The control objective is to avoid hitting the track bounds, swing the pole, turn it up, and stabilize upwards. The problem is described in detail in Appendix B.

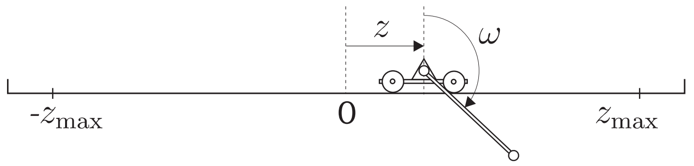

Figure 7.1: The Cart-Pole Swing-Up.

7.2.1 Controller and Critic

We design a controller for the Cart-Pole Swing-Up in the most straightforward way. The function $\widetilde{\theta}(x; w_\theta)$ is represented by a neural network. It produces a center of the distribution that an action is drawn from. We employ the family of normal distributions $N(\theta, 4)$ as φ. The action (limited to the interval $[-10, 10]$) is equal to the force F_t applied to the cart every 0.1 sec.

$$F_t = \begin{cases} -10 & \text{if } u_t < -10 \\ u_t & \text{if } u_t \in [-10, 10] \\ 10 & \text{otherwise} \end{cases}$$

Both $\widetilde{\theta}$ and the Critic approximator $\widetilde{V}$ employed by IRAWC are implemented by neural networks. The state vector x_t feeding the approximators is normalized, namely

$$x_t = \left[\frac{z}{2}, \frac{\dot{z}}{3}, \frac{\sin\omega}{0.8}, \frac{\cos\omega}{0.8}, \frac{\dot{\omega}}{4}\right]^T$$

Note that the plant's state evolves in real time, different than discrete time t.

7.2.2 Reinforcements and training

A typical reward used in the Cart-Pole Swing-Up problem is equal to the elevation of the pole top. Initially, the waving pole hovers, and the rewards are close to -1. When the goal of control is reached and the pole is stabilized upwards, the rewards are close to 1. The state of the plant is *acceptable* if and only if $z \in [-2.4, 2.4]$.

Param.	RAWC	RAWC(λ)	SIRAWC	SIRAWC(λ)	IRAWC	IRAWC(λ)
M^θ	20	20	20	20	20	20
M^V	40	40	40	40	40	40
β_i^θ	3.10^{-3}	3.10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
β_i^V	3.10^{-3}	3.10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
λ		0.5		0.5		0.5
n_∞			10	10	1000	1000
n_1			0.25	0.25	0.25	0.25
n_2			2000	2000		

Table 7.4: Parameters of randomized Actor-Critic algorithms confronted with the Cart-Pole Swing-Up.

The reinforcement is calculated as

$$r_{t+1} = -0.2 \left|u_t - F_t\right| + \begin{cases} -30 & \text{if } x_{t+1} \text{ is not acceptable,} \\ \text{otherwise:} & \\ \cos\omega & \text{if } |\dot\omega| < 2\pi \\ -1 & \text{if } |\dot\omega| \geq 2\pi \end{cases}$$

for ω and $\dot\omega$ measured at the moment $t+1$.

The learning process consists of a sequence of trials. The trial may end in two ways. First, the plant's state may become unacceptable. In this case an appropriately low reinforcement is emitted. Otherwise, the trial lasts for a random amount of time drawn from the exponential distribution with the expected value equal to 20 sec. At the beginning of each trial the state is reset by drawing z and ω from the uniform distributions $U(-2.4, 2.4)$ and $U(0, 2\pi)$, respectively, and setting $\dot z$, $\dot\omega$ to zero.

The discount factor γ is equal to 0.95. The G function keeping $\widetilde{\theta}(x_t; w_\theta)$ within a bounded area is as follows:

$$G(g, \theta) = \begin{cases} g & \text{if } \theta \in [-10, 10] \\ \max\{g, 0.2\} & \text{if } \theta < -10 \\ \min\{g, -0.2\} & \text{if } \theta > 10 \end{cases}$$

7.2.3 Experiments and discussion

We identify parameters of a Cart-Pole Swing-Up controller with the use of two classical algorithms (RAWC and RAWC(λ)) and four algorithms introduced here (IRAWC, IRAWC(λ), SIRAWC, SIRAWC(λ)). Table 7.4 contains parameters of the algorithms and Figure 7.2 shows appropriate learning curves. A control policy

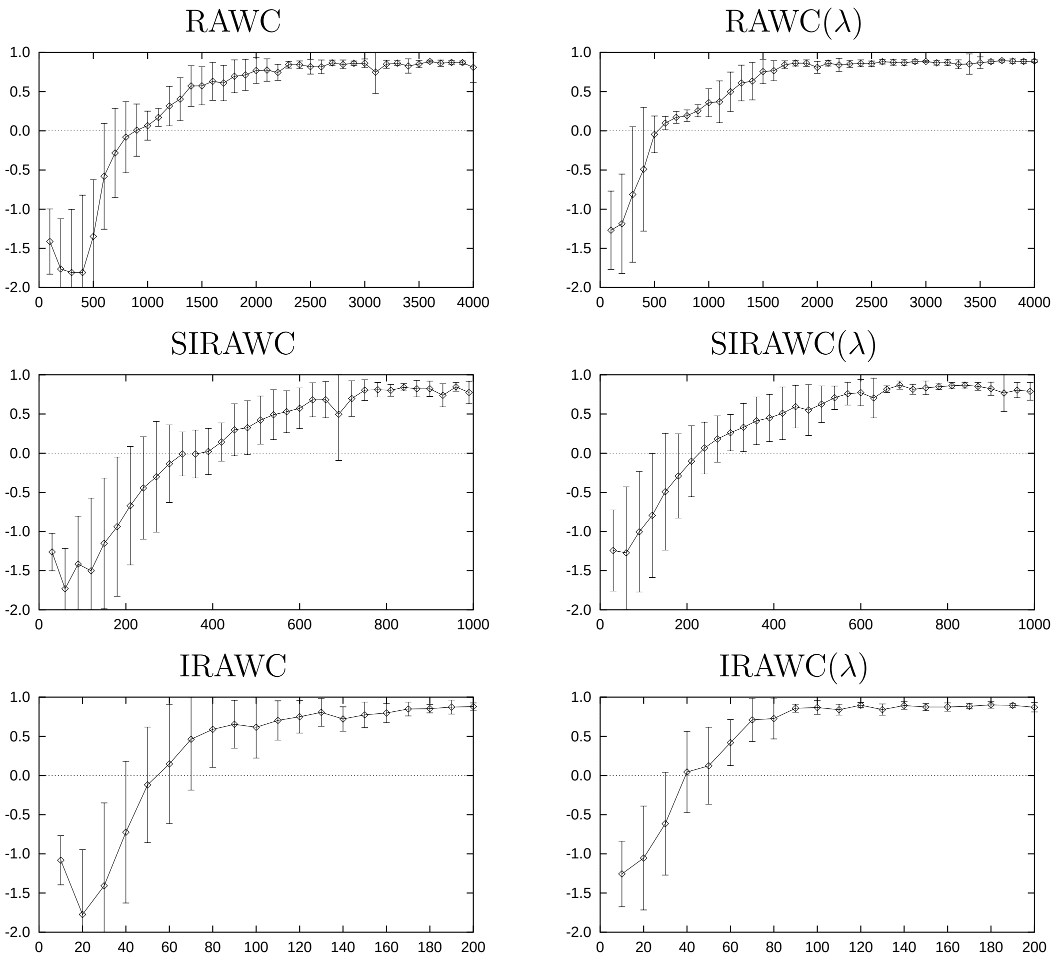

Figure 7.2: Randomized Actor-Critic algorithms applied to the Cart-Pole Swing-Up; average reinforcement vs. trial number. Each point averages reinforcements in a certain number (different for different algorithms) of consecutive trials and each curve averages 10 runs.

for the Cart-Pole Swing-Up may be considered satisfying when it allows a controller to receive average reinforcement 0.75. The traditional algorithm, RAWC obtains such a policy after 2000 trials (about 8 hours of real time of the plant). The basic algorithm we propose, IRAWC, needs only 120 trials (about 30 minutes). It is hence 17 times faster.

The behavior of SIRAWC is a little disappointing. The algorithm obtains a good control policy after about 750 control trials (3 hours). It could seem that the estimator of the gradient of $\Phi(w_\theta)$ calculated on the basis of 2000 most recent examples was quite good. By manipulating the value of the n_1 parameter we control the speed of movement along this gradient. Obviously, because of the limited accuracy of the estimator, n_1 can not be very large. As it turns out, the value greater than 10 deteriorates the final policy performance. For the largest n_1 value that does not deteriorate the final performance, SIRAWC is only about 2.5 times faster than RAWC.

Let us analyze the role of the λ parameter. In the experiments all algorithms applying $\lambda = 0.5$ converge about 1.5 times faster than their original versions. Furthermore, their convergence was more stable. RAWC(λ), SIRAWC(λ), and IRAWC(λ) obtain good control policies after 1500 trials (7 hours), 650 trials (2.5 hours), and 80 trials (20 minutes), respectively. However, the final performance seems not to depend on λ. We use $\lambda > 0$ in order to reduce bias of the estimator of the gradient $\nabla\Phi$ of the quality index. Apparently this bias reduction is more beneficial in early stages of learning than in the end.

Experiments with the Cart-Pole Swing-Up are reported e.g. in [13]. The author of this paper applies a randomized Actor-Critic method similar to RAWC to the plant and obtains almost the same result as reported here. However, the subject of that paper is not the algorithm similar to RAWC. It introduces a model-based algorithm similar to HDP (see Section 2.4.1) which controls the plant well after about 750 trials. Let us compare the methods we propose against model-based algorithms more traditional than the one introduced in [13].

7.2.4 Comparison to Adaptive Critic Designs

There are a few studies checking Adaptive Critic Designs as Reinforcement Learning methods [37, 20], i.e. as methods that learn to control an initially unknown plant. The experiments we describe below concern two methods. The first one is model-free AD-HDP (Table 7.5). The second one is a version of short-memory HDP that learns the model of plant's dynamics on-line (Table 7.6). The algorithm maintains parametric approximators of the model functions R (2.4) and S (2.5). The first one, $\widetilde{R}$ is parameterized by the vector w_R and the second one, $\widetilde{S}$ is parametrized by w_S.

```
Set $t := 1$. Initialize $w_A$ and $w_Q$ randomly.
Repeat until convergence of $w_A$:
    1. Draw $u_t \sim N(\widetilde{A}(x_t; w_A), \sigma^2)$
    2. Apply the control stimulus $u_t$, observe the next state $x_{t+1}$ and the reinforcement $r_{t+1}$
    3. Policy improvement. Adjust $\widetilde{A}(x_t; w_A)$ along $\frac{\mathrm{d}}{\mathrm{d}\widetilde{A}(x_t;w_A)}\widetilde{Q}(x_t, \widetilde{A}(x_t; w_A); w_Q)$:
        $$w_A := w_A + \beta^A_t \frac{\mathrm{d}}{\mathrm{d}w_A}\widetilde{A}(x_t; w_A)\; G\left(\frac{\mathrm{d}\widetilde{Q}(x_t, \widetilde{A}(x_t; w_A); w_Q)}{\mathrm{d}\widetilde{A}(x_t; w_A)}, \widetilde{A}(x_t; w_A)\right)$$
    4. Policy evaluation. Adjust $\widetilde{Q}(x_t, u_t; w_Q)$ towards $r_{t+1}+\gamma\widetilde{Q}(x_t, u_{t+1}; w_Q)$:
        $$q_t := r_{t+1} + \gamma\widetilde{Q}(x_{t+1}, \widetilde{A}(x_{t+1}; w_A); w_Q)$$
        $$w_Q := w_Q + \beta^Q_t \frac{\mathrm{d}\widetilde{Q}(x_t, u_t; w_Q)}{\mathrm{d}w_Q}\left(q_t - \widetilde{Q}(x_t, u_t; w_Q)\right)$$
    5. Set $t := t + 1$ and repeat from Step 1.
```

Table 7.5: The implementation of Action-Dependent HDP.

In both designs the problem occurs that in early stages of learning, when Critics' networks are not trained yet, they drive Actors' networks training in a casual direction. To avoid divergence, $\widetilde{A}(x; w_A)$ must be artificially kept in a bounded area. A simple way to assure this is to employ the G function (7.1) which we use for Actor training in IRAWC.

We identify parameters of a Cart-Pole Swing-Up controller with the use of AD-HDP and On-Line Modeling HDP. The parameters of the algorithms are gathered in Table 7.7 and their values used in the experiments are gathered in Table 7.8. Figure 7.3 shows learning curves obtained by the algorithms when randomized control ($\sigma^2 = 4$ in Tables 7.6 and 7.5) is applied. The algorithms are convergent. Regretfully, we do not present learning curves for the algorithms when they apply deterministic control because they usually do not converge.

In Appendix A we analyze what happens when Adaptive Critic Designs try to identify plant's model and optimize the policy in the same time. We indicate there reasons why ACDs may not converge when deterministic control is in use. The argument is that deterministic control does not allow to gather information about

Set $t := 1$. Initialize w_A, w_V, w_R, and w_S randomly.
Repeat until convergence of w_A:

1. Draw $u_t \sim N(\widetilde{A}(x_t; w_A), \sigma^2)$
2. Apply the control stimulus u_t, observe the next state x_{t+1} and the reinforcement r_{t+1}
3. *Model improvement.*
$$\begin{aligned} w_R &:= w_R + \beta_t^R \frac{\mathrm{d}}{\mathrm{d}w_R} \widetilde{R}(x_t, u_t; w_R) \left(r_{t+1} - \widetilde{R}(x_t, u_t; w_R) \right) \\ w_S &:= w_S + \beta_t^S \frac{\mathrm{d}}{\mathrm{d}w_S} \widetilde{S}(x_t, u_t; w_S) \left(s_{t+1} - \widetilde{S}(x_t, u_t; w_R) \right) \end{aligned}$$
4. *Policy improvement.* For
$$Q(x, u; w_V, w_R, w_S) = \widetilde{R}(x, u; w_R) + \gamma \widetilde{V}(S(x, u; w_S); w_V)$$
adjust $\widetilde{A}(x_t; w_A)$ along $\frac{\mathrm{d}}{\mathrm{d}\widetilde{A}(x_t; w_A)} Q(x_t, \widetilde{A}(x_t; w_A); w_V, w_R, w_S)$:
$$w_A := w_A + \beta_t^A \frac{\mathrm{d}\widetilde{A}(x_t; w_A)}{\mathrm{d}w_A} \, G\left(\frac{\mathrm{d}Q(x_t, \widetilde{A}(x_t; w_A); w_V, w_R, w_S)}{\mathrm{d}\widetilde{A}(x_t; w_A)}, \widetilde{A}(x_t; w_A) \right)$$
5. *Policy evaluation.* Adjust $\widetilde{V}(x_t; w_V)$ towards $Q(x_t, \widetilde{A}(x_t; w_A), w_V, w_R, w_S)$:
$$w_V := w_V + \beta_t^V \frac{\mathrm{d}\widetilde{V}(x_t; w_V)}{\mathrm{d}w_V} \left(Q(x_t, \widetilde{A}(x_t; w_A); w_V, w_R, w_S) - \widetilde{V}(x_t; w_V) \right)$$
6. Set $t := t + 1$ and repeat from Step 1.

Table 7.6: The implementation of On-Line Modeling HDP.

M^A	number of hidden neurons of $\widetilde{A}$
M^Q	number of hidden neurons of $\widetilde{Q}$
M^V	number of hidden neurons of $\widetilde{V}$
M^R	number of hidden neurons of $\widetilde{R}$
M^S	number of hidden neurons of $\widetilde{S}$
β_i^A	step parameter of $\widetilde{A}$
β_i^Q	step parameter of $\widetilde{Q}$
β_i^V	step parameter of $\widetilde{V}$
β_i^R	step parameter of $\widetilde{R}$
β_i^S	step parameter of $\widetilde{S}$
σ^2	variance of the distribution that blur control.

Table 7.7: Parameters of Adaptive Critic Designs.

Param.	AD-HDP	OLM-HDP
M^A	20	20
M^Q	60	
M^V		40
M^R		40
M^S		40
β_i^A	10^{-3}	3.10^{-3}
β_i^Q	0.01	
β_i^V		0.01
β_i^R		0.01
β_i^S		0.01

Table 7.8: Parameters of ACDs applied to the Cart-Pole Swing-Up.

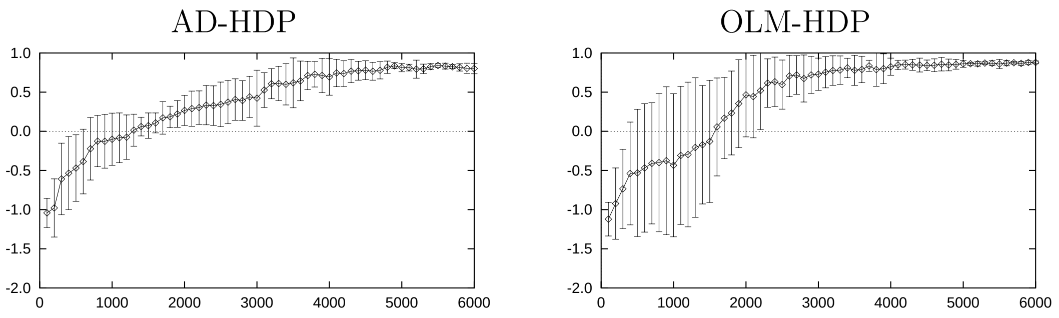

Figure 7.3: Randomized Adaptive Critic Designs applied to the Cart-Pole Swing-Up; average reinforcement vs. trial number. Each point averages reinforcements in 100 consecutive trials and each curve averages 10 runs.

consequences of an application of a policy different that the one in use. The reservations presented in Appendix A check out in our experiments. On the other hand, in [13] it is shown that our reservations do not apply in every case. The algorithm presented in this reference successfully identifies a model of the plant and a control policy in the same time. The experiments we present seem to justify the following conclusion: It may happen that replacing deterministic control with randomized one in Adaptive Critic Design improves their behavior. However, randomized control, once applied, enables one to resign from identification of a plant's model (or approximation of the Q function). Hence, in some cases, ACDs may be successfully replaced with randomized Actor-Critics.

7.3 Robot Weightlifting

The issue we now discuss is learning to control the lift presented at Figure 7.4. From the point of view of robotics, the lift is a manipulator that has three degrees of freedom and each of them is controlled. The plant is described in detail in Appendix B. The control objective is to elevate the load from its lowest position $\omega^T = [\pi, \pi, \pi]$ to its highest position $\omega^T = [0, 0, 0]$ and stabilize it there.

7.3.1 Controller and Critic

In the design of the controller for the lift we follow [33]. The controller is hierarchical; at the lowest level, the plant is controlled by the PD regulator

$$\tau^* = k_c - K_p\omega - K_v\dot{\omega} \tag{7.2}$$

where k_c is a 3-dimensional vector and K_p and K_v are 3x3 matrices calculated every 0.1 sec. by a higher control level. Torques τ applied to the joints are calculated

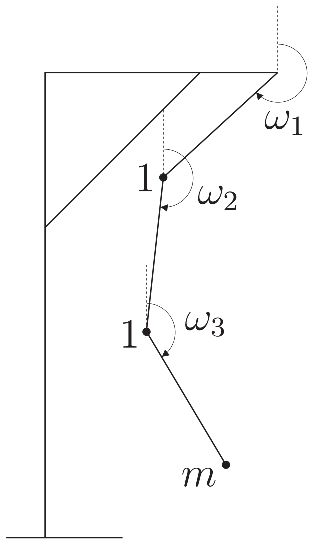

Figure 7.4: The lift used in the Robot Weightlifting problem.

by restricting τ^* to the cube $[-50, 50]^3$. The idea presented in [33] is that in order to control the lift properly, it is enough to combine two PD regulators: the "bending" one $\langle k_c^B, K_p^B, K_v^B \rangle$, which operates when the load is below the first joint and "straightening" one $\langle k_c^S, K_p^S, K_v^S \rangle$, which operates when the load is above. Obviously $k_c^S \equiv [0, 0, 0]^T$. We will gradually switch between these two regulators depending on the elevation $y_3(\omega)$ of the load:

$$k_c \cong k_c^B w(\omega) \tag{7.3}$$

$$K_p \cong K_p^B w(\omega) + K_p^S(1 - w(\omega)) \tag{7.4}$$

$$K_v \cong K_v^B w(\omega) + K_v^S(1 - w(\omega)) \tag{7.5}$$

where

$$w(\omega) = 0.5(1 - \sin(\pi y_3(\omega)/3))$$
$$y_3(\omega) = \cos\omega_1 + \cos\omega_2 + \cos\omega_3$$

The approximate equalities "$\cong$" in (7.3) – (7.5) come from the fact that Actor does not impose k_c, K_p, and K_v directly, but generates them from normal distributions with the centers given by the right hand sides of the approximate equations.

Actor's policy $\pi(w_\theta)$ manifests itself in generating k_c, K_p, and K_v every 0.1 second. $\widetilde{\theta}$ is described by right hand sides of equations (7.3) – (7.5). The equations produce $3 + 9 + 9 = 21$-element θ vector. Vector k_c^B and matrices K_p^B, K_v^B, K_p^S, and K_v^S, form a 39-element vector w_θ of Actor's parameters. Actions are drawn from $N(\theta, \sigma^2 I)$, where I is the 21-dimensional unit matrix.

We have some preliminary knowledge of Actor's parameters. We know that (7.2) should produce values in the cube $[-50, 50]^3$, hence each Actor's parameter should be roughly kept within the interval $[-100, 100]$. Furthermore, the logic of PD regulator implies that matrices K_v^B and K_v^S are positively defined. We set their initial values to $50\,I$. All the other parameters we initialize with zero values.

The $\widetilde{V}$ approximator employed by the algorithm is a neural network. A state vector x_t feeding the $\widetilde{V}$ approximator is normalized, namely

$$x_t = \left[\frac{\omega_1}{3}, \frac{\dot\omega_1}{4}, \frac{\omega_2}{3}, \frac{\dot\omega_2}{4}, \frac{\omega_3}{3}, \frac{\dot\omega_3}{4}\right]$$

7.3.2 Reinforcements and training

Reinforcement in this problem involves a penalty for difference between the final position and the actual one. We may consider Euclidean distance

$$r_{t+1,1} = -d_1(\omega) - d_2(\omega) - d_3(\omega)$$

where d_i is a cartesian distance between i-th joint and its final position. We may also consider the angular one

$$r_{t+1,2} = -|\omega_1| - |\omega_1| - |\omega_1|.$$

The main part of reinforcement is calculated as a penalty for hitting the platform and an average penalty for distance from the desired position, namely

$$r_{t+1,3} = \begin{cases} -100 \text{ if } x_{t+1} \text{ is not acceptable, otherwise:} \\ 10 - \frac{5}{12} r_{t+1,1} - \frac{5}{3\pi} r_{t+1,2} \end{cases}$$

Additionally, the controller is penalized when Actor's parameters exceed their bounds

$$r_{t+1,4} = -0.03 \|\widetilde{\theta}(x_{t+1}; w_\theta) - (\text{throw of } \widetilde{\theta}(x_{t+1}; w_\theta) \text{ on } [-100, 100]^{21})\|_1.$$

It is also penalized when $\omega_2 < \omega_1$ or $\omega_3 < \omega_2$ for the load below the first link

$$r_{t+1,5} = \begin{cases} 0 \text{ if } y_3(\omega_{t+1}) >= 0, \text{ otherwise:} \\ -\max\{0, \omega_1 - \omega_2, \omega_2 - \omega_3\} \end{cases}$$

Without this last penalty, the policy is occasionally getting stuck in a local minimum. Finally

$$r_{t+1} = r_{t+1,3} + r_{t+1,4} + r_{t+1,5}$$

The learning process consists of a sequence of *trials*. The trial begins with $\omega = [\pi, \pi, \pi]^T$, $\dot\omega = [0, 0, 0]^T$, and m is drawn from the uniform binary distribution $\{1.5, 3.5\}$ (see Fig. 7.4). The trial may end in one of two possible situations. First,

Param.	RAWC	RAWC(λ)	SIRAWC	SIRAWC(λ)	IRAWC	IRAWC(λ)
M^V	100	100	100	100	100	100
β_i^θ	10^{-1}	10^{-1}	3.10^{-2}	3.10^{-2}	3.10^{-2}	3.10^{-2}
β_i^V	10^{-3}	10^{-3}	3.10^{-4}	3.10^{-4}	3.10^{-4}	3.10^{-4}
λ		0.5		0.5		0.5
n_∞			10	10	500	500
n_1			0.25	0.25	0.25	0.25
n_2			2000	2000		
			SILRAWC	SILRAWC(λ)	ILRAWC	ILRAWC(λ)
M^V			100	100	100	100
β_i^θ			2.10^{-2}	2.10^{-2}	2.10^{-2}	2.10^{-2}
β_i^V			2.10^{-4}	2.10^{-4}	2.10^{-4}	2.10^{-4}
λ				0.5		0.5
n_∞			10	10	500	500
n_1			0.25	0.25	0.25	0.25
n_2			2000	2000		

Table 7.9: Parameters of algorithms applied to the Robot Weightlifting.

the arm's state becomes unacceptable — it simply hits the platform or itself. This is associated with an appropriately low reinforcement. Second, the trial lasts for a random real time drawn from the exponential distribution with the expected value equal to 10 sec.

The discount factor γ is set to 0.95. The G function keeping $\widetilde{\theta}(x_t; w_\theta)$ within a bounded area is as follows:

$$G_j(g,\theta) = \begin{cases} g_j & \text{if } \theta_j \in [-100, 100] \\ \max\{g_j, 0.025\} & \text{if } \theta_j < -100 \\ \min\{g_j, -0.025\} & \text{if } \theta_j > 100 \end{cases}$$

In the case of the ILRAWC algorithm we employed the labelling defined in (6.19) and (6.20), where $n = 2$, $\delta_1 = 0$, and $\delta_2 = \Delta/2$.

7.3.3 Experiments and discussion

We identify parameters of a Robot Weightlifting controller with the use of two classical algorithms (RAWC and RAWC(λ)) and eight algorithms introduced here (IRAWC, IRAWC(λ), SIRAWC, SIRAWC(λ), ILRAWC, ILRAWC(λ), SILRAWC, and SILRAWC(λ)). Table 7.10 contains parameters of all algorithms applied to the Robot Weightlifting problem and Figure 7.5 shows appropriate learning curves. A

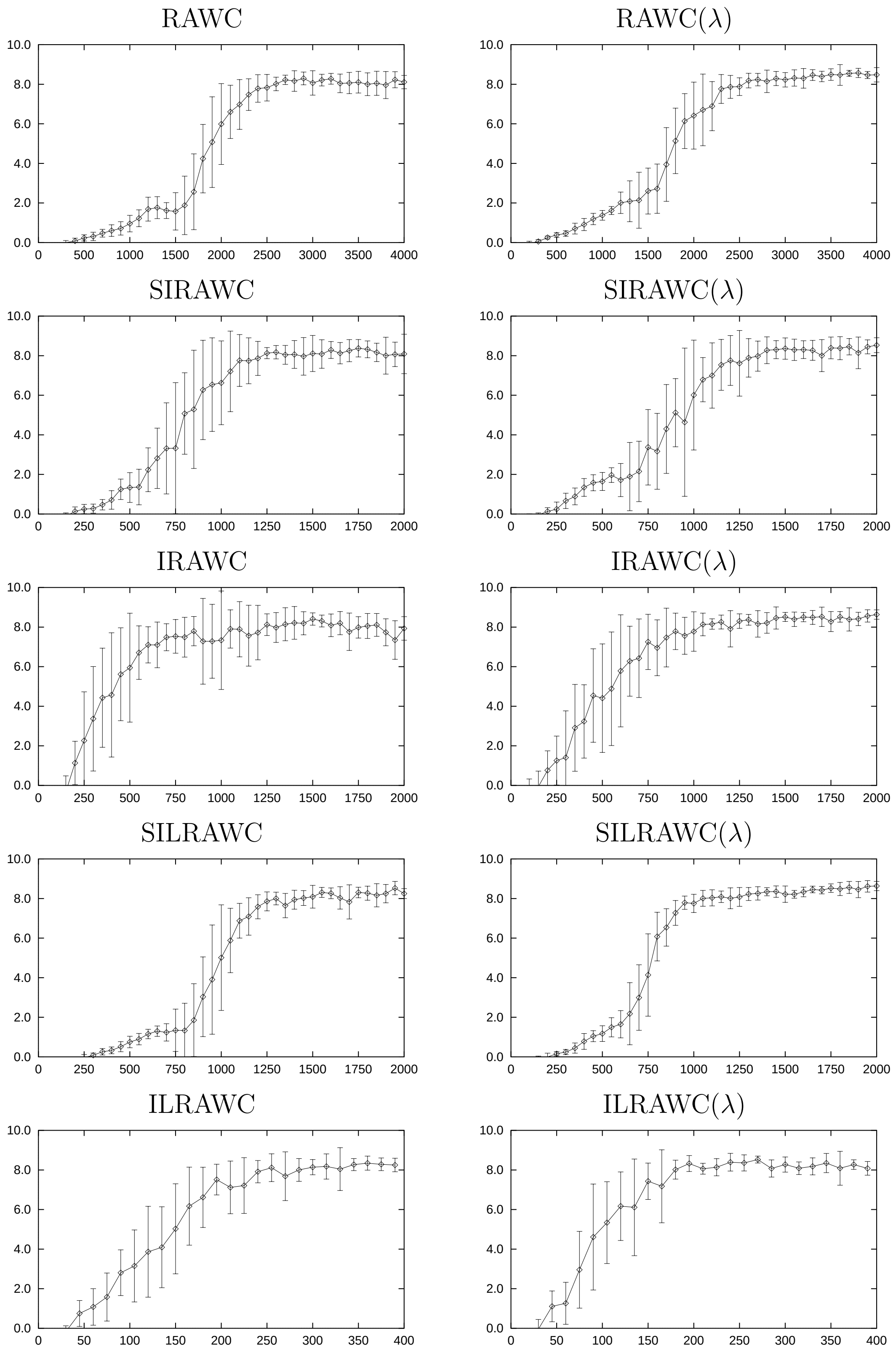

Figure 7.5: Randomized Actor-Critic algorithms applied to the Robot Weightlifting; average reinforcement vs. trial number. Each point averages reinforcements in a certain number (different for different algorithms) of consecutive trials and each curve averages 10 runs.

control policy for the plant may be considered satisfying when it allows a controller to receive average reinforcement 7.5. The traditional algorithm, RAWC obtains such a policy after 2300 trials (about 4.5 hours of real time of the plant). Intensive RAWC (IRAWC) needs 700 trials (about 1.5 hours, 3 times faster). Semi–Intensive RAWC (SIRAWC) needs 1200 trials (about 2 hours), and Intensive Labelled RAWC (ILRAWC) needs only 230 trials (27 minutes, 10 times less than RAWC).

In this experiments the RAWC, SIRAWC, and IRAWC algorithms search for 21-elements sets of parameters for two PD controllers. What RAWC and SIRAWC try to do is a local steepest descent in both spaces of parameters. What IRAWC does may be rather understood as a thorough search of both spaces. Because the spaces are large, their thorough search must be slow. ILRAWC tries to do something else, it searches thoroughly the 3-dimensional space of first low-level controls in each state. Because the dimensionality of the space to search is much smaller, there is nothing surprising that the algorithm works much faster. The idea of labels that the algorithm is based on seems to work very well.

Algorithms with λ-estimators of future rewards converge at least as fast as their basic equivalents. E.g. ILRAWC(λ) obtains a good control policy after 170 trials which translates into 20 minutes of real time of the plant.

Our results can also be compared with those presented in [33], where a little different training was applied. The training based on the Simple Random Search method started when a control policy was able to lift 0.5 kg. The weights were increased gradually from trial to trial to train the system to lift the largest possible load. A single run ended after 5000 trials when the plant was ready to lift as much as 7 kg. Obviously this is a more difficult task, also because it is not stationary. In this type of problems remembering previous events does not help very much and IRAWC nor ILRAWC in their pure forms are not applicable.

7.4 Narendra's MIMO

Narendra's MIMO (multiple-input-multiple-output) problem is an issue of control of a certain artificial discrete time plant. The plant is known as a testbed for many algorithm discussed in the literature on Approximate Dynamic Programming (see Section 2.4). The control objective is for the plant's output to follow a certain path. The objective is difficult to achieve because of the highly nonlinear dynamics of the plant. The MIMO problem is described in detail in Appendix B.

7.4.1 Controller and Critic

We design a controller for Narendra's MIMO in the most straightforward way. $\widetilde{\theta}(x; w_\theta)$ function is implemented by a neural network. It produces a center of the (normal) distribution that an action is drawn from.

Approximators $\widetilde{\theta}$ and $\widetilde{V}$ employed by IRAWC are implemented as neural networks. Action u_t, which is normalized to cube $[-1,1]^2$, is scaled to control stimuli:

$$\begin{aligned} a_{t,1} &= 0.8 u_{t,1} \\ a_{t,2} &= 0.3 u_{t,2} \end{aligned}$$

The state vector x_t feeding the approximators is normalized, namely

$$x_t = \frac{1}{2}\left[z_{t,1}, z_{t,2}, z_{t,3}, z^d_{t+1,1}, z^d_{t+1,2}\right]^T$$

7.4.2 Reinforcements and training

We employ reinforcement equal to a negative sum of penalties:

$$\begin{aligned} r_{t+1} = \;& -\left(z_{t+1,1} - z^d_{t+1,1}\right)^2 && //z_{t+1,1} \text{ differs from its desired value} \\ & -\left(z_{t+1,2} - z^d_{t+1,2}\right)^2 && //z_{t+1,2} \text{ differs from its desired value} \\ & -0.1\max\{|u_{t,1}| - 1, 0\} && //u_{t,1} \text{ exceeds [-1,1] interval} \\ & -0.1\max\{|u_{t,2}| - 1, 0\} && //u_{t,2} \text{ exceeds [-1,1] interval} \end{aligned}$$

The learning process consists of a sequence of *trials*. Each trial is 200 steps long and begin with all the state variables equal to zero.

Evolution of desired values $x^d_{1,t}$ and $x^d_{2,t}$ is driven by the following equations:

$$\begin{aligned} k_{t,1} &= k_{t-1,1} + \xi_{t,1} \\ z^d_{t,1} &= 0.75\sin(\pi k_{t,1}/25) + 0.75\sin(\pi k_{t,1}/5) \\ k_{t,2} &= k_{t-1,2} + \xi_{t,2} \\ z^d_{t,2} &= 0.75\sin(\pi k_{t,2}/15) + 0.75\sin(\pi k_{t,2}/10) \end{aligned}$$

where $\xi_{t,1}$ and $\xi_{t,2}$ are random variables drawn independently from the uniform distribution $U(0,2)$.

The discount factor γ is set to 0.8. The G function keeping $\widetilde{\theta}(x_t; w_\theta)$ within a bounded area is as follows:

$$G_j(g,\theta) = \begin{cases} g_j & \text{if } \theta_j \in [-1,1] \\ \max\{g_j, 0.1\} & \text{if } \theta_j < -1 \\ \min\{g_j, -0.1\} & \text{if } \theta_j > 1 \end{cases}$$

Param.	RAWC	RAWC(λ)	SIRAWC	SIRAWC(λ)	IRAWC	IRAWC(λ)
M^θ	40	40	40	40	40	40
M^V	80	80	80	80	80	80
β_i^θ	3.10^{-6}	3.10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}
β_i^V	3.10^{-2}	3.10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}
λ		0.5		0.5		0.5
n_∞			10	10	500	500
n_1			0.25	0.25	0.25	0.25
n_2			2000	2000		

Table 7.10: Parameters of algorithms applied to the MIMO.

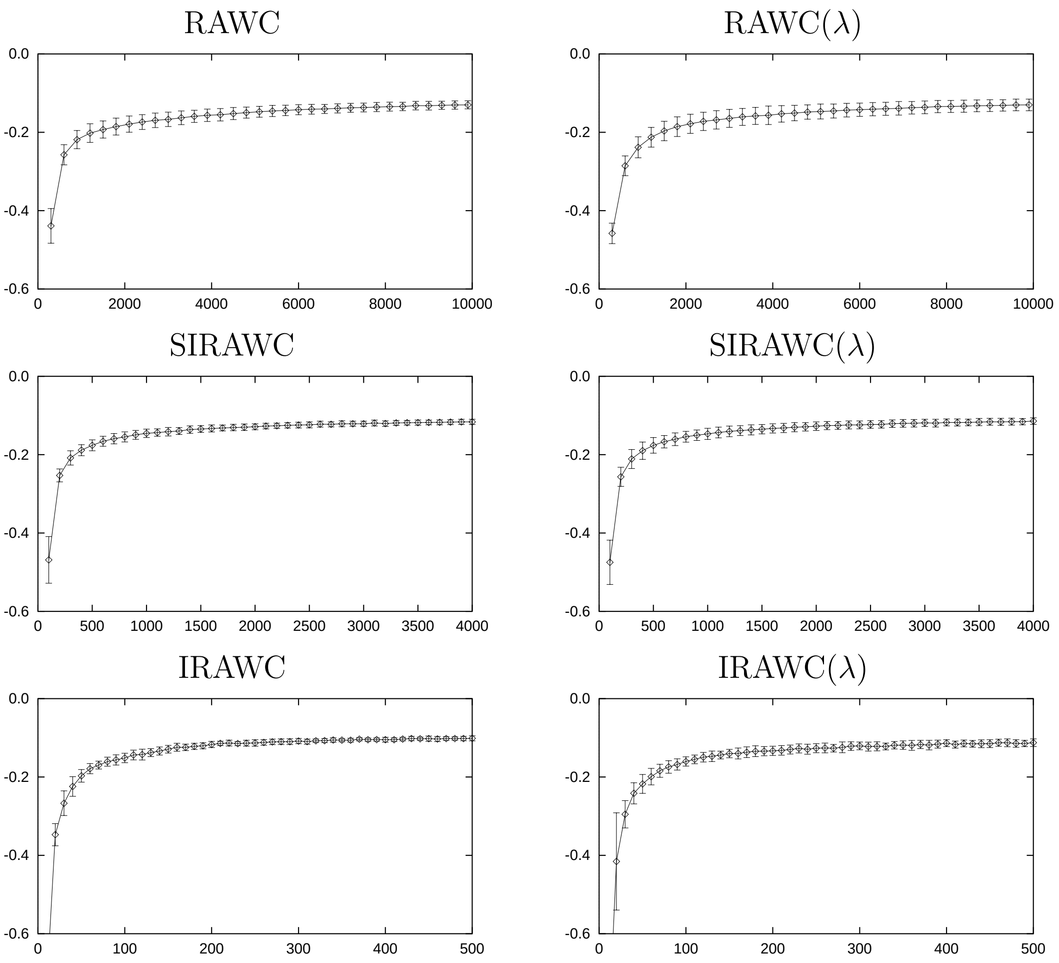

Figure 7.6: Randomized Actor-Critic algorithms applied to the MIMO control problem; average reinforcement vs. trial number. Each point averages reinforcements in a certain number (different for different algorithms) of consecutive trials and each curve averages 10 runs.

7.4.3 Experiments and discussion

We identify parameters of a controller of Narendra's MIMO with the use of two classical algorithms (RAWC and RAWC(λ)) and four algorithms introduced here (IRAWC, IRAWC(λ), SIRAWC, and SIRAWC(λ)). Table 7.10 contains parameters of algorithms and Figure 7.6 shows appropriate learning curves. The curves are almost perfectly regular. They converge asymptotically to certain limits. That corresponds very well to the idea of learning understood as a process of permanent improvement that is never completed.

After about 200 trials IRAWC reaches average reinforcement equal to -0.115. The subsequent improvement proceeds still yet it is insignificant. The traditional algorithm, RAWC does not reach average reinforcement -0.115 even after 10000 trials when we stop experiments. At this moment the controller optimized by the algorithm receives average reinforcement equal to -0.129 which IRAWC reaches after 150 trials. These numbers imply that IRAWC is 67 times faster than RAWC.

The SIRAWC algorithm, the semi-intensive version of IRAWC, reaches average reinforcement equal to -0.115 after 4000 trials (20 times slower than IRAWC) and average reinforcement equal to -0.129 after 2000 trials (13 times slower than IRAWC). Efficiency of SIRAWC is hence comparable to efficiency of RAWC rather than IRAWC.

λ-estimators of future rewards influence behavior of RL algorithms applied to this problem insignificantly. It seems that they do not have an impact on efficiency of RAWC. They improve a little the speed of SIRAWC and even worsen a little the speed of IRAWC.

Our results may be compared with those reported in reference [32]. Authors train there Recurrent Neural Networks (RNNs) and Feedforward Neural Networks (FNNs) to control the MIMO. They proceed accordingly to the following 3-steps procedure:

1. The plant is subjected to random control stimuli. Its responses are registered and the neural model of the plant's dynamics is built on the basis of the gathered observations.

2. A control policy is determined with the use of simulations based on the plant's model. Within the simulations the specialized *skyline* reference model (the model of dynamics of z_t^d) is in use.

3. The performance of the policy is verified with the use of a plant whose parameters are randomly modified ($\pm 20\%$) from the parameters of the plant that served for the model identification. The reference model used in the test is different than the skyline.

The concussion presented in [32] is that RNNs are more robust to changes of plant's parameters than FNNs trained by means of Adaptive Critic Designs. We have conducted similar experiments. The conclusions may be summarized as follows.

1. The robustness of the methods presented here to mismatching of plant's parameters is smaller than the robustness of RNNs and their specialized training method. It is comparable to the robustness of FNNs trained by means of Adaptive Critic Designs.

2. All Actor-Critic algorithms we analyze here require much less real time of the plant to determine a policy than Step 1. of the above procedure required to identify a satisfying model of the plant.

7.5 Summary

We applied short-term-memory and long-term-memory randomized Actor-Critic algorithms to three non-trivial learning control problems. The experiments imitated a situation where control of an unknown plant is to be optimized with the use of trials and errors in real time. We accepted the fact that before control is optimized, it is inappropriate — a controller had not yet learned to behave properly. However, we were interested in minimization of this time.

In all experiments the algorithms introduced here turned out to be at least 10 times faster than their existing equivalents. The basic introduced algorithm, IRAWC needed 120 trials (30 minutes of plant's real time) to learn to control the Cart-Pole Swing-Up. This was 17 times less than the traditional algorithm, RAWC. In a certain comparison IRAWC turned out to learn to control Narendra's MIMO as many as 67 times faster than RAWC. The version of IRAWC designed for continuous time problems, LIRAWC, needed 230 trials (27 minutes) to learn to control the Robot Weightlifting, which was 10 times less than RAWC needed.

It is important to note, that not always switching from Random Actor With Critic (RAWC) to Intensive Random Actor With Critic (IRAWC) shortened the time of learning at least 10 times. In the case of hierarchical control, when a Reinforcement Learning algorithm determines a multi-dimensional parameter for a lower control level, IRAWC might be as few as 2 times faster. In such cases, the idea of labels presented in Section 6.4 worked very well. In the case of the Robot Weightlifting, the ILRAWC algorithm that implemented this idea was 10 times faster than RAWC.

In all cases algorithms based on λ-estimators of future rewards converged faster than their basic equivalents. However, the improvement was not very significant. In the case where the difference was the largest (the Cart-Pole Swing-Up), IRAWC(λ)

converged 1.5 times faster than IRAWC. Furthermore, policies ultimately given by the algorithms based on the λ-estimators were not better than policies obtained by the basic equivalents. λ-estimators were applied in order to reduce bias of the estimator of the gradient $\nabla\Phi$ of the quality index. Apparently this bias reduction was more beneficial in early stages of learning than in the end.

Semi-intensive algorithms discussed in Section 6.3 converged about two times faster than their short-term-memory equivalents. This improvement is rather disappointing. In the course of these experiments, we imitated the situation where a computer's RAM memory could store only 2000 control events. This number of events translated into several control trials. The result we obtained is very significant. We had to decrease intensity of an internal loop in each semi-intensive algorithm. Otherwise, we saw a deterioration of an ultimate policy. The small intensity of the internal loop translated into the slow convergence of the algorithm. The choice of the assumed capacity of the RAM memory (2000 control steps) was obviously arbitrary. Probably if the capacity had been larger, the larger intensity of the internal loop could have been in use. However, conversely to long-term-memory algorithms, the semi-intensive ones are still based on stochastic approximation. Because the estimators they use are good, they may converge faster than the short-term-memory methods, yet the improvement is insignificant.

Chapter 8

Conclusions

This chapter closes the dissertation. By looking backwards at the previous chapters, we summarize what has been contributed by this research and decide whether its goals have been achieved. We also discuss possible directions of future work.

8.1 Contributions

The key contribution of this dissertation was the combination of two ideas. The first one concerned algorithmic issues associated with Reinforcement Learning. All but few RL algorithms are so constructed that they process information on consecutive control steps sequentially, piece by piece. This sequentiality constraint seemed to prevent the algorithms from efficiently exploiting all the information that the control steps carry. The idea was to construct an algorithm that stored all the history of plant-controller interactions in a database and exploited it in a certain computation process that went simultaneously with the control process.

The second idea concerned the nature of the computation process. Its objective was to optimize an estimator of a global performance index parameterized by the policy parameters. This estimator was based on data on plant-controller interactions gathered in the database.

Implementation of the two above ideas yielded the Intensive Random Actor With Critic algorithm (IRAWC) presented in Chapter 5. In Chapter 6 this algorithm was enhanced in several ways. First, the basic estimators of future rewards was replaced with λ-estimators to give the IRAWC(λ) algorithm. Second, the semi-intensive algorithms were presented that combined the introduced methodology with the existing approach. This way SIRAWC and SIRAWC(λ) algorithms arose. Third, the idea of IRAWC was extended to continuous time problems to give ILRAWC and ILRAWC(λ).

The proposed algorithms had the form of certain optimization issues. In Section 5.6 the numerical methods were proposed to tackle these particular issues. The methods were simple yet still giving satisfying behavior. In Section 6.2 it was shown how to handle additional difficulties associated with implementation of the algorithms that used λ-estimators of future rewards.

In Chapter 7 the proposed methods were experimentally compared against popular existing methods (which were called in this dissertation RAWC and RAWC(λ)). In all experiments the introduced methods turned out to be at least 10 times faster than the existing ones. The basic introduced algorithm, IRAWC needed 120 trials (30 minutes of plant's real time) to learn to control the Cart-Pole Swing-Up. This was 17 times less than the traditional algorithm, RAWC, needed. In a certain comparison IRAWC turned out to learn to control Narendra's MIMO as many as 67 times faster than RAWC. The version of IRAWC designed for continuous time problems, LIRAWC needed 230 trials (27 minutes) to learn to control the Robot Weightlifting which was 10 times less than RAWC needed. We claim that this quantitative improvement translates into a qualitative one. Namely, if the process lasts for half-an-hour, control improvement is observable to supervising humans. If the improvement does not take place, the supervisors may adjust the settings of the optimization process until the improvement does take place. However, if the process takes several hours, the improvement is difficult to observe by humans and they are not in control of the process. From the point of somebody who is interested in commercial application of these methods, this difference seems to be crucial. Suppose a factory management wants to get control of a new plant optimized. They may choose between a trial-and-error method and other ones. It is much more likely that they choose the trial-and-error method if they suspect that the optimization process will be finished in half-an-hour and will be fully controlled by a qualified personnel. However, if the situation above can not be guaranteed, the management is likely to consider the trial-and-error optimization method too risky.

The basic fact that the introduced algorithms indeed "remember" all the historical events rises an issue of feasibility. Obviously, there exist such reinforcement learning problems where it is impossible to keep the entire history of plant-controller interactions because it gets too long before the problem is solved. On the other hand, this issue is less serious than it seems to be at first. For example, the original version of IRAWC applied to the Robot Weightlifting usually kept in the memory about 80000 historical events at the moment the policy became satisfying. The data structures took about 25MB of RAM memory. Only if the learning process needed more than 1 day, it would fill 512 MB of RAM memory and in this way would exceed the memory capacity of a typical (at year 2004) PC. However, it is unlikely that a trial-and-error method of control optimization would be used if it took days instead of hours. In other words, we can either guarantee that the control system learning will be completed in a short time and then computer memory limitations do not matter or the method would not be applied at all.

We are now in a position that allows us to discuss theses of this dissertation stated in Chapter 1.

1. It is possible design an RL algorithm based on batch estimation. It is possible to construct an estimator of a global performance index parameterized by the policy parameters. Optimization of this estimator with respect to the parameters leads to the optimization of the performance. An example of algorithm constructed in this way is IRAWC.

2. It is possible to design a fast RL algorithm that does not process data sequentially as they are generated by the control process. Such algorithms may optimize the estimator of the performance index with respect to the policy parameters. They may have properties very similar to those of the methods based on stochastic approximation, yet they exploit available data much more efficiently.

3. We have introduced a family of RL algorithms. They are based on batch estimation and do not process data sequentially. They are feasible: when applied to problems of moderate complexity, they are able to optimize the control policy far before the stored data fill in the memory capacity of a typical computer. The introduced algorithms have the form of certain optimization issues. We have proposed numerical methods to tackle these particular issues. The reported experiments show that the methods are efficient enough to determine control policies in real time of plants.

8.2 Future work

The work on long-term-memory Actor-Critic algorithms is far from being complete and a number of open questions still remain. At the moment, we can see the following paths of future work on long-term-memory randomized Actor-Critics before these algorithms are ready to solve real-life problems. These paths concern (i) hypothetical yet not proved properties of these algorithms, (ii) automatic optimization of their open parameters, and (iii) adaptation of these algorithms to continuous time problems.

In is known that methods similar to RAWC are convergent [45, 17] if Critic remains in a special relation with Actor and is a linear function of its parameters. IRAWC, as a long-term-memory version of RAWC, is designed to reach the same goals, just in a different way. However, a proof of convergence of IRAWC or its modifications is an open problem.

Usually results in the field of Reinforcement Learning are presented in the form of learning curves. They are to imply that an algorithm learns to control a plant as fast as the curve shows. Usually however, such a curve is a result of hundreds of experiments aiming at human-hand-made parameters selection. Even though it is a common practice, it is a little dishonest to suggest that an algorithm learns something in n trials when its parameters were determined in, say, $20n$ trials. Unfortunately we did not avoid following this practice. Long-term-memory Actor-Critics require providing parameters: associated with Actor/Critic training and with control randomization.

We defined long-term-memory Actor-Critics as requiring of solving certain optimization problems. In fact, the generic definition of the algorithms provided in Chapter 5 did not suggest any particular methods for solving these problems. In fact, there are very well developed, fully autonomous optimization techniques that do not require providing any parameters. If Actor and/or Critic are implemented as neural network, one may choose from among a number of methods of fully autonomous network training. However, some of them are better suited for our purposes and some worse. This subject certainly requires further investigation.

The methods introduced in this dissertation require randomization of control selection. Unfortunately it is unknown in advance what should be the amount of randomization that would be big enough for exploration yet little enough not to spoil control. This issue of exploration-exploitation balance is known to be extremely difficult. The methods we introduced here gather all the history of plant-controller interactions in a database. Hypothetically, it is possible to infer from the collected data what should be the amount of randomization of a control selection in each state that would be the most beneficial for the information collected in the data. Traditional, incremental RL methods do not collect such data and keep no information that could be enriched. Hopefully future work on this intuition will shed light on the issue of exploration-exploitation balance.

We presented a certain idea concerning reinforcement learning in continuous time. An implementation of this idea requires introducing hierarchy into the control policy. Within this idea control is not continuous but piecewise continuous. However, it is not always possible nor it is always convenient for a control system to be hierarchical. In many applications it is not feasible for control to be discontinuous. Even though the solution we propose works very well in our experiments, it seems to require a lot of work aiming at its adaptation to the practice of control systems design.

Appendix A

Is reinforcement learning without exploration possible?

The basic form of HDP is not an algorithm of reinforcement learning. It is rather a method of Approximate Dynamic Programming designed to be in use for determining control policy in simulations. There are, however, suggestions [60, 31] that one can replace a constant model of a plant with the one built on-line on the basis of controller-plant interactions. In fact, very few reports on successful implementations of model-learning HDP and model-free ADHDP have been presented. Why their performance may be worse than expected?

Suppose, that instead of functions R and S modeling the plant's dynamics, we approximate those functions basing on observations made in the course of control. We employ approximators $\widetilde{S}$ and $\widetilde{R}$ parametrized by w_S and w_R, respectively, namely

$$\begin{aligned}\widetilde{S}(x,u;w_S) &\cong \mathcal{E}(x_{t+1}|x_t=x,u_t=u)\\ \widetilde{R}(x,u;w_R) &\cong \mathcal{E}(r_{t+1}|x_t=x,u_t=u).\end{aligned}$$

Suppose HDP makes the parameters w_A of the approximator $\widetilde{A}$ converge to some limit vector w^*. Suppose the approximators $\widetilde{S}$ and $\widetilde{R}$ are good enough to be able to fit the training set very precisely. However, in the limit, their training sets consist of tuples $\langle\langle x,u\rangle,x'\rangle$ (and $\langle\langle x,u\rangle,r\rangle$ respectively) such that $u=\widetilde{A}(x;w^*)$. There is no reason to expect that the equations

$$\begin{aligned}\widetilde{S}(x,u;w_S) &\cong \mathcal{E}(x_{t+1}|x_t=x,u_t=u)\\ \widetilde{R}(x,u;w_R) &\cong \mathcal{E}(r_{t+1}|x_t=x,u_t=u)\end{aligned}$$

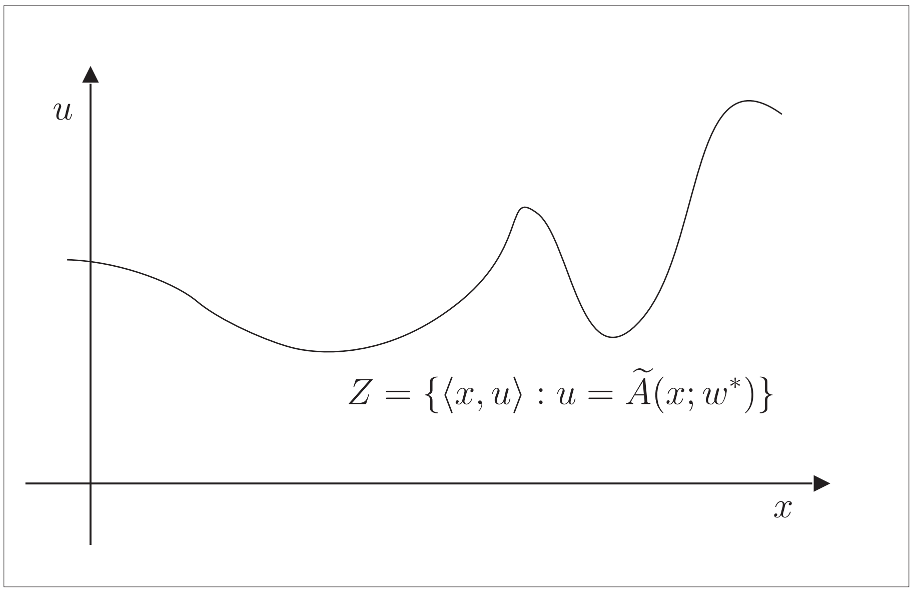

Figure A.1: The domain of functions approximated by $\widetilde{R}$, $\widetilde{S}$, and $\widetilde{Q}$ is the entire state-control space. The set Z of points used to approximate functions $\widetilde{R}$, $\widetilde{S}$, and $\widetilde{Q}$ for HDP algorithms is typically a hyperplane in this space. Since Z is singular (in a the sense of lower dimension than that of the state-control space), the gradients of approximations may not reflect gradients of original functions.

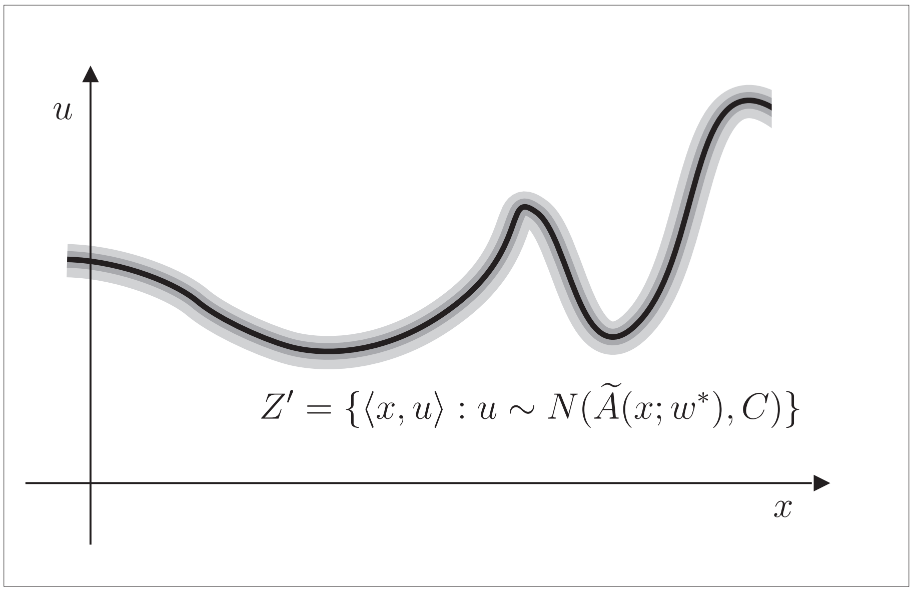
u
$Z' = \{\langle x, u \rangle : u \sim N(\widetilde{A}(x; w^*), C)\}$
x

Figure A.2: Suppose the set Z' of points employed to approximate $\widetilde{R}$, $\widetilde{S}$, and $\widetilde{Q}$ makes a “blurred hyperplane” in the state-control space. Since Z' is not singular (has the same dimension as the state-control space), the gradients of approximations may achieve the required accuracies.

take place for $u \neq \widetilde{A}(x; w^*)$. Consequently, there is no reason to expect that even approximate equations

$$\frac{\mathrm{d}}{\mathrm{d}u}\widetilde{S}(x, u; w_S) \cong \frac{\mathrm{d}}{\mathrm{d}u}\mathcal{E}(x_{t+1}|x_t = x, u_t = u) \tag{A.1}$$

$$\frac{\mathrm{d}}{\mathrm{d}u}\widetilde{R}(x, u; w_R) \cong \frac{\mathrm{d}}{\mathrm{d}u}\mathcal{E}(r_{t+1}|x_t = x, u_t = u) \tag{A.2}$$

hold for $u = \widetilde{A}(x; w^*)$. As a result, the gradient $\frac{\mathrm{d}}{\mathrm{d}u}Q_{HDP}(x, u)$ calculated in Step 2.2. of HDP (Table 2.3) to enable the optimization may not approximate $\frac{\mathrm{d}}{\mathrm{d}u}Q^{\pi(w_A)}(x, u)$ with the required accuracy.

A similar reasoning explains disadvantages of ADHDP.

A simple way to overcome the above difficulties is to apply non-deterministic control. Suppose the controls are drawn from some nonsingular distribution, for example the normal one:

$$u_t \sim N(\widetilde{A}(x_t; w_A), C)$$

This gives the approximators an opportunity to learn the appropriate functions on a nonsingular training set. Consequently, the approximate equations (A.1) and (A.2) may be expected to hold. Figures A.1 and A.2 illustrate both the problem and the remedy.

Our experiments show that the above modification makes HDP and ADHDP converge faster and more stable. Randomization however, once applied, enables much more, namely avoiding a necessity of building the plant's model.

Appendix B

Plants

B.1 Cart-Pole Swing-Up

The Cart-Pole Swing-Up consists of a cart moving along a track and a pole hanging freely from the cart. Control signal defines force applied to the cart. At the beginning of each trial, the pole hangs freely from the cart. The control objective is to avoid hitting the track bounds, swing the pole, turn it up, and stabilize upwards. The problem has been presented in [12]. A similar task involving only keeping the pole up is known as the pole balancing problem [4].

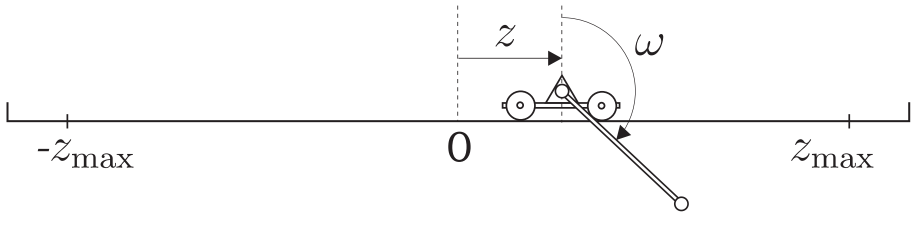

Figure B.1: The Cart-Pole Swing-Up.

There are four state variables: position of the cart z, pole's angle ω, and their derivatives $\dot{z}$, $\dot{\omega}$. A single control variable confined to the interval $[-10, 10]$ becomes a force applied to the cart F is. The state of the plant is *acceptable* if and only if $z \in [-2.4, 2.4]$. Motion of the cart and the pole is driven by the following equations:

$$\ddot{\omega} = \left(\left[\frac{-F - m_p l \dot{\omega}^2 \sin\omega + \mu_c \operatorname{sgn} \dot{z}}{m_c + m_p} \right] \cos\omega + g \sin\omega + \frac{\mu_p \dot{\omega}}{m_p l} \right) l^{-1} \left[\frac{4}{3} - \frac{m_p \cos^2 \omega}{m_c + m_p} \right]^{-1}$$

$$\ddot{z} = \frac{F + m_p l [\dot{\omega}^2 \sin\omega - \ddot{\omega} \cos\omega] - \mu_c \operatorname{sgn} \dot{z}}{m_c + m_p}$$

Parameters of the above equations have the following values and meaning:

z	— position of the cart,
$z_{max} = 2.4$	— max. acceptable position of the cart,
ω	— arc of the pole,
F	— force applied to cart's center of mass,
$g = 9.81$	— acceleration due to gravity,
$m_c = 1$	— mass of the cart,
$m_p = 0.1$	— mass of the pole,
$l = 0.5$	— half-length of the pole,
$\mu_c = 0.0005$	— friction coefficient of the cart on track,
$\mu_p = 0.000002$	— friction coefficient of the pole on cart.

B.2 Robot Weightlifting

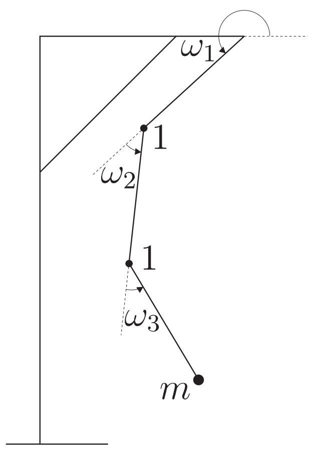

Figure B.2: The lift used in the Robot Weightlifting problem. m denotes the joint mass of the manipulator and the elevated load.

Let us consider a lift like the one above. The lift is a three degrees of freedom robot arm. Each its link is 1 meter long, weighs 1kg, and its mass is concentrated in its ends. At each joint the range of motion is confined to $[-150, 150]$ degrees. A manipulator that is at the end of the last link is attached to a load. We analyze a problem of finding a policy that enables the lift to elevate a load that weighs 1kg or 3kg. Mass of the load is chosen randomly with equal probability at the beginning of each control trial. m is hence drawn from the set $\{1.5, 3.5\}$.

A controller operates by imposing torque limited to $[-50, 50]$ Nm at each joint. The objective of the plant is to move the load from the lowest position to the highest position, i.e. from $\omega = [-\pi/2, 0, 0]^T$ to $\omega = [\pi/2, 0, 0]^T$. The arm is motionless at the beginning of the trial, as it should be at the end.

Motion of the lift can be modeled by the following equations. τ_i denotes torch applied to the i-th joint.

$$B(\omega)\ddot{\omega} + c(\omega, \dot{\omega}) + h(\omega) = \tau$$

$$\begin{aligned}
B_{1,1} &= 3 + 2\cos\omega_2 + m(3 + 2\cos\omega_2 + 2\cos(\omega_2+\omega_3) + 2\cos\omega_3) \\
B_{1,2} = B_{2,1} &= 1 + \cos\omega_2 + m(2 + \cos\omega_2 + \cos(\omega_2+\omega_3) + 2\cos\omega_3) \\
B_{1,3} = B_{3,1} &= m(1 + \cos(\omega_2+\omega_3) + \cos\omega_3) \\
B_{2,2} &= 1 + m2(1 + \cos\omega_3) \\
B_{2,3} = B_{3,2} &= m(1 + \cos\omega_3) \\
B_{3,3} &= m
\end{aligned}$$

$$\begin{aligned}
c_1 =& -\dot{\omega}_1\dot{\omega}_2 2(\sin\omega_2 + m(\sin\omega_2 + \sin(\omega_2+\omega_3))) \\
& -\dot{\omega}_1\dot{\omega}_3 m2(\sin(\omega_2+\omega_3) + \sin\omega_3) \\
& -\dot{\omega}_2^2(\sin\omega_2 + m(\sin\omega_2 + \sin(\omega_2+\omega_3))) \\
& -\dot{\omega}_2\dot{\omega}_3 m2(\sin(\omega_2+\omega_3) + \sin\omega_3) \\
& -\dot{\omega}_3^2 m(\sin(\omega_2+\omega_3) + \sin\omega_3) \\
c_2 =& \dot{\omega}_1^2(\sin\omega_2 + m(\sin\omega_2 + \sin(\omega_2+\omega_3))) \\
& -\dot{\omega}_1\dot{\omega}_3 m2\sin\omega_3 \\
& -\dot{\omega}_2\dot{\omega}_3 m2\sin\omega_3 \\
& -\dot{\omega}_3^2 m\sin\omega_3 \\
c_3 =& \dot{\omega}_1^2 m(\sin(\omega_2+\omega_3) + \sin\omega_3) \\
& +\dot{\omega}_1\dot{\omega}_2 m2\sin\omega_3 \\
& +\dot{\omega}_2^2 m\sin\omega_3
\end{aligned}$$

$$\begin{aligned}
h_1 =& g((2+m)\cos\omega_1 + (1+m)\cos(\omega_1+\omega_2) + m\cos(\omega_1+\omega_2+\omega_3)) \\
h_2 =& g((1+m)\cos(\omega_1+\omega_2) + m\cos(\omega_1+\omega_2+\omega_3)) \\
h_3 =& gm\cos(\omega_1+\omega_2+\omega_3)
\end{aligned}$$

In the reference [33], where the Robot Weightlifting task was introduced, the control issue was defined a little differently. Namely, the problem was to control the lift to elevate as heavy load as possible. It was hence a nonstationary task. We are interested in speed of learning to solve a stationary problem rather than in finding the best possible control policy. We thus somewhat modified the control objective.

B.3 Narendra's MIMO

Narendra's MIMO (multiple-input-multiple-output) problem is an issue of control of a certain artificial discrete time plant. The control objective is for the plant's output to follow a certain path. The problem has been presented in [27] and is widely used in the Adaptive Critic Designs literature (e.g. [32]) as a benchmark.

There are 3 state variables $z_{t,1}, z_{t,2}, z_{t,3}$ and 2 control variables $a_{t,1}, a_{t,2}$. The motion equations of the plant are as follows:

$$\begin{aligned}
z_{t,1} &= \alpha_1 z_{t-1,1}\sin(\alpha_2 z_{t-1,2}) \\
&\quad + \left(\alpha_3 + \frac{\alpha_4 z_{t-1,1} a_{t,1}}{1 + z_{t-1,1}^2 a_{t,1}^2}\right) a_{t,1} \\
&\quad + \left(\alpha_5 z_{t-1,1} + \frac{\alpha_6 z_{t-1,1}}{1 + z_{t-1,1}^2}\right) a_{t,2} \\
z_{t,2} &= z_{t-1,3}(\alpha_7 + \alpha_8 \sin(\alpha_9 z_{t-1,3})) \\
&\quad + \frac{\alpha_{10} z_{t-1,3}}{1 + z_{t-1,3}^2} \\
z_{t,3} &= (\alpha_{11} + \alpha_{12}\sin(\alpha_{13} z_{t-1,1})) a_{t,2}
\end{aligned}$$

Parameters of the above equations have the following values:

Param.	Value	Param.	Value	Param.	Value
α_1	0.9	α_5	1.0	α_9	4.0
α_2	1.0	α_6	2.0	α_{10}	1.0
α_3	2.0	α_7	1.0	α_{11}	3.0
α_4	1.5	α_8	1.0	α_{12}	1.0
				α_{13}	2.0

The control objective is to minimize squared distance between state variables $z_{t,1}$ and $z_{t,2}$ and their desired values $z_{t,1}^d$ and $z_{t,2}^d$. Both $z_{t,1}^d$ and $z_{t,2}^d$ evolve accordingly to a certain stochastic model and only their immediate future values are available, that is $z_{t+i,j}^d$ are unknown at step t for $i > 1$. The model of $z_{t,j}^d$ dynamics is as follows:

$$\begin{aligned}
k_{t,1} &= k_{t-1,1} + \xi_{t,1} \\
z_{t,1}^d &= 0.75\sin(\pi k_{t,1}/25) + 0.75\sin(\pi k_{t,1}/5) \\
k_{t,2} &= k_{t-1,2} + \xi_{t,2} \\
z_{t,2}^d &= 0.75\sin(\pi k_{t,2}/15) + 0.75\sin(\pi k_{t,2}/10)
\end{aligned}$$

where $\xi_{1,t}$ and $\xi_{2,t}$ are random variables drawn independently from the uniform distribution $U(0, 2)$.

In the literature on Adaptive Critic Designs (e.g. [32]), a control policy for the MIMO plant is identified with the use of $z^d_{t,1}$ and $z^d_{t,2}$ that are piecewise constant in time. When the identification is complete, the policy is verified with the use of $z^d_{t,1}$ and $z^d_{t,2}$ generated differently. Namely, their dynamics is similar to the one we describe above, yet $\xi_{1,t}$ and $\xi_{2,t}$ are not random but uniformly equal to 1. In reinforcement learning it is a rare practice to verify the identified control policy with the use of a problem different than the one that one solved during training. Hence our modification.

Bibliography

[1] C. W. Anderson, "Learning to Control an Inverted Pendulum Using Neural Networks," *IEEE Control System Magazine,* vol. 9, pp. 31-37, April 1989.

[2] K. J. Åström and B. Wittenmark, *Adaptive Control,* Addison-Wesley, 1989.

[3] L. Baird and A. Moore, "Gradient Descent for General Reinforcement Learning," *Advances In Neural Information Processing Systems,* vol. 11, pp. 968-974, MIT Press, 1999.

[4] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike Adaptive Elements That Can Learn Difficult Learning Control Problems," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 13(5), pp. 834-846, 1983.

[5] A. G. Barto and T. G. Dietterich, "Reinforcement Learning and Its Relationship to Supervised Learning," in [38], pp. 47-64.

[6] R. E. Bellman, *Dynamic Programming,* Princeton University Press: New Jersey, 1957.

[7] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming,* Athena Scientific, 1997.

[8] X.-R. Cao, "From Perturbation Analysis to Markov Decision Processes and Reinforcement Learning," *Discrete Event Dynamic Systems: Theory and Applications,* 13, no. 1-2, pp. 9-39, 2003.

[9] Y. Chauvin, D. E. Rumelhart, eds. *Backpropagation: Theory, Architectures, and Applications,* Erlbaum, Hillsdale: New Jersey, 1990.

[10] P. Cichosz, *Reinforcement Learning by Truncating Temporal Differences,* Ph.D. dissertation, Department of Electronics and Information Technology, Warsaw University of Technology, 1997.

[11] R. Coulom, *Reinforcement Learning Using Neural Networks, with Applications to Motor Control,* Ph.D. dissertation, Institut National Polytechnique de Grenoble, 2002.

[12] K. Doya, "Efficient Nonlinear Control with Actor-Tutor Architecture," *Advances in Neural Information Processing Systems,* vol. 9, pp. 1012-1018, MIT Press: Cambridge, MA, 1997.

[13] K. Doya, "Reinforcement Learning in Continuous Time and Space," *Neural Computation,* 12:243-269, 2000.

[14] G. J. Gordon, "Stable Function Approximation in Dynamic Programming," *Proceedings of the 12th International Conference on Machine Learning,* pp. 261-268, Morgan Kaufmann, 1995.

[15] V. Gullapalli, "A Stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions," *Neural Networks,* 3, pp. 671-692, 1990.

[16] H. Kimura and S. Kobayashi, "An Analysis of Actor/Critic Algorithm Using Eligibility Traces: Reinforcement Learning with Imperfect Value Functions," *Proceedings of the 15th International Conference on Machine Learning,* pp. 278-286, Morgan Kaufmann, 1998.

[17] V. R. Konda and J. N. Tsitsiklis, "Actor-Critic Algorithms," *SIAM Journal on Control and Optimization,* Vol. 42, No. 4, pp. 1143-1166, 2003.

[18] H. J. Kushner and G. G. Yin, *Stochastic Approximation Algorithms and Applications,* Springer: New York, 1997.

[19] M. G. Lagoudakis and R. Paar, "Model-Free Least-Squares Policy Iteration," *Advances in Neural Information Processing Systems,* vol. 14, pp. 1547-1554, 2002.

[20] D. Liu, X. Xiong, and Y. Zhang, "Action-Dependent Adaptive Critic Designs", *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, Washington, DC, July 2001, pp. 990-995.

[21] J. M. Mendel and R. W. McLaren, "Reinforcement Learning Control and Pattern Recognition Systems," in J. M. Mendel and K. S. Fu (eds.), *Adaptive Learning and Pattern Recognition Systems: Theory and Applications,* pp. 287-318, Academia Press: New York, 1970.

[22] J. R. Millan, D. Posenato, and E. Dedieu, "Continuous-Action Q-Learning," *Machine Learning,* vol. 49, pp. 247-265, Kluwer Academic Publishers, 2002.

[23] W. T. Miller, R. S. Sutton, and P. J. Werbos, (eds.), *Neural Networks for Control,* MIT Press: Cambridge, MA, 1990.

[24] M. L. Minsky, *Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Problems,* Ph.D. dissertation, Princeton University, 1954.

[25] A. W. Moore and C. G. Atkeson, "Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time," *Machine Learning,* vol. 13, pp. 103-130, October, 1993.

[26] D. E. Moriarty and R. Miikkulainen, "Efficient Reinforcement Learning Through Symbiotic Evolution," *Machine Learning,* vol. 22, pp. 11-32, 1997.

[27] K. S. Narendra and K. Mukhopadhyay, "Adaptive Control of Dynamical Systems Containing Neural Networks," *Neural Networks,* vol. 7(5), 737-752, 1994.

[28] D. Precup, R. S. Sutton, S. Singh, "Eligibility Traces for Off-Policy Policy Evaluation," *Proceedings of the 17th International Conference on Machine Learning,* pp. 759-766, Morgan Kaufmann, 2000.

[29] D. Precup, R. S. Sutton, S. Dasgupta, "Off-Policy Temporal-Difference Learning with Function Approximation," *Proceedings of the 18th International Conference on Machine Learning,* pp. 417-424, Morgan Kaufmann, 2001.

[30] D. V. Prokhorov, R. A. Santiago, and D. C. Wunsch, "Adaptive Critic Designs: A Case Study for Neurocontrol," *Neural Networks*, vol. 8, pp. 1367-1372, 1995.

[31] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE Transactions on Neural Networks*, vol. 8, pp. 997-1007, Sept. 1997.

[32] D. V. Prokhorov, G. Puskorius, and L. Feldkamp, Dynamical Neural Networks for Control. In J. Kolen and S. Kremer (Eds.) *A Field Guide to Dynamical Recurrent Networks,* IEEE Press, 2001.

[33] M. T. Rosenstein and A. G. Barto, "Robot Weightlifting By Direct Policy Search," *Proceedings of The 17th International Joint Conference on Artificial Intelligence,* Seattle, Washington, August 2001, pp. 839-846.

[34] R. Rubinstein, *Simulation and The Monte Carlo Method,* Wiley: New York, 1981.

[35] A. L. Samuel, "Some Studies in Machine Learning Using The Game of Checkers," in E. A. Feigenbaum and J. Feldman (eds.) *Computers and Thought,* pp. 71-105, McGraw-Hill: New York, 1963.

[36] C. R. Shelton, "Importance Sampling for Reinforcement Learning with Multiple Objectives," AI Technical Report 2001-003, MIT, August 2001.

[37] J. Si and Y.-T. Wand, "On-Line Learning Control by Association and Reinforcement," *IEEE Transactions on Neural Networks,* vol. 12, pp. 264-276, March 2001.

[38] J. Si, A. G. Barto, W. B. Powell, D. Wunsch II, (eds.) *Handbook of Learning and Approximate Dynamic Programming,* IEEE Press: New York, 2004.

[39] J. Sklansky, “Learning Systems for Automatic Control,” *IEEE Transactions on Automatic Control,* vol. AC-11, no. 1, January 1966.

[40] J. T. Spooner, M. Maggiore, R. Ordonez, and K. M. Passino, *Stable Adaptive Control and Estimation of Nonlinear Systems,* Wiley, 2002.

[41] R. S. Sutton, *Temporal Credit Assignment in Reinforcement Learning*, Ph.D. dissertation, University of Massachusetts, Department of Computer and Information Science, 1984.

[42] R. S. Sutton, “Learning to Predict by the Methods of Temporal Differences,” *Machine Learning,* vol. 3, pp. 9-44, 1988.

[43] R. S. Sutton, “Integrated Architectures For Learning, Planning, and Reacting Based on Approximating Dynamic Programming,” *Proceedings of the 7th International Conference on Machine Learning,* pp. 216-224, Morgan Kaufmann, 1990.

[44] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[45] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy Gradient Methods for Reinforcement Learning with Function Approximation,” *Advances in Information Processing Systems 12,* pp. 1057-1063, MIT Press, 2000.

[46] E. L. Thorndike, *Animal Intelligence,* Hafner, Darien, CT, 1911.

[47] J. N. Tsitsiklis and B. Van Roy, “An Analysis of Temporal-Difference Learning with Function Approximation”, *IEEE Transactions on Automatic Control,* Vol. 42, No. 5, May 1997, pp. 674-690.

[48] A. M. Turing, “Computing Machinery and Intelligence,” in. E. A. Feigenbaum and J. Feldman (eds.), *Computers and Thought,* pp. 11-15, McGraw-Hill, New York, 1963.

[49] C. Watkins, “Learning From Delayed Rewards,” Ph.D. dissertation, Cambridge University Press, Cambridge, England, 1989.

[50] C. Watkins and P. Dayan, “Q-Learning,” *Machine Learning,* vol. 8, pp. 279-292, 1992.

[51] P. Wawrzyński and A.Pacut, “A Simple Actor-Critic Algorithm for Continuous Environments,” *Proceedings of the 10th IEEE International Conference on Methods and Models in Automation and Robotics,* Międzyzdroje, August 2004, pp. 1143-1149.

[52] P. Wawrzyński and A.Pacut, “Intensive Versus Nonintensive Actor-Critic Algorithms of Reinforcement Learning,” *Proceedings of the 7th International Conference on Artificial Intelligence and Soft Computing,* Zakopane, June 2004, pp. 934-941, Springer-Verlag, Vol. 3070/2004.

[53] P. Wawrzyński and A. Pacut, “Model-Free Off-Policy Reinforcement Learning in Continuous Environment,” *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks,* Budapest, July 2004, pp. 1091-1096.

[54] D. A. White and D. A. Sofge, Eds., *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches,* Van Nostrand Reinhold: New York, 1992.

[55] B. Widrow, N. K. Gupta, and S. Maitra, “Punish/Reward: Learning with a Critic in Adaptive Threshold Systems,” *IEEE Transactions on Systems, Man, and Cybernetics,* vol. SMC-3, no. 5, September 1973.

[56] B. Widrow and E. Walach, *Adaptive Inverse Control,* Prentice Hall: Englewood Cliffs, NJ, 1996.

[57] R. Williams, “Simple Statistical Gradient Following Algorithms for Connectionist Reinforcement Learning,” *Machine Learning,* vol. 8, pp. 299-256, 1992.

[58] P. J. Werbos, “Beyond regression: New Tools for Prediction and Analysis in the Behavioral Sciences,” Ph.D. dissertation, Committee on Applied Mathmatics, Harvard University, Cambridge, MA, 1974.

[59] P. J. Werbos, “Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence,” *General Systems Yearbook,* vol. 22, pp. 25-38, 1977.

[60] P. J. Werbos, “A Menu of Designs for Reinforcement Learning Over Time,” (Chapter 3) of W.T. Miller, R.S. Sutton, and P.J. Werbos, (Eds.), *Neural Networks for Control,* MIT Press: Cambrige, MA, 1990.

[61] P. J. Werobs, “Backpropagation Through Time: What It Is and How To Do It,” *Proceedings of the IEEE,* vol. 78, no. 10, pp. 1550-1560, 1990.

[62] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting,* Wiley, 1994.