POLITECNICO DI MILANO

Facoltà di Ingegneria

Scuola di Ingegneria Civile, Ambientale e Territoriale

Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in

Environmental and Land Planning Engineering

A template for master thesis at DEIB

Supervisor:

PROF. VILFREDO PARETO

Assistant Supervisor:

DR. PAUL KRUGMAN

Master Graduation Thesis by:

ANDREA COMINOLA
Student Id n. 111111

EMANUELE MASON
Student Id n. 222222

Academic Year 2011-2012

POLITECNICO DI MILANO

Facoltà di Ingegneria

Scuola di Ingegneria Civile, Ambientale e Territoriale

Dipartimento di Elettronica, Informazione e Bioingegneria

Corso di Laurea Magistrale in

Ingegneria per l'Ambiente e il Territorio

Un modello per tesi di laurea magistrale al DEIB

Relatore:

PROF. VILFREDO PARETO

Correlatore:

DR. PAUL KRUGMAN

Tesi di Laurea Magistrale di:

ANDREA COMINOLA
Matricola n. 111111

EMANUELE MASON
Matricola n. 222222

Anno Accademico 2011-2012

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

The template has been adapted by Emanuele Mason, Andrea Cominola and Daniela Anghileri: *A template for master thesis at DEIB*, June 2015

Here you can put your dedication, like:

To time, that do not go backwards

— A & D & E

ACKNOWLEDGMENTS

Here you can put acknowledgements to people that helped you during the thesis. Remember that helping students to write thesis is part of the job of some of them, and they're also paid for that. Please make sure to thank them for what they weren't supposed to do.

Remember also that this page is part of your thesis. I know that your boyfriend/girlfriend is very important to you and you cannot live without her/him, as it is for me. But there's no need to put her/his name here unless she/he gave a proper contribution to this work. Same goes for friends, parents, drinking buddies and so on.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

MDP	Markov Decision Process
MDPs	Markov Decision Processes
RL	Reinforcement Learning
DP	Dynamic Programming
BMDP	Bounded-parameter MDP
PGPE	Policy Gradients with Parameter-Based Exploration
DPG	Deterministic Policy Gradient
MC	Monte Carlo
TD	Temporal Difference
IVI	Interval Value Iteration
LC	Lipschitz Continuous
PS	Policy Search

RBF Radial Basis Functions

ABSTRACT

An abstract is a brief of a research article, thesis, review, conference proceeding or any in-depth analysis of a particular subject or discipline, and is often used to help the reader quickly ascertain the paper's purpose. When used, an abstract always appears at the beginning of a manuscript or typescript, acting as the point-of-entry for any given academic paper or patent application. Abstracting and indexing services for various academic disciplines are aimed at compiling a body of literature for that particular subject.

Max 2200 characters, spaces included.

SOMMARIO

Per abstract si intende il sommario di un documento, senza l'aggiunta di interpretazioni e valutazioni. L'abstract si limita a riassumere, in un determinato numero di parole, gli aspetti fondamentali del documento esaminato. Solitamente ha forma "indicativo-schematica"; presenta cioé notizie sulla struttura del testo e sul percorso elaborativo dell'autore.

Max 2200 caratteri compresi gli spazi.

ESTRATTO

"... il testo delle tesi redatte in lingua straniera dovrà essere introdotto da un ampio estratto in lingua italiana, che andrà collocato dopo l'abstract."

PREFACE

One of the challenges that arise in Reinforcement Learning (RL) is the trade-off between exploration and exploitation. To obtain an high reward, an agent must perform actions that it has found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before [Sutton and Barto, 2018]. An intuitive strategy to address the issue is to perform random actions in the beginning of the task, so as to test as many actions as possible in a first phase and then exploit the most rewarding actions in the long term. By executing random actions in any state, the agent potentially executes and evaluates the entire range of feasible behaviours. An approach of this kind strongly supports exploration and provides future advantages to the agent.

If the agent evolves into a simulator, no concerns are related to the strategy, however when the agent interacts with a real-life environment serious drawbacks come up. The lack of command on the agent's behaviour can lead to dangerous actions that damage the agent's hardware or, even worse, harm the humans that operate in the environment. Because of this safety issue, RL techniques are scarcely used in fields where they could provide outstanding benefits as industrial robotics, surgery [Baek et al., 2018] or autonomous driving. Again, another important scenario in which the randomness of actions is unwanted is finance: certainly nobody wants to perform stochastic operations involving their money. In RL several definitions of safety have been proposed (we discuss them in section 1.5), however the safety problem due to randomness of actions is poorly addressed. The reason is that in RL the stochasticity is essential for exploration. Indeed, if the agent always does the same action in every state, it cannot find any better action that improves its behaviour.

Driven by these motivations, we explore an alternative strategy, that ensures safeness in every instant of the agent-environment interaction by monitoring the explorative behaviour of the agent. The solution we propose is suitable for regular environments, i. e. environments in which performing the same action in similar states produces similar effects. In these environments, once an action is executed in a certain state, we can evaluate its effect in other (similar) states, without necessarily redoing the action. This assumption provides a sort of exploration that the agent can exploit instead of relying on the stochasticity of its actions. As a result, in any state the agent performs only a deterministic action but evaluates all the actions that have been executed in

similar states. Only a subset of the possible action is evaluated, then the learning ability of the agent is significantly reduced in this approach: learning an optimal behaviour becomes infeasible or requires an excessive amount of time. On the other hand, the agent is able to learn and improve his behaviour without the need to perform random actions.

descrizione delle sezioni

We present our work...

1 STATE OF THE ART

According to [Sutton and Barto, 2018], Reinforcement Learning (RL) consists in learning what to do so as to maximize a numerical reward signal. The learner must discover which actions yield the most reward by trying them. Actions may affect not only the immediate reward but also the next situation and all subsequent rewards. The problem of RL is formalized with an idea coming from dynamical systems theory, specifically with the Markov Decision Processes (MDPs) that we detail in section 1.1.

Chapter 1 provides a strong theoretical contribution to our work. After the introduction on MDPs in section 1.1, section 1.2 refers to the *Tabular Methods* used to solve MDPs. Even if the application of these methods is limited by computational costs, they provides basic concepts in RL. Section 1.3 explores *Policy Search (PS)*, a class of methods that solve RL problems with a different approach. PS offers several advantages in the robotic field. Then, we present two special types of MDPs in section 1.4: *Bounded MDPs*, suitable to model uncertainty in the environment, and *Lipschitz MDPs*, suitable to model regularity in the environment. Finally, section 1.5 presents some existing approaches to the safety issue in RL and section 1.6 presents some topics related to state discretization.

1.1 Markov Decision Processes (MDPs)

MDPs are a mathematical framework used for modeling RL problems in which an *agent*, by interacting with an *environment*, learns how to achieve a goal. MDPs are a formalization of sequential decision making, where the decision taken by the agent influences immediate and future rewards. The agent interacts with the environment by selecting the actions to perform and receiving rewards and information on the new situation. Rewards are provided by the environment in the form of scalar values, the agent aims to maximize the sum of them over time.

Definition 1.1 (MDP)**.** The Markov Decision Process (MDP) is described by a five-tuple $M = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where:

- $\mathcal{S}$ is the state space, with $\mathcal{S} \subseteq \mathbb{R}^N$.
- $\mathcal{A}$ is the action space, with $\mathcal{A} \subseteq \mathbb{R}^D$.
- $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, with $P(s'|s, a)$ denoting the probability of reaching state s' from state s by taking

action a. $P(.|s,a)$ is a distribution of probability on the arriving state, then for any state-action pair (s,a), the following equality holds:

$$\sum_{s' \in \mathcal{S}} P(s'|s,a) = 1. \tag{1.1}$$

- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, with $R(s,a)$ denoting the expected reward from taking action a in state s.
- $\gamma \in [0,1)$ is the discount factor, eventually used to discount the present effect of future rewards.

The transition function P defines the dynamics of the MDP and ensures the *Markov property*: the probability of reaching s_t depends only on the immediately previous state and action, s_{t-1} and a_{t-1}, and not on earlier states and actions.

The interaction between agent and environment can be better explained with symbols: at each time step t the agent receives a representation of the state $s_t \in \mathcal{S}$, on that basis it selects an action $a_t \in \mathcal{A}$. One time step later, the agent receives a reward r_{t+1} and finds itself in a new state $s_{t+1} \in \mathcal{S}$. We define the MDP according to [Puterman, 2014]. In general, the state space $\mathcal{S}$ and the action space $\mathcal{A}$ can be finite or continuous sets. We focus on continuous set MDPs as these are the most suitable for modeling continuous control problems.

1.1.1 *Policy and Value Functions*

The behaviour of the agent is modeled with a *policy* $\pi : \mathcal{S} \mapsto \Delta(\mathcal{A})$, i.e. a mapping from states to probabilities of selecting each possible action. The agent learns a policy according to its goal of maximizing the sum of rewards collected during the task. Specific details on how to learn the optimal policies are given later, in section 1.2 and section 1.3. The sum of rewards obtained by the agent, if we consider the time steps $k \in (t,T]$, where the time step T represents the horizon of the task, is called *return* G_t. In general, the return is defined as a sum of discounted rewards:

Uso G come nome?

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k, \tag{1.2}$$

where r_k is the reward obtained at step k. The horizon T can be finite or infinite. In the first case, the task is said to be *episodic* and the agent-environment interaction breaks naturally into episodes. In the second case (i.e. $T = \infty$), the task is said to be *continuing* and a discount factor $\gamma < 1$ is required in order to obtain a return $G_t < \infty$.

In this work, we denote with $\pi(a|s)$ the probability of performing the action a in the state s, according to the policy π. Since $\pi(s)$ is a distribution of probability, the following equality holds:

$$\sum_{a \in \mathcal{A}} \pi(a|s) = 1 \quad \forall s \in \mathcal{S}. \tag{1.3}$$

If, for each state $s \in \mathcal{S}$, there exists an action a such that $\pi(a|s) = 1$, the policy is deterministic.

Given a policy π, it is possible to compute the *value function* $V^\pi : \mathcal{S} \to \mathbb{R}$. This is a widely used function in RL that measures how good it is for the agent to be in a given state $s \in \mathcal{S}$, according to the expected return obtainable from that state. Since the rewards that the agent can expect to receive in the future depend on the actions taken in any state, the value function is defined with respect to policies: $V^\pi(s)$ is the expected sum of discounted rewards that the agent collects by starting at state s and following policy π. The value function can be defined recursively via the Bellman equation:

$$V^\pi(s) = \int_{\mathcal{A}} \pi(a|s) \Big(R(s,a) + \gamma \int_{\mathcal{S}} P(s'|s,a) V^\pi(s') \, ds' \Big) \, da. \tag{1.4}$$

For control purposes, we can also define an action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$Q^\pi(s,a) = R(s,a) + \gamma \int_{\mathcal{S}} P(s'|s,a) \int_{\mathcal{A}} \pi(a'|s') Q^\pi(s',a') \, da' \, ds'. \tag{1.5}$$

$Q^\pi(s,a)$ represents the expected return obtained from taking the action a in state s and then following the policy π.

Finally, we denote with

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{1.6}$$

the advantage function of policy π, that represents the advantage in terms of value functions given by performing action a in state s, instead of the action prescribed by $\pi(s)$.

1.1.2 *Performance Measure*

In order to evaluate how good a policy π is, we consider the expected return obtained starting from a state $s \in \mathcal{S}$ drawn from the initial-state distribution μ and following the policy π. The *performance measure* $J(\pi)$ expressed in the form of an expected value is:

$$J(\pi) = \mathop{\mathbb{E}}_{s_0 \sim \mu} G_0 \quad = (1-\gamma)^{-1} \mathop{\mathbb{E}}_{s \sim \delta^\pi, a \sim \pi} R(s,a) \quad = \mathop{\mathbb{E}}_{s \sim \mu} V^\pi(s), \tag{1.7}$$

where δ^π is the γ-discounted future-state distribution. This function is defined as:

$$\delta^\pi(s) = (1-\gamma) \mathop{\mathbb{E}}_{s_0 \sim \mu} \sum_{t=0}^{\infty} \gamma^t P^\pi(S_t = s | S_0 = s_0) \tag{1.8}$$

and represents the probability of being in a certain state s during the execution of the task, provided that the policy is π and the initial state distribution is μ.

The performance measure is used to identify the optimal policy π^* that we want to learn in the RL problem as:

$$\pi^* \in \arg\max_\pi J(\pi). \tag{1.9}$$

1.2 TABULAR SOLUTION METHODS

Solving a RL task means finding a policy that maximizes the return obtained by the agent in the task. First, we consider finite MDPs, a subset of the MDPs defined in 1.1. In order to reason with finite MDPs, we consider the state space $\mathcal{S}$ and the action space $\mathcal{A}$ as finite sets of discrete values and we replace all the integrals appearing in the expressions reported so far with summations. The methods used to solve problems involving finite MDPs are called *tabular* because the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are small enough for the value functions to be represented in a tabular format.

Policies can be partially ordered according to their value function:

$$\pi' \geqslant \pi \iff V^{\pi'}(s) \geqslant V^\pi(s) \quad \forall s \in \mathcal{S}. \tag{1.10}$$

In finite MDPs, there always exists a deterministic optimal policy π^* such that $\pi^* \geqslant \pi \quad \forall \pi \in \Pi$, said Π the set of policies. The value function V^* computed according to π^* has the following property:

$$V^*(s) = \max_\pi V^\pi(s) \quad \forall s \in \mathcal{S}. \tag{1.11}$$

Optimal value function V^* and optimal action-value function Q^* can be written with the Bellman optimality equations:

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right) \tag{1.12}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a'). \tag{1.13}$$

If the agent has a *complete knowledge* of the environment, i. e. the agent knows the transition function P and the reward function R of the MDP M representing the environment, Dynamic Programming (DP) can be

used to solve the RL problem. Details are provided in subsection 1.2.1. Instead, if the agent has an *incomplete knowledge* of the environment, the unknown functions P and R of MDP M can be estimated from experience. Details are provided in subsection 1.2.2.1.

When the problems involve finite MDPs with a larger state space or continuous MDPs, tabular methods are no more suitable because value functions cannot be represented in tables. Two main approaches are possible:

- the value functions can be estimated with *function approximators* and used to solve the task;
- the optimal policies can be directly searched in the space of policies, without the necessity of computing any value function. Details are provided in (section 1.3).

1.2.1 *Dynamic Programming*

Dynamic Programming (DP) is a collection of algorithms that can be used to compute optimal policies, given a model of the environment in the form of an MDP. Since DP is computationaly expensive and requires finite MDPs, its utility in solving RL problems is limited. However DP provides strong foundation for understanding the following methods. DP offers two algorithms that compute the optimal value functions: *policy iteration* and *value iteration*.

1.2.1.1 *Policy Iteration*

In policy iteration, two consecutive operations are performed on a policy π in order to obtain a policy π' that is better according to the ordering rule in (1.10). These operations are called *policy evaluation* and *policy improvement* and the goal of the algorithm is to obtain the optimal value function V^* through the sequence:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V^*. \tag{1.14}$$

Policy evaluation consists in computing the value function V^π for every state $s \in \mathcal{S}$, according to the current policy π, by iteratively applying the following updating rule:

$$V_{k+1}(s) = \sum_a \pi(a|s)\Big(R(s,a) + \gamma \sum_{s'} P(s'|s,a)V_k(s')\Big), \tag{1.15}$$

until $V_{k+1}(s) = V_k(s) \quad \forall s \in \mathcal{S}$. When it happens, the convergence to V^π is obtained. The existence of V^π is guaranteed as long as $\gamma < 1$.

enough for convergence? cit some paper?

Policy improvement consists in an update of policy π in a new deterministic policy π', according to the value function V^π computed

in the previous policy evaluation. For each state $s \in \mathcal{S}$, indeed, the policy π' is computed according to the following rule:

$$\pi'(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi}(s') \right). \tag{1.16}$$

When $\pi'(s) = \pi(s) \quad \forall s \in \mathcal{S}$ or π' is as good as π (i. e. $V^{\pi} = V^{\pi'}$), the algorithm terminates. The policy π' is considered to be optimal, then $\pi^* = \pi'$.

1.2.1.2 *Value Iteration*

The main drawback in policy iteration is that, for each policy evaluation, multiple iterations of the updating rule (1.15) are required. If we truncate the policy evaluation after just one application of the updating rule, we obtain the value iteration. In value iteration, the value function at each step is updated for all $s \in \mathcal{S}$ as follows:

$$V_{k+1}(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s') \right). \tag{1.17}$$

cit smth related to the convergence?

Differently from policy iteration, we don't compute any intermediate policy π_i. However, the sequence $\{V_k\}$ of the value functions converges to V^* and the optimal policy π^* is obtained applying (1.16), with V^* instead of V^{π}.

1.2.2 *Uncomplete Knowledge of MDPs*

In many cases, however, it is not possible to apply DP. Some alternatives are introduced below for sake of completeness. We don't detail these methods because they are not related to our work.

1.2.2.1 *Tabular Alternatives to DP*

If we don't have a complete knowledge of the environment, we can learn the unknown dynamics from experience, i. e. from the samples collected by the agent while interacting with the environment. The samples are tuples $\langle s, a, r, s' \rangle_t$, with $s, s' \in \mathcal{S}, a \in \mathcal{A}$ and $r = R(s,a)$. They contain information related to the interaction agent-environment at time step t. The two main classes of algorithms are Monte Carlo (MC) and Temporal Difference (TD). MC methods involve episodic tasks, TD learning involves continuing tasks.

1.2.2.2 *Approximate Solution Methods*

The methods present so far are not suitable to solve RL problems that involve a larger state space. In these problems we cannot expect to obtain the optimal policy, our goal is to find a good approximate solution using limited computational resources. In an environment

with a continuous state space $\mathcal{S}$, the agent will always visit states s never seen before. Then, the experience we gather gives information on a subset of states but we need to generalize it to all the state space $\mathcal{S}$. The generalization comes up in the form of *function approximations*. These functions are estimated from samples and approximate value functions of continuous state space $\mathcal{S}$ in place of tables.

Several methods rely on value function approximation in order to provide the solution of RL problem. However, function approximation arises new issues. First of all, convergence assurances are difficult to be given: if the policy is greedy, an arbitrary small change in the estimated value of an action can cause it to be, or not be, selected [Sutton et al. (1999)]. Approximated value functions U_t can introduce a bias that prevents the method to converge to a local optimum.

too complex functions as neural networks with a lot of parameters as approximate value fun?

1.3 POLICY SEARCH

In contrast with value-based methods, Policy Search (PS) methods use parametrized policies π_θ, where $\theta \in \Theta$. They directly operate in the parameter space Θ to find the optimal parametrized policy and typically avoid to learn a value function. PS copes with high dimensional state space $\mathcal{S}$ and action space $\mathcal{A}$ and offers better convergence assurances compared to the methods that involve function approximation. Because of this, PS methods have been found to be suitable in robotic applications.

Most of the algorithms in PS are *model-free* (i. e. they don't require a model of the environment) because directly learning a policy is often easier than learning an accurate model. These methods update the policy directly exploiting the sampled trajectories, hence it is important to define an exploration strategy that provides variety in trajectories. Exploration can be performed both in the action space and in the parameter space. The former one is implemented by adding a noise ϵ directly to the executed actions, the noise is generally sampled from a zero-mean Gaussian distribution. The latter consists in a perturbation of the parameter vector θ of π_θ. The magnitude of noise present in any kind of exploration depends on some parameters. These parameters can also be updated by the algorithms: usually the size of exploration is gradually decreased to fine tune the policy parameters. [Deisenroth, Neumann, and Peters (2013)].

1.3.1 *Policy Representations*

In our work, we focus on the optimization of deterministic parametric policies of the form $\pi_\theta : \mathcal{S} \to \mathcal{A}$, with $\theta \in \Theta \subseteq \mathbb{R}^{m \times D}$. We will often abbreviate π_θ as θ in subscripts and function arguments, e. g.

$V^\theta \equiv V^{\pi_\theta}$, $J(\theta) \equiv J(\pi_\theta)$. The simplest way of parametrizing π_θ is by means of a linear mapping. The linear policy is defined as

$$\pi_\theta(s) = \theta^T \boldsymbol{\phi}(s), \tag{1.18}$$

where $\theta \in \mathbb{R}^{m \times D}$ and $\boldsymbol{\phi} : \mathcal{S} \to \mathbb{R}^m$ is a feature function. This can be the state itself or, for instance, a set of Radial Basis Functions (RBF). An example of RBF is the Gaussian

$$\phi_i(s; \mu_i, \sigma_i) = \exp\left\{-(s - \mu_i)^2/(2\sigma_i^2)\right\}, \tag{1.19}$$

where μ_i and σ_i are hyperparameters, $i = 1, \ldots, m$. More complex policy parametrizations include deep neural networks (Duan et al., 2016). Stochastic policies randomize over actions. In continuous settings, this is typically done by adding a Gaussian noise, e. g. for a linear policy $a \sim \mathcal{N}(\theta^T \boldsymbol{\phi}(s), \Sigma)$, where $\Sigma \in \mathbb{R}^{D \times D}$ is a covariance matrix.

A common parametrization for countinuous actions represented by real numbers, is the normal distribuion. The policy can be defined as the normal probability density over a scalar action, with mean μ and standard deviation σ given by parametric function approximators that depend on the state:

$$\pi_\theta(a|s) = \frac{1}{\sigma(s,\theta)\sqrt{2\pi}} \exp\Big(-\frac{(a - \mu(s,\theta))^2}{2\sigma(s,\theta)^2}\Big), \tag{1.20}$$

where $\mu(s,\theta) = \theta_\mu^T \boldsymbol{\phi}(s)$ and $\sigma(s,\theta) = \exp\Big(\theta_\sigma^T \boldsymbol{\phi}(s)\Big)$. The policy's parameter vector is $\theta = [\theta_\mu, \theta_\sigma]^T$.

1.3.2 *Policy Gradient*

The most important class of algorithms in PS is the one in which the algorithms learn the parameterized policy $\pi_\theta, \theta \in \mathbb{R}^d$ based on the gradient of some scalar performance measure $J(\theta)$ with respect to the policy parameter θ. These methods seek to maximize performance, so their updates approximate gradient ascent in J:

$$\theta \leftarrow \theta + \alpha \widehat{\nabla J}(\theta), \tag{1.21}$$

where $\widehat{\nabla J}(\theta) \in \mathbb{R}^d$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument θ and α is the step size, a scalar that controls the size of each update and can change through time. In policy gradient methods, the policy $\pi(a|s,\theta)$ has to be differentiable with respect to its parameters $\theta \in \mathbb{R}^d \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$.

With policy parametrization the action probabilities change smoothly as a function of the learned parameter, whereas for *greedy policies* (i. e.

policies that select the action to be performed in any state as the one having the highest value according to some value functions) the action probabilities may change enormously for a small change in the estimated action values. Stronger convergence guarantees are available for policy-gradient methods and approximate gradient ascent can be performed [1.21]. According to [Peters and Schaal, 2008], if the gradient estimate is unbiased and learning rates fulfill

$$\sum_{t=1}^{\infty} \alpha_t = \infty \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty \tag{1.22}$$

the learning process is guaranteed to converge to at least a local optimum. Policy-gradient methods can also incorporate domain knowledge in the policy definition and can be made safe by design.

sightly different from the citation

Some methods also learn approximation to value-functions and are called *actor-critic methods*, where "actor" is a reference to the learned policy, and "critic" refers to the learned value function. In these methods, the function approximation for value function introduces bias but reduces variance and accelerates learning.

1.3.3 *Policy Gradient Theorem*

It may seem challenging to change the policy parameter in a way that ensures improvement. The problem is that the performance $J(\theta)$, i. e. the measure to maximize, depends on both the action selections and the distribution of states in which those selctions are made, and both of these are affected by the policy parameter θ. We have to estimate the performance gradient with respect to θ but the gradient is affected by the unknown effect of policy changes. Fortunately, there is a theoretical answer to this challenge in the form of the *policy gradient theorem*, which provides an analytic expression for $\nabla_\theta J(\theta)$ that does not involve the derivative of the state distribution $\delta^\theta(s)$.

The theorem is from [Sutton et al. (1999)] and we enunciate it in the case of continuous state space $\mathcal{S}$ and action space $\mathcal{A}$:

$$\nabla_\theta J(\theta) = \int_s \delta^\theta(s) \int_a \nabla_\theta \pi(a|s) Q^\theta(s,a)\, da\, ds. \tag{1.23}$$

Starting from (1.23), the performance gradient $\nabla_\theta J(\theta)$ can be estimated from samples in different ways. In subsection 1.3.4 we present some policy gradient algorithms. Generally, a batch of N trajectories is sampled and N single estimates of gradient are averaged to obtain the final estimate. This estimate is the $\widehat{\nabla J}(\theta)$ used in (1.21) to perform an iteration that updates the policy parameter θ.

1.3.4 *Policy Gradient Algorithms*

In this section we present some policy gradient algorithms that have some interesting properties for our work. We compare these existing algorithms and their properties with the algorithm proposed in our work. The first algorithm presented is REINFORCE [1.3.4.1], a method directly derived from the Policy Gradient Theorem [subsection 1.3.3] that shows how the gradient of performance can be easily estimated from samples. Then, we present two method that learn deterministic policies. The difference between these two methods is the way in which exploration is performed: in Policy Gradients with Parameter-Based Exploration (PGPE) [1.3.4.2] exploration is performed in the policy space while in Deterministic Policy Gradient (DPG) [1.3.4.3] exploration is performed by a behavioural policy (i.e. a policy that collects samples, different from the policy that will be learnt).

1.3.4.1 *REINFORCE*

In REINFORCE the gradient of performance $\nabla_\theta J(\theta)$ can be estimated from a single trajectory $\langle s_0, a_0, r_1, s_1, a_1, ..., s_{T-1}, a_{T-1}, r_T \rangle$ without that any sort of perturbation is performed on parameters θ, according to [Williams (1992)]. Starting from (1.23), we can write the gradient as an expected value depending on the state distribution δ^θ and the parameterized policy π_θ. Then, the samples collected following π_θ can be used for the estimate of $\nabla_\theta J(\theta)$:

$$\begin{aligned}\nabla_\theta J(\theta) &= \int_s \delta^\theta(s) \int_a \pi(a|s) \frac{\nabla_\theta \pi(a|s)}{\pi(a|s)} Q^\theta(s,a)\, da\, ds \\ &= \mathop{\mathbb{E}}_{s_t \sim \delta^\pi, a_t \sim \pi} \left[\frac{\nabla_\theta \pi(a_t|s_t)}{\pi(a_t|s_t)} Q^\theta(s_t, a_t) \right] \\ &= \mathop{\mathbb{E}}_{s_t \sim \delta^\pi, a_t \sim \pi} \left[\nabla_\theta \log \pi(a_t|s_t) Q^\theta(s_t, a_t) \right] \qquad (1.24)\end{aligned}$$

where (1.24) is obtained by considering $\nabla_\theta \log \pi(a_t|s_t) = \frac{\nabla_\theta \pi(a_t|s_t)}{\pi(a_t|s_t)}$ and $Q^\theta(s_t, a_t)$ can be approximated by a function.

REINFORCE has good theoretical convergence properties: by construction, the expected update computed at each iteration increases the performance. However, it presents ah high variance that slows the convergence process. In order to mitigate this problem, we consider *baseline functions* $b : \mathcal{S} \to \mathbb{R}$, i.e. functions that don't depend on the action and can reduce variance in the estimate without introducing bias. Indeed, we observe that if $b(s)$ does not depend on a we can modify (1.23) in the following way:

$$\nabla_\theta J(\theta) = \int_s \delta^\theta(s) \int_a \nabla_\theta \pi(a|s) \Big(Q^\theta(s,a) - b(s) \Big)\, da\, ds. \qquad (1.25)$$

The new term appearing in (1.25) doesn't affect the value of gradient:

$$\int_a b(s)\nabla_\theta \pi(a|s)\,da = b(s)\nabla_\theta \int_a \pi(a|s)\,da = b(s)\nabla_\theta 1 = 0. \quad (1.26)$$

An intuitive baseline $b(s)$ that can be used is the value function $V^\theta(s)$. In some tasks, $Q^\theta(s,a)$ can assume high values and the use of $V^\theta(s)$ as baseline provides an effect of normalization. The difference in expression (1.25) has the meaning of advantage function $A^\theta(s,a)$:

$$\nabla_\theta J(\theta) = \int_s \delta^\theta(s) \int_a \nabla_\theta \pi(a|s)\Big(Q^\theta(s,a) - V^\theta(s)\Big)\,da\,ds. \quad (1.27)$$

In practice, the value functions Q^θ and V^θ are not available and have to be estimated. In episodic tasks, the return G_t can be computed from every time step t, then a value $Q(s_t,a_t)$ can be assigned to every state-action pair (s_t,a_t) sampled in the trajectory. As a result, starting from a trajectory $\langle s_0,a_0,r_1,s_1,a_1,...,s_{T-1},a_{T-1},r_T\rangle$ terminating in time step T, the gradient of performance is estimated as:

$$\widehat{\nabla_\theta J}(\theta) = \sum_{k=0}^{T-1} \nabla_\theta \log \pi_\theta(a_k|s_k)\Big(\sum_{l=k+1}^{T} \gamma^{l-1} r_l - b_l\Big), \quad (1.28)$$

where the variance-minimizing baseline [Peters and Schaal, 2008] is:

controllare indici delle formule

$$b_l = \frac{\Big(\sum_{k=0}^{l} \nabla_\theta \log \pi^\theta(a_k|s_k)\Big)^2 \gamma^{l-1} r_l}{\Big(\sum_{k=0}^{l} \nabla_\theta \log \pi^\theta(a_k|s_k)\Big)^2} \quad (1.29)$$

This is one of the most common policy gradient algorithms and we use it in our work to compare our algorithm with a standard algorithm in policy search.

1.3.4.2 *PGPE*

Policy Gradients with Parameter-Based Exploration (PGPE)[Sehnke et al. (2008)] is a method that estimates a gradient by directly sampling in parameter space, this feature reduces the variance in the estimates in comparison to REINFORCE. In PGPE the policy is defined by a distribution over the parameters of deterministic controllers that we indicate with the function $\mu_\theta : \mathcal{S} \to \mathcal{A}$. At the beginning of each step, the parameter θ of the controller are sampled and then, the deterministic policy μ_θ generated from θ is followed for all the episode length. This is the reason why the variance is lower in PGPE than in REINFORCE. Indeed, in REINFORCE a repetitive sampling from a stochastic policy is performed and the variance increases even in the same episode, since every state depends on the previous history of the episode that grows in time.

As we said, in PGPE the stochasticity that in REINFORCE was given by a stochastic policy π_θ is replaced by a probability distribution over the parameter θ themselves. In turn, this probability distribution is parameterized with parameter ρ, independent from θ. Given a parameter space Θ for (deterministic) controllers $\mu_\theta, \theta \in \Theta$, the policy considered to perform exploration in the task is:

$$\pi_\rho(a|s) = \int_\Theta p_\rho(\theta)\delta_{\mu_\theta(s)}\, d\theta, \tag{1.30}$$

where $\delta_{\mu_\theta(s)}$ is the Dirac delta function corresponding to the deterministic controller μ_θ depending on parameter θ sampled from a distribution with probability $p_\rho(\theta)$.

Given a trajectory $h \in \mathbb{H}$, where $\mathbb{H}$ is the set of possible trajectories, a probability $p_\rho(h, \theta)$ of sampling parameter θ and trajectory h and defined $G(h)$ the return of trajectory h, we can define a suitable performance measure $J(\rho)$ as:

$$J(\rho) = \int_\Theta \int_{\mathbb{H}} p_\rho(h, \theta) G(h)\, dh\, d\theta. \tag{1.31}$$

From 1.31, we can write the expected value of the gradient of J w.r.t. the distribution parameter ρ as:

$$\widehat{\nabla_\rho J}(\rho) \approx \nabla_\rho \log p_\rho(\theta) G(h), \tag{1.32}$$

where θ is sampled at the beginning of the episode and h is resulting from the deterministic controller μ_θ. Usually, we can consider the parameter ρ learnt by the algorithm as $\rho = \Big(\{\mu_i\}, \{\sigma_i\}\Big)$, where μ_i and σ_i are the parameters that determine an indipendent normal distribution $p_{\rho_i}(\theta)$ for each parameter $\theta_i \in \Theta$.

1.3.4.3 *DPG*

Deterministic Policy Gradient (DPG) [Silver et al. (2014)] is a class of algorithms that learn a parametric deterministic policy. This is similar to what we want to achieve but the problem is that during the learning phase DPG collect samples from a stochastic behavioural policy and then unexpected actions can be performed.

Policy gradient algorithms are perhaps the most popular class of continuous action reinforcement learning algorithms, starting from the results obtained by the *policy gradient theorem*, we consider a deterministic policy $\mu_\theta : S \mapsto A$ with parameter vector $\theta \in \mathbb{R}^n$ and performance objective $J(\mu_\theta) = \mathbb{E}_{s\sim\delta^\mu}[r(s, \mu_\theta(s))]$. By applying the chain rule we see that the policy improvement may be decomposed:

$$\nabla_\theta Q^\mu(s, \mu_\theta(s)) = \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}$$

Suppose that $\nabla_\theta \mu_\theta(s)$ and $\nabla_a Q^\mu(s,a)$ exist, then:

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\delta^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}]$$

We use the deterministic policy gradient theorem to derive an off-policy actor-critic algorithm. It learns a deterministic target policy $\mu_\theta(s)$ from trajectories generated by an arbitrary stochastic behaviour policy $\pi(a|s)$. A critic estimates the action-value function $Q^w(s,a) \approx Q^\mu(s,a)$.

1.4 SPECIAL MDPs

1.4.1 *Bounded MDPs*

A Bounded-parameter MDP (BMDP) [Givan, Leach, and Dean, 2000] is a five-tuple $\langle \mathcal{S}, \mathcal{A}, P_\updownarrow, R_\updownarrow, \gamma \rangle$, where $\mathcal{S}$, $\mathcal{A}$ and γ are defined as for (finite) MDPs, and $P_\updownarrow, R_\updownarrow$ are analogous to the MDP transition and reward functions, but yield closed real intervals instead of real values: given a lower bound $\underline{P}$ and an upper bound $\overline{P}$, $P_\updownarrow = [\underline{P}; \overline{P}]$. Similarly we can specify the interval $R_\updownarrow$. This can be used to model uncertainty on the true nature of a decision process. To ensure that $P_\updownarrow$ admits only well-formed transition functions, we require that for any action a and state s, the sum of the lower bounds of $P_\updownarrow(s'|s,a)$ over all states s' must be less than or equal to one, while the upper bounds must sum to a value greater than or equal to one.

A BMDP $M_\updownarrow = \langle \mathcal{S}, \mathcal{A}, P_\updownarrow, R_\updownarrow, \gamma \rangle$ defines a set of exact MDPs. For any exact MDP $M = \langle \mathcal{S}', \mathcal{A}', P', R', \gamma' \rangle$, we have $M \in M_\updownarrow$ if $\mathcal{S} = \mathcal{S}', \mathcal{A} = \mathcal{A}', \gamma = \gamma'$, and for any action a and states s, s', $R'(s,a)$ belongs to the interval $R_\updownarrow(s,a)$ and $P'(s'|s,a)$ belongs to the interval $P_\updownarrow(s'|s,a)$. An interval value function $V_\updownarrow$ is a mapping from states to closed real intervals. We use such functions to indicate that the value of a given state for any exact MDP falls within the selected interval. As in the case of (exact) value functions, interval value functions are specified w.r.t. a fixed policy π, i. e. :

$$V_{\updownarrow,\pi}(s) = \left[\underline{V}_\pi(s), \overline{V}_\pi(s)\right] = \left[\min_{M\in M_\updownarrow} V_{M,\pi}(s), \max_{M\in M_\updownarrow} V_{M,\pi}(s)\right]. \quad (1.33)$$

According to[Givan, Leach, and Dean, 2000], there exists in $M_\updownarrow$ an MDP that simultaneously achieves $\underline{V}_\pi(s)$ for all $s \in \mathcal{S}$ and another one that achieves $\overline{V}_\pi(s)$ for all $s \in \mathcal{S}$.

The notion of optimal value function in BMDPs requires an ordering rule for intervals. We can define two different possible orderings:

$$[l_1, u_1] \leqslant_{pes} [l_2, u_2] \Leftrightarrow \begin{cases} l_1 < l_2, \text{ or} \\ l_1 = l_2 \text{ and } u_1 \leqslant u_2 \end{cases} \tag{1.34}$$

$$[l_1, u_1] \leqslant_{opt} [l_2, u_2] \Leftrightarrow \begin{cases} u_1 < u_2, \text{ or} \\ u_1 = u_2 \text{ and } l_1 \leqslant l_2 \end{cases} \tag{1.35}$$

We use these orderings rules to partially order interval value functions in the following way:

$$V_{1\updownarrow} \leqslant V_{2\updownarrow} \iff V_{1\updownarrow}(s) \leqslant_* V_{2\updownarrow}(s) \quad \forall s \in \mathcal{S}, \tag{1.36}$$

with $\leqslant_*$ defined as $\leqslant_{pes}$ in (1.34) or $\leqslant_{opt}$ in (1.35).

As stated in [Givan, Leach, and Dean, 2000], there exists at least one optimistically and one pessimistically optimal policy:

$$V^*_{\updownarrow opt} = \max_{\pi \in \Pi} V_{\updownarrow,\pi} \text{ using } \leqslant_{opt} \text{ to order interval value functions,}$$
$$V^*_{\updownarrow pes} = \max_{\pi \in \Pi} V_{\updownarrow,\pi} \text{ using } \leqslant_{pes} \text{ to order interval value functions.}$$

In order to better understand the meaning of the optimal policies, we consider a game in which we choose a policy π and then a second player chooses an MDP $M \in M_\updownarrow$ to evaluate the policy. $\overline{V}^*_{opt}$ is the best value function we can obtain if the second player cooperates in the game, $\underline{V}^*_{pes}$ is the best value function obtainable if the second player is an adversary.

We define the Interval Value Iteration (IVI) algorithm that computes optimal value intervals, it is similar to the standard value iteration presented in subsection 1.2.1.2. In IVI the updating rule is:

$$V_{\updownarrow,k+1}(s) = \max_{a \in \mathcal{A}, \leqslant_*} \left[\min_{M \in M_\updownarrow} VI_{M,a}(\underline{V}_k)(s), \max_{M \in M_\updownarrow} VI_{M,a}(\overline{V}_k)(s) \right], \tag{1.37}$$

where $\leqslant_*$ is $\leqslant_{pes}$ or $\leqslant_{opt}$. Given an MDP M with transition function P and reward function R, an action $a \in \mathcal{A}$ and a value function v, $VI_{M,a}(v)(s)$ is a single iteration of policy evaluation, where the action is fixed and

$$VI_{M,a}(v)(s) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s'). \tag{1.38}$$

In (1.37) it is required to perform two iterations VI. The two value functions used to perform the iterations (i. e. the functions used instead of v in (1.38)) are obtained from the interval value function

$V_{\updownarrow,k}$. For the first iteration, $v = \underline{V}_k$, with $\underline{V}_k$ the lower bounds in $V_{\updownarrow,k}$; for the second one $v = \overline{V}_k$ with $\overline{V}_k$ the upper bounds in $V_{\updownarrow,k}$.

Instead of searching in the set $M_{\updownarrow}$, the MDP M that minimizes (maximizes) the expressions under min(max) operator in (1.37) can be directly obtained by computing an exact transition function P from the interval transition function $P_{\updownarrow}$ of $M_{\updownarrow}$. In order to do that, the arriving states $s' \in \mathcal{S}$ are sorted in increasing (decreasing) order according to their $\underline{V}(\overline{V})$. Then, for all the state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$ and given an ordering of arriving states $s'_1, s'_2, ..., s'_k$, we calculate the index r, with $1 \leqslant r \leqslant k$, that maximizes the following expression without letting it exceed 1:

$$\sum_{i=1}^{r-1} \overline{P}(s'_i|s, a) + \sum_{i=r}^{k} \underline{P}(s'_i|s, a). \tag{1.39}$$

The exact transition function $P(.|s, a)$ is defined by assigning the upper bound $\overline{P}(s'|s, a)$ to the transition probabilities involving states s' with an index lower than r, the lower bound $\underline{P}(s'|s, a)$ to the transition probabilities involving states s' with an index greater than r and the probability that ensures $\sum_{s' \in \mathcal{S}} P(s'|s, a) = 1$ to the state with index r.

1.4.2 *Lipschitz MDPs*

We introduce the notion of *Lipschitz continuity* in order to define some properties of regularity in the MDP. These properties can be exploited in policy gradient algorithms, in [Pirotta, Restelli, and Bascetta (2015)] is shown how they can ensure a performance improvement at each iteration of policy-parameter updates. In this section we provide basic concepts related to Lipschitz continuity and useful bounds that will be exploited in our work.

Definition 1.2 (Lipschitz continuity)**.** Given two metric sets (X, d_X) and (Y, d_Y) where d_X and d_Y denote the corresponding metric functions, a function $f : X \to Y$ is called L_f-Lipschitz Continuous (LC) if

$$\forall (x_1, x_2) \in X^2, d_Y(f(x_1), f(x_2)) \leqslant L_f d_X(x_1, x_2). \tag{1.40}$$

The smallest L_f for which (1.40) holds is called the Lipschitz constant of f. For real-valued functions (e. g. the reward function R), we use the Euclidean distance as metric for the codomain distance. For the transition function P and the policies π we need to introduce a distance between probability distributions. We consider the Kantorovich or L^1-Wasserstein metric on probability measures, defined as follows.

Definition 1.3 (Kantorovich metric)**.** Given two probability measures p and q, the Kantorvich measure $\mathcal{K}(p, q)$ is:

$$\mathcal{K}(p, q) = \sup_f \left\{ \left| \int_x f(x) d\Big(p(x) - q(x)\Big) dx \right| : L_f \leqslant 1 \right\}. \tag{1.41}$$

In this work, we restrict our attention to Lipschitz-continuous MDPs and policies. We make similar assumptions as in [Pirotta, Restelli, and Bascetta, 2015]. For the MDP, we require both the continuity of the transition model and the reward:

Assumption 1 (Lipschitz MDP)**.** *For all* $s, \widetilde{s} \in \mathcal{S}$ *and* $a, \widetilde{a} \in \mathcal{A}$*:*

$$\mathcal{K}\left(P(\cdot|s, a), P(\cdot|\widetilde{s}, \widetilde{a})\right) \leqslant L_P \, d_{\mathcal{SA}}\left((s, a), (\widetilde{s}, \widetilde{a})\right), \tag{1.42}$$

$$|R(s, a) - R(\widetilde{s}, \widetilde{a})| \leqslant L_R \, d_{\mathcal{SA}}\left((s, a), (\widetilde{s}, \widetilde{a})\right), \tag{1.43}$$

for some positive real constants L_P *and* L_R*.*

where $d_{\mathcal{SA}}\left((s, a), (\widetilde{s}, \widetilde{a})\right) = \|s - \widetilde{s}\| + \|a - \widetilde{a}\|$ is the taxicab norm on $\mathcal{S} \times \mathcal{A}$. We also require our deterministic policy to be continuous both w.r.t. the input state and its parameters:

Assumption 2 (Lipschitz Policies)**.** *For all* $s, \widetilde{s} \in \mathcal{S}$ *and* $\theta, \widetilde{\theta} \in \Theta$*:*

$$\|\pi_\theta(s) - \pi_\theta(\widetilde{s})\| \leqslant L_{\pi_\theta} \|s - \widetilde{s}\|, \tag{1.44}$$

$$\left\|\pi_\theta(s) - \pi_{\widetilde{\theta}}(s)\right\| \leqslant L_\Theta \left\|\theta - \widetilde{\theta}\right\|, \tag{1.45}$$

for some positive real constants $\{L_{\pi_\theta}\}_{\theta \in \Theta}$ *and* L_Θ*.*

We use the euclidean norm to measure distances on $\mathcal{S}$, $\mathcal{A}$ and Θ, but everything works for general metrics. In the following, we will always assume that $L_P(1 + L_{\pi_\theta}) < \gamma^{-1}$. These assumptions are enough to guarantee the continuity of the value functions w.r.t. states and actions:

Lemma 1.1 (Rachelson and Lagoudakis (2010))**.** *Under Assumptions 1 and 2, for all* $s, \widetilde{s} \in \mathcal{S}$*,* $a, \widetilde{a} \in \mathcal{A}$ *and* $\theta \in \Theta$*:*

$$\left|V^\theta(s) - V^\theta(\widetilde{s})\right| \leqslant L_{V^\theta} \|s - \widetilde{s}\|, \tag{1.46}$$

$$\left|Q^\theta(s, a) - Q^\theta(\widetilde{s}, \widetilde{a})\right| \leqslant L_{Q^\theta} \, d_{\mathcal{SA}}\left((s, a), (\widetilde{s}, \widetilde{a})\right), \tag{1.47}$$

where $L_{Q^\theta} = \frac{L_R}{1 - \gamma L_P(1 + L_{\pi_\theta})}$ *and* $L_{V^\theta} = L_{Q^\theta}(1 + L_{\pi_\theta})$*.*

and also of the future-state distributions w.r.t. policy parameters:

Lemma 1.2 (Pirotta, Restelli, and Bascetta (2015))**.** *Under Assumptions 1 and 2, for all* $\theta, \widetilde{\theta} \in \Theta$*:*

$$\mathcal{K}\left(\delta^\theta, \delta^{\widetilde{\theta}}\right) \leqslant L_{\delta^\theta} \left\|\theta - \widetilde{\theta}\right\|, \tag{1.48}$$

where $L_{\delta^\theta} = \gamma L_P L_{\pi_\theta} / (1 - \gamma L_P(1 + L_{\pi_\theta}))$*.*

1.5 SAFE RL

1.6 STATE DISCRETIZATION

BIBLIOGRAPHY

Baek, Donghoon, Minho Hwang, Hansoul Kim, and Dong-Soo Kwon (June 2018). "Path Planning for Automation of Surgery Robot based on Probabilistic Roadmap and Reinforcement Learning." In: pp. 342–347 (cit. on p. xvii).

Deisenroth, Marc, Gerhard Neumann, and Jan Peters (Aug. 2013). *A Survey on Policy Search for Robotics*. Vol. 2 (cit. on p. 7).

Duan, Yan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel (2016). "Benchmarking Deep Reinforcement Learning for Continuous Control." In: *ICML*. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1329–1338 (cit. on p. 8).

Givan, Robert, Sonia M. Leach, and Thomas L. Dean (2000). "Bounded-parameter Markov decision processes." In: *Artif. Intell.* 122.1-2, pp. 71–109 (cit. on pp. 13, 14).

Knuth, D. E. (1974). "Computer Programming as an Art." In: *Communications of the ACM* 17.12, pp. 667–673 (cit. on p. 21).

Peters, Jan and Stefan Schaal (2008). "Reinforcement learning of motor skills with policy gradients." In: *Neural networks : the official journal of the International Neural Network Society* 21 4, pp. 682–97 (cit. on pp. 9, 11).

Pirotta, Matteo, Marcello Restelli, and Luca Bascetta (2015). "Policy gradient in Lipschitz Markov Decision Processes." In: *Machine Learning* 100.2-3, pp. 255–283 (cit. on pp. 15, 16).

Puterman, Martin L (2014). *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons (cit. on p. 2).

Rachelson, Emmanuel and Michail G. Lagoudakis (2010). "On the locality of action domination in sequential decision making." In: *ISAIM* (cit. on p. 16).

Sehnke, Frank, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber (Sept. 2008). "Policy Gradients with Parameter-Based Exploration for Control." In: pp. 387–396 (cit. on p. 11).

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller (June 2014). "Deterministic Policy Gradient Algorithms." In: *31st International Conference on Machine Learning, ICML 2014* 1 (cit. on p. 12).

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press (cit. on pp. xvii, 1).

Sutton, Richard S., David A. McAllester, Satinder P. Singh, and Yishay Mansour (1999). "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In: *NIPS* (cit. on pp. 7, 9).

Williams, Ronald J. (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning." In: *Machine Learning* 8, pp. 229–256 (cit. on p. 10).

APPENDIX EXAMPLE: CODE LISTINGS

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Knuth, "Computer Programming as an Art," 1974

A.1 THE `listings` PACKAGE TO INCLUDE SOURCE CODE

Source code is usually not part of the text of a thesis, but if it is an original contribution it makes sense to le the code speak by itself instead of describing it. The package `listings` provide the proper layout tools. Refer to its manual if you need to use it, an example is given in listing A.1.

Listing A.1: Code snippet with the recursive function to evaluate the pdf of the sum Z_N of N random variables equal to X.

```
std::vector<int> values_of_x(number_of_values_of_x,
  min_value_of_x);
for (unsigned int i = 1; i < number_of_values_of_x; i++) {
  values_of_x[i] = values_of_x[i - 1] + 1;
}
prob_x = 1.0 / number_of_values_of_x;
std::vector<std::vector<double> > p_z;
for (unsigned int idx = 0; idx < p_z.size(); idx++) {
  p_z[idx] = std::vector<double>(
    (max_value_of_x * (idx + 1) - min_value_of_x
      * (idx + 1)) + 1, INIT_VALUE);
}

double prob(int Z, int value_of_z) {
  if (value_of_z < min_value_of_x * Z ||
    value_of_z > max_value_of_x * Z) {
      return 0.0;
  }
  if (value_of_z < min_value_of_z ||
    value_of_z > max_value_of_z) {
      return 0.0;
  }
  int idx_value_of_z = -(min_value_of_z - value_of_z);
  int idx_N = Z - 1;
  if (p_z[idx_N][idx_value_of_z] == -2.0) {
    if (Z > 1) {
      double pp = 0.0;
      for (unsigned int i = 0; i < number_of_values_of_x; i++) {
        pp += prob(Z - 1, value_of_z - values_of_x[i], p);
      }
      p_z[idx_N][idx_value_of_z] = prob_x * pp;
    } else {
      if (Z == 1) {
        for (unsigned int j = 0; j < number_of_values_of_x; j++)
            {
          if (value_of_z == values_of_x[j]) {
            p_z[idx_N][idx_value_of_z] = prob_x;
            break;
          }
        }
      }
      if (p_z[idx_N][idx_value_of_z] == INIT_VALUE) {
        p_z[idx_N][idx_value_of_z] = 0.0;
      }
    }
  }
  return p_z[idx_N][idx_value_of_z];
}
```