POLITECNICO DI MILANO

Facoltà di Ingegneria

Scuola di Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in

Computer Science and Engineering – Ingegneria Informatica

Adaptive Batch Size for Safe Policy Gradient Methods

Supervisor:

MARCELLO RESTELLI

Assistant Supervisor:

MATTEO PIROTTA

Master Graduation Thesis by:

MATTEO PAPINI
Student Id n. 850421

Academic Year 2016-2017

ACKNOWLEDGMENTS

Todo

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

MDP	Markov Decision Process
RL	Reinforcement Learning
MC	Monte Carlo
TD	Temporal Difference
EM	Expectation Maximization
REPS	Relative Entropy Policy Search
CPI	Conservative Policy Iteration
USPI	Unique-parameter Safe Policy Improvement
MSPI	Multiple-parameter Safe Policy Improvement
LPI	Linearized Policy Improvement
TRPO	Trust Region Policy Optimization

ABSTRACT

Todo

SOMMARIO

Todo

1 INTRODUCTION

This is an introduction.

2 REINFORCEMENT LEARNING PRELIMINARIES

In this chapter we provide an introduction to the reinforcement learning framework and to policy gradient. The aim is to introduce concepts that will be used extensively in the following chapters. For a complete treatment of reinforcement learning, refer to [20]. In Section 2.1 we present the Markov Decision Process model and define the Reinforcement Learning problem. In Section 2.2 we provide an overview of Reinforcement Learning algorithms, while Section 2.3 is entirely dedicated to policy gradient methods.

2.1 MARKOV DECISION PROCESSES

In this section we provide a brief treatment on the model at the root of most Reinforcement Learning (RL) algorithms, including policy gradient methods: the Markov Decision Process (MDP). Following the opposite direction than the one usually adopted by Reinforcement Learning textbooks, we first formalize continuous MDPs. The discrete MDP model will be introduced later as a special case. This is justified by the scope of our work.

2.1.1 *The model*

Markov decision processes model the very general situation in which an agent interacts with a stochastic, partially observable environment in order to reach a goal. At each time step t, the agent receives observations from the environment. From the current observations and previous interactions, the agent builds a representation of the current state of the environment, s_t. Then, depending on s_t, it takes action a_t. Finally, it is given a (possibly negative) reward depending on how much its behavior is compliant with the goal, in the form of a scalar signal r_t. The dynamics of the environment, as represented by the agent, must satisfy two fundamental properties:

- Stationarity: the dynamics of the environment does not change over time.

- Markov property: s_{t+1} must depend only on s_t and a_t.

The dynamics can still be stochastic. Under this framework, the goal of the agent can be restated as the problem of collecting as much reward as possible. For now, and every time it is not specified otherwise, we will refer to continuing (i.e. never-ending) tasks, in which future rewards are exponentially discounted.

Formally, a discrete-time continuous Markov decision process is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu \rangle$, where:

- $\mathcal{S} \subseteq \mathbb{R}^n$ is an n-dimensional continuous state space. Elements of this set are the possible states of the environment as observed by the agent.
- $\mathcal{A} \subseteq \mathbb{R}^m$ is an $m-$dimensional continuous action space, from which the agent can pick its actions.
- $\mathcal{P}\colon \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{S})$ is a Markovian transition model, where $\mathcal{P}(s' \mid s, a)$ defines the transition density between states s and s' under action a, i.e. the probability distribution over the next state s' when the current state is s and the chosen action is a. In general, we denote with $\Delta(X)$ the set of probability distributions over X.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to [-R, R]$ is the reward function, such that $\mathcal{R}(s, a)$ is the expected value of the reward received by taking action a in state s and $R \in \mathbb{R}^+$ is the maximum absolute-value reward.
- $\gamma \in [0, 1)$ is the discount factor for future rewards.
- $\mu \in \Delta(\mathcal{S})$ is the initial state distribution, from which the initial state is drawn.

The behavior of the agent is modeled with as a stationary, possibly stochastic policy:

$$\pi\colon \mathcal{S} \mapsto \Delta(\mathcal{A}),$$

such that $\pi(s)$ is the probability distribution over the action a to take in state s. We denote with $\pi(a \mid s)$ the probability density of action a under state s. An agent is said to follow policy π if it draws actions from such conditional distribution. We say that a policy π is deterministic if for each $s \in \mathcal{S}$ there exists some $a \in \mathcal{A}$ such that $\pi(a \mid s) = 1$. With abuse of notation, we denote with $\pi(s)$ the action selected with probability 1 by a deterministic policy π in state s.

2.1.2 *Reinforcement Learning: problem formulation*

Given a MDP, the goal of Reinforcement Learning is to find an optimal policy π^* without an a-priori model of the environment, i.e. without knowing $\mathcal{P}, \mathcal{R}, \mu$. The optimal policy must be learned by direct interaction with the environment (which may be a real system or a simulation). The objective of the agent is to maximize the total discounted reward, called return ν:

$$\nu = \sum_{t=0}^{\infty} \gamma^t r_t$$

To evaluate how good a policy π is, we need a measure of performance. A common choice is the expected value of the return over π and the initial state distribution μ:

$$J_{\mu}^{\pi} = \mathbf{E}_{\mu,\pi}[\nu],$$

which we will sometimes call simply 'performance'. It is convenient to introduce a new distribution, called discounted stationary distribution or discounted future-state distribution d_{μ}^{π}:

$$d_{\mu}^{\pi}(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^{t}\Pr(s_t = s \mid \pi, \mu),$$

such that $d_{\mu}^{\pi}(s)$ is the probability of being in state s at any time step, after weighting time steps according to the discount factor. The stationary distribution allows to better define the expected return J as:

$$J_{\mu}^{\pi} = \int_{\mathcal{S}} d_{\mu}^{\pi}(s)\int_{\mathcal{A}}\pi(a \mid s)\mathcal{R}(s,a)\mathrm{d}a\mathrm{d}s.$$

By introducing the joint distribution $\zeta_{\mu,\pi}$ between d_{μ}^{π} and π, we can express the expected return even more compactly:

$$J_{\mu}^{\pi} = \mathop{\mathbf{E}}_{(s,a)\sim\zeta_{\mu,\pi}}[\mathcal{R}(s,a)]$$

The RL problem can then be formulated as a stochastic optimization problem:

$$\pi^{*} \in \arg\max_{\pi} J(\pi),$$

where π^{*} is the optimal policy. It is well known that when the space of possible policies is unconstrained, every MDP admits an optimal policy.

2.1.3 *Value functions*

Value functions provide a measure of how valuable a state, or a state-action pair, is under a policy π. They are powerful theoretical tools and are employed in many practical RL algorithms. The state value function $V^{\pi}\colon \mathcal{S} \mapsto \mathbb{R}$ assigns to each state the expected return that is obtained by following π starting from state s, and can be defined recursively by the following equation:

$$V^{\pi}(s) = \int_{\mathcal{A}}\pi(a|s)\left(\mathcal{R}(s,a) + \gamma\int_{\mathcal{S}}\mathcal{P}(s'|s,a)V^{\pi}(s')\mathrm{d}s'\right)\mathrm{d}a,$$

which is known as Bellman's expectation equation. The expected return $J_{\mu}(\pi)$ can be expressed in terms of the state-value function as:

$$J_{\mu}^{\pi} = \int_{\mathcal{S}}\mu(s)V^{\pi}(s)\mathrm{d}s.$$

An optimal policy maximizes the value function in each state simultaneously, i.e. $\pi^{*} \in \arg\max_{\pi} V^{\pi}(s)\ \forall s \in \mathcal{S}$.

For control purposes, it is more useful to define an action-value function $Q^\pi \colon \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, such that $Q^\pi(s, a)$ is the return obtained starting from state s, taking action a and following policy π from then on. Also the action-value function is defined by a Bellman equation:

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

Finally, the difference between the two value functions is known as advantage function:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s).$$

Intuitively, $A(s, a)$ represents the advantage of taking action a in state s instead of drawing an action from the policy. The advantage function allows to define a useful measure of policy improvement. The policy advantage of policy π' w.r.t. π under initial state distribution μ is:

$$\mathbb{A}_{\pi,\mu}(\pi') = \int_{\mathcal{S}} d_\mu^\pi(s) \int_{\mathcal{A}} \pi'(a \mid s) A^\pi(s, a) da ds.$$

Intuitively, $\mathbb{A}_{\pi,\mu}(\pi')$ measures how much π' selects advantageous actions w.r.t. π.

2.2 OVERVIEW OF REINFORCEMENT LEARNING ALGORITHMS

2.2.1 *Partial taxonomy*

Reinforcement learning algorithms can be classified according to many different criteria.

First, it is useful to discriminate between the target policy, which is the output of the learning algorithm, and behavior policies, which are policies used to interact with the environment. This produces a first distinction:

- On-policy algorithms: the behavior policy coincides with the target policy.
- Off-policy algorithms: a target policy is learned independently from the adopted behavior policies.

If we consider the degree of interleaving between learning and experience, we have the following classification (which is independent on whether experience is real or simulated):

- Continuing algorithms: learning is on-line. The target policy is updated possibly at each time step.

- Episodic algorithms: experience is divided into finite segments, called episodes. The target policy is updated in between episodes.
- Batch algorithms: Learning ad experience are separated. The target policy is learned by looking at data previously collected with a set of behavior policies.

Most of the RL algorithms that we will consider are on-policy and episodic. An episode of length H under policy π is typically obtained by starting from a state drawn from μ, following π and stopping after H time steps. We call trajectory the sequence $\langle s_t, a_t, r_t \rangle_{t=0,\dots,H}$, where each reward r_t was obtained by taking action a_t in state s_t.

Another useful distinction is the following:

- Model-based algorithms: the agent has an explicit model of the environment.
- Model-free algorithms: the agent has no model of the environment.

When the model of the environment is exact, the RL problem reduces to a dynamic programming problem. A more interesting situation is when the environment dynamics must be estimated from data. However, we will mainly consider model-free algorithms.

Finally, most RL methods can be classified into the two following classes:

- Value function methods rely on the computation of a value function to learn better and better policies.
- Direct policy search methods directly search for an optimal solution in the space of policies.

Most value-function based algorithms fall into the policy iteration family. Policy iteration is actually a dynamic programming algorithm, but most model-free value-function based methods follow the general scheme of policy iteration, hence the name. We will briefly discuss policy iteration in the following. Policy gradient is a direct policy search method, and is thoroughly described in Section 2.3.

2.2.2 *Policy Iteration*

Policy iteration is typically applied to discrete MDPs. Discrete MDPs can be defined as a special case of MDPs, as defined in the previous section, where both the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are discrete, finite sets. All the definitions introduced so far can be adapted to the discrete case simply by replacing integrals with sums.

We report the general policy iteration scheme. Starting from an initial policy, $\pi \leftarrow \pi_0$, policy iteration alternates between the two following phases:

1. Policy evaluation: Given the current policy π, compute (or estimate) action-value function Q^π.

2. Policy improvement: Compute a better policy π' based on Q^π, then set $\pi \leftarrow \pi'$ and go to 1.

A typical policy update for phase 2 is the greedy policy improvement:

$$\pi'(s) \in \arg\max_{a \in \mathcal{A}} Q^\pi(s, a),$$

which employs deterministic policies. When the value function Q^π can be computed exactly for each state-action pair, policy iteration with greedy policy improvement is guaranteed to converge to an optimal policy.

In most RL problems the value function Q^π cannot be computed exactly, but can be estimated from samples. Given the sampling nature of the approach, some stochasticity must be included in the policy to ensure sufficient exploration of the state-action space. Several algorithms are known for discrete MDPs with reasonably small state and action spaces [20]. These algorithms are also called tabular, because the value function can be stored in a finite table. Tabular algorithms mainly differ for the way the value function is estimated. The two main families are Monte Carlo (MC) methods, which are episodic, and Temporal Difference (TD) algorithms, which are continuing. Convergence guarantees are available for most of these algorithms.

Unfortunately, tabular algorithms cannot be applied to the more general case of continuous MDPs. One possibility is to discretize the state and action spaces. This is possible, but not trivial, and may not be suitable for most tasks. Moreover, tabular methods are unfeasible even for discrete MDPs when the cardinality of the state-action space is too high. Another possibility is to replace the value function with a function approximator. The supervised learning literature provides very powerful function approximators, such as neural networks, which are able to approximate functions of arbitrary complexity starting from a finite parametrization. However, the convergence properties of policy iteration methods employing function approximators are not well understood. The problem is that two levels of approximation are used: the function approximator tries to map action-state pairs to returns, which in turn are sampled from limited experience, creating a sort of moving target situation. Although function approximation have been successfully applied to complex RL problems, its lack of theoretical guarantees makes it unfeasible for many relevant applications. These problems are discussed in more depth in Section 2.3.1.

2.2.3 *Policy search*

Contrary to value function approaches, direct policy search consists in exploring a subset of policy space directly to find an optimal policy. The fact that it does not require to estimate the value function makes this approach suitable for continuous MDPs, such as the one arising in robotics. For a recent survey on policy search methods refer to [5]. The following section is entirely dedicated to policy gradient. Other notable model-free policy search methods are Expectation Maximization (EM) algorithms and Relative Entropy Policy Search (REPS).

2.3 POLICY GRADIENT

In this section we will focus on a direct policy search approach with good theoretical properties and promising practical results: policy gradient.

2.3.1 *Why policy gradient*

As mentioned in Section 2.2.2, policy iteration requires to employ function approximators to deal with continuous MDPs. We now present in more detail the main problems of this approach:

- Function approximation methods have no guarantee of convergence, not even to locally optimal policies, and there are even examples of divergence. Besides compromising the learning process, in some application such instabilities may damage the system.

- Greedy policies based on approximated value functions can be highly sensitive to small perturbations in the state, making them unfeasible for tasks characterized by noise, e.g. control tasks with noisy sensors.

- Given the complexity of the value function representation (think of a multi-layer perceptron), it is difficult to isolate undesired behaviors, such as potentially dangerous actions.

Policy gradient methods bypass all these problems by searching directly for the optimal policy. Convergence to a local optimum is guaranteed [10], uncertainty in the state have more controllable effects, policies can be made safe by design and prior domain knowledge can be exploited to design ad-hoc policies for specific tasks. Moreover, approximate value functions can still be used to guide the search for the optimal policy, such as in actor-critic methods.

Indeed, policy gradient methods (and other direct policy search approaches) have been successfully applied to complex control tasks. In [19] the authors were able to solve a robot standing task, where a (simulated) biped robot must keep an erect position while perturbed by external forces. In [9] Kober and Peters used policy search in combination with imitation learning to teach the Ball-in-a-Cup game to a real robotic arm. A similar approach was used in [12] for a baseball swing task. In [11] policy search was used to play simulated robot table tennis.

2.3.2 *Definition*

Policy gradient is a kind of policy search based on gradient descent, in which the search is restricted to a class of parameterized policies:

$$\Pi_\theta = \{\pi_\theta : \theta \in \boldsymbol{\Theta} \subseteq \mathbb{R}^m\},$$

where θ is the parameter vector. The parametric policy π_θ can have any shape, but is required to be differentiable in θ. Like all gradient descent methods, the parameter vector is updated in the direction of the gradient of a performance measure, which is guaranteed to be the direction of maximum improvement. In our case the performance measure is the expected return $J_\mu^{\pi_\theta}$, which we overload as $J_\mu(\theta)$ for the sake of readability. The gradient of the expected return w.r.t. the parameter vector, $\nabla_\theta J_\mu(\theta)$, is called policy gradient. Starting from an arbitrary initial parametrization θ_0, policy gradient methods performs updates of the kind:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J_\mu(\theta),$$

where $\alpha \geqslant 0$ is the learning rate, also called step size. The update is performed at each learning iteration. Convergence to at least a local optimum is guaranteed by the gradient descent update.

2.3.3 *Policy Gradient Theorem and eligibility vectors*

The Policy Gradient Theorem is a result by Sutton [21] that relates the policy gradient to the value function Q^π:

Theorem 2.1 (Continuous version of Theorem 1 from [21])**.** For any MDP

$$\nabla_\theta J_\mu(\theta) = \int_{\mathcal{S}} d_\mu^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a \mid s) Q^\pi(s, a) da ds.$$

The Policy Gradient Theorem is of key importance for many policy gradient algorithms.

2.3.4 *Stochastic gradient descent*

In many practical tasks it is impossible, or unfeasible, to compute the exact policy gradient $\nabla_\theta J_\mu(\theta)$, but a gradient estimate $\hat{\nabla}_\theta J_\mu(\theta)$ can be estimated from sample trajectories. The gradient estimate is often averaged over a number N of such trajectories, also called a batch. The size N of the batch is called batch size. The stochastic gradient descent update is:

$$\theta \leftarrow \theta + \alpha \hat{\nabla}_\theta J_\mu(\theta),$$

where, at each learning iteration, N trajectories need to be performed. Convergence to a local optimum is still guaranteed, provided that the angular difference between $\nabla_\theta J_\mu(\theta)$ and $\hat{\nabla}_\theta J_\mu(\theta)$ is less than $\pi/2$.

2.3.5 *A general policy gradient algorithm*

We provide here the skeleton of a general policy gradient algorithm:

Algorithm 2.1 A general policy gradient algorithm

```
set initial policy parametrization $\theta$
while not converged do
    perform N trajectories following policy $\pi_\theta$
    compute policy gradient estimate $\hat{\nabla}_\theta J_\mu(\theta)$
    $\theta \leftarrow \theta + \alpha \hat{\nabla}_\theta J_\mu(\theta)$
```

Many elements are left open in this schema: how to pick batch size N and step size α, what class of policies to use, how to compute the gradient estimate and how to devise a practical stopping condition. In the following sections we will see actual policy gradient algorithms, all adding details or introducing small variations to this general schema.

2.3.6 *Finite differences*

Among the oldest policy gradient methods we find finite-difference methods, which arose in the stochastic simulation literature. The idea is to perturb the policy parameter θ many times, obtaining $\theta + \Delta\theta_1 \dots \theta + \Delta\theta_K$, and estimate for each perturbation the expected return difference $\Delta\hat{J}_i \simeq J(\theta + \Delta\theta_i) - J(\theta)$, by performing a number of sample trajectories. After collecting all these data, the policy gradient can be estimated by regression as:

$$\hat{\nabla}_\theta J_\mu(\theta) = (\boldsymbol{\Delta\Theta}^T \boldsymbol{\Delta\Theta})^{-1} \boldsymbol{\Delta\Theta}^T \boldsymbol{\Delta\hat{J}}$$

where $\boldsymbol{\Delta\Theta}^T = [\Delta\theta_1, \dots, \Delta\theta_K]^T$ and $\boldsymbol{\Delta\hat{J}} = [\Delta\hat{J}_1, \dots, \Delta\hat{J}_K]^T$. This method is easy to implement and very efficient when applied to deterministic tasks or pseudo-random number simulations. In real control tasks though, perturbing the policy parametrization without incurring in instability may not be trivial and noise can have a disastrous impact on performance. Moreover, this kind of gradient estimation requires a large number of trajectories.

2.3.7 *Likelihood Ratio*

The likelihood ratio approach allows to estimate the gradient even from a single trajectory, without the need of perturbing the parametrization. This method is due to Williams [24], but is better understood from the perspective of the later Policy Gradient Theorem [21]. By applying trivial differentiation rules to the expression of the policy gradient (see Theorem 2.1), we have:

$$\nabla_\theta J_\mu(\theta) = \int_{\mathcal{S}} d_\mu^\pi(s) \int_{\mathcal{A}} \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a) \mathrm{d}a \mathrm{d}s.$$

This is known as the REINFORCE trick. The term

$$\nabla_\theta \log \pi_\theta(a \mid s) = \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$

has many names: eligibility vector, characteristic eligibility, score function and likelihood ratio. In terms of the joint distribution $\zeta_{\mu,\theta}$, the policy gradient is just:

$$\nabla_\theta J_\mu(\theta) = \mathop{\mathbf{E}}_{(s,a)\sim\zeta_{\mu,\theta}} \left[\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a) \right].$$

Since sampling state-action pairs from ζ is equivalent to following policy π_θ, the gradient can be estimated from a single trajectory as:

$$\tilde{\nabla}_\theta J_\mu(\theta) = \langle \nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a) \rangle_H,$$

where $\langle \cdot \rangle_H$ denotes sample average over H time steps. Of course a more stable estimate can be obtained by averaging again over a batch of N trajectories:

$$\hat{\nabla}_\theta J_\mu(\theta) = \langle \langle \nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a) \rangle_H \rangle_N.$$

2.3.8 *Baselines*

A problem of the REINFORCE approach is the large variance of the estimate, which results in slower convergence. The Policy Gradient Theorem still holds when an arbitrary baseline $b(s)$ is subtracted from the action-value function, as shown in [24]:

$$\nabla_\theta J_\mu(\theta) = \mathop{\mathbf{E}}_{(s,a)\sim\zeta} \left[\nabla_\theta \log \pi_\theta(a \mid s) (Q^\pi(s,a) - b(s)) \right].$$

The baseline can be chosen to reduce the variance of the gradient estimate without introducing any bias, speeding up convergence. A natural choice of baseline is the state-value function V^π, because it normalizes the value of the actions according to the overall value of the state, preventing overestimation of the possible improvement. This is equivalent to employ the advantage function in place of the action-value function:

$$\begin{aligned}\nabla_\theta J_\mu(\theta) &= \underset{(s,a)\sim\zeta}{\mathbf{E}}\left[\nabla_\theta \log \pi_\theta(a \mid s)(Q^\pi(s,a) - V^\pi(s))\right] \\ &= \underset{(s,a)\sim\zeta}{\mathbf{E}}\left[\nabla_\theta \log \pi_\theta(a \mid s)(A^\pi(s,a))\right].\end{aligned}$$

The usage of the advantage function with the eligibility vector has an intuitive meaning: to follow the policy gradient direction means to assign to advantageous actions a higher probability of being taken.

2.3.9 *The REINFORCE algorithm*

In practical tasks, the exact value function Q^π is not available and must be estimated. Episodic likelihood ratio algorithms estimate it from the total return at the end of the episode, assigning to each visited state-action pair a value in retrospective. The REINFORCE gradient estimator [24] uses the total return directly:

$$\hat{\nabla}_\theta J_\mu^{RF}(\theta) = \left\langle \sum_{k=1}^{H} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \left(\sum_{l=1}^{H} \gamma^{l-1} r_l^n - b \right) \right\rangle_N .$$

A problem of the REINFORCE algorithm is the high variance of the gradient estimate, which can be reduced by using a proper baseline. A variance-minimizing baseline can be found in [13]:

$$b = \frac{\left\langle \left(\sum_{k=1}^{H} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \right)^2 \sum_{l=1}^{H} \gamma^{l-1} r_l^n \right\rangle_N}{\left\langle \left(\sum_{k=1}^{H} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \right)^2 \right\rangle_N}.$$

Note that, in the case $|\theta| > 1$, all operations among vectors are to be intended per-component or, equivalently, each gradient component $\hat{\nabla}_{\theta_i} J_\mu(\theta)$ must be computed separately.

2.3.10 *The PGT/G(PO)MDP algorithm*

A problem of the REINFORCE gradient estimator is that the value estimate for (s_k^n, a_k^n) depends on past rewards. Two refinements are the PGT gradient estimator [22]:

$$\hat{\nabla}_\theta J_\mu^{PGT}(\theta) = \left\langle \sum_{k=1}^{H} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \left(\sum_{l=k}^{H} \gamma^{l-1} r_l^n - b \right) \right\rangle_N ,$$

and the G(PO)MDP gradient estimator [3]:

$$\hat{\nabla}_\theta J_\mu^{G(PO)MDP}(\theta) = \left\langle \sum_{l=1}^{H} \left(\sum_{k=1}^{l} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \right) (\gamma^{l-1} r_l^n - b_l) \right\rangle_N .$$

Although the algorithms are different, the gradient estimator that is computed is exactly the same, i.e. $\hat{\nabla}_\theta J_\mu^{PGT}(\theta) = \hat{\nabla}_\theta J_\mu^{G(PO)MDP}(\theta)$, as shown in [13]. From now on we will refer to the PGT form. In the case $b = 0$, the PGT gradient estimation has been proven to suffer from less variance than REINFORCE [25] under mild assumptions. A variance-minimizing baseline for PGT is provided in [13]. Differently from the REINFORCE case, a separate baseline b_l is used for each time step l:

$$b_l = \frac{\left\langle \left(\sum_{k=1}^{l} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \right)^2 \gamma^{l-1} r_l^n \right\rangle_N}{\left\langle \left(\sum_{k=1}^{l} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \right)^2 \right\rangle_N}.$$

2.3.11 *Natural gradients*

The algorithms described so far use the so-called vanilla policy gradient, i.e. an unbiased estimate of the true gradient w.r.t. policy parameters. Since such algorithms perform a search in parameter space Θ, the performance is strongly depends on the policy parametrization. Natural policy gradients allow to search directly in policy space, independently on the parametrization. The natural policy gradient update rule is [7]:

$$\theta \leftarrow \theta + F_\theta \nabla_\theta J_\mu(\theta),$$

where F_θ is the Fisher information matrix:

$$\begin{aligned} F_\theta &= \int_{\mathcal{S}} d_\mu^\pi(s) \int_{\mathcal{A}} \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s)^T da ds \\ &\simeq \langle \nabla_\theta \log \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s)^T \rangle_H . \end{aligned}$$

The parameter update is in the direction of a rotated policy gradient. Since the angular difference is never more than $\pi/2$ [2], convergence to a local optimum is still guaranteed. Although a good policy parametrization can yield better performance for specific problems, natural policy gradients offer a more general approach and have shown effective in practice [12].

2.3.12 *Actor-critic methods*

Actor-critic methods try to combine policy search and value function estimation to perform low-variance gradient updates. The "actor" is a parametrized policy π_θ, which can be updated as in vanilla policy gradient algorithms. For this reason, the policy gradient methods described so far are also called actor-only. The "critic" is an estimate of the action-value function $\hat{Q}_{\boldsymbol{w}}$, where $\boldsymbol{w}$ is a parameter vector. The approximate value-function methods mentioned in Section 2.2.2 can be thought as critic-only. The RL literature provides a great number of methods to learn a value function estimate from experience [20]. We provide an example of episodic value-function parameter update:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta \langle (R_t - \hat{Q}_{\boldsymbol{w}}(s_t, a_t)) \nabla_{\boldsymbol{w}} \hat{Q}_{\boldsymbol{w}}(s_t, a_t) \rangle_H,$$

where $R_t = \sum\limits_{k=0}^{\infty} \gamma^k r_{t+k}$ and β is just a learning rate. Actor and critic meet when $\hat{Q}_{\boldsymbol{w}}$ is used to compute the policy gradient used to update θ:

$$\theta \leftarrow \theta + \langle \nabla_\theta \log \pi_\theta(a \mid s) \hat{Q}_{\boldsymbol{w}}(s, a) \rangle_H. \tag{2.1}$$

This approach exploits the predictive power of approximate value functions while preserving the nice theoretical properties of the policy gradient. Under some assumption on the critic function (see Theorem 2 in [21]), the Policy Gradient Theorem is still valid when $\hat{Q}_{\boldsymbol{w}}$ is used in place of the true value function Q^π:

$$\nabla_\theta J_\mu(\theta) = \int_{\mathcal{S}} d_\mu^\pi(s) \int_{\mathcal{A}} \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) \hat{Q}_{\boldsymbol{w}}(s, a) da ds.$$

This means that parameter update (2.1) is still in the direction of the policy gradient, guaranteeing convergence to a local optimum. Additionally, an approximator of the value function V^π can be used as a baseline to reduce variance. For a recent survey of actor-critic methods refer to [6], which reports also examples of the usage of natural gradients in combination with the actor-critic approach.

2.3.13 *Common policy classes*

Although there are no restrictions on parameterized policies, some specific classes are used extensively in applications and have well studied properties. One of the most popular is the Gaussian policy, which in its most general form is defined as:

$$\pi_\theta(a \mid s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \left(\frac{a - \mu_\theta(s)}{\sigma_\theta(s)} \right)^2 \right).$$

A common form is the Gaussian policy with fixed standard deviation σ and mean linear in a feature vector $\boldsymbol{\phi}(s)$:

$$\pi_\theta(a \mid s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \left(\frac{a - \theta^T \boldsymbol{\phi}(s)}{\sigma} \right)^2 \right),$$

having eligibility vector:

$$\nabla_\theta \log \pi_\theta(a \mid s) = \frac{(a - \theta^T \boldsymbol{\phi}(s))\boldsymbol{\phi}(s)}{\sigma^2}.$$

When the action space $\mathcal{A}$ is discrete, a common choice is the softmax policy:

$$\pi_\theta(a \mid s) = \frac{e^{\theta^T \boldsymbol{\phi}(s,a)}}{\sum\limits_{a' \in \mathcal{A}} e^{\theta^T \boldsymbol{\phi}(s,a')}},$$

having eligibility vector:

$$\nabla_\theta \log \pi_\theta(a \mid s) = \boldsymbol{\phi}(s, a) - \sum_{a' \in \mathcal{A}} \pi_\theta(s \mid a)\boldsymbol{\phi}(s, a')$$

3 SAFE REINFORCEMENT LEARNING: STATE OF THE ART

The aim of this chapter is to present the state of the art in safe approaches to Reinforcement Learning, also known as conservative or monotonically improving methods.

3.1 PROBLEM DEFINITION AND MOTIVATION

3.1.1 *The policy oscillation problem*

Safe RL algorithms address a problem known as policy oscillation, first studied in the context of approximate policy iteration and not yet fully understood (see for instance [4], [23]). Policy oscillation affects approximate RL algorithms and manifests as a non-monotonic performance $J_\mu(\pi)$ over learning iterations. In parametric methods, a consequence is that the convergence of the parameter vector to a stationary point may not correspond to an increasing performance, but to an oscillating one. Even when the trend of $J_\mu(\pi)$ is generally positive, large oscillations can be unacceptable for some applications. Among the main recognized causes of policy oscillation are the entity of approximation errors and the magnitude of policy updates. To explain the phenomenon in more detail, we restrict our analysis to policy gradient methods. However, the nature of the problem is fundamentally the same also for policy iteration methods, as shown by Wagner [23]. For policy gradient methods, non-monotonic improvement means that in some cases $J_\mu(\theta + \alpha\hat{\nabla}_\theta J_\mu(\theta)) \leqslant J_\mu(\theta)$. The two main causes are:

1. The magnitude of the updates, controlled by the step size α. Large parameter updates may result in overshooting in policy space, producing oscillation around a local optimum.

2. The policy gradient estimation error. The variance of $\hat{\nabla}_\theta J_\mu(\theta)$ affects the quality of the updates in terms of direction, and may even produce updates in non-improving directions.

3.1.2 *The need for safe methods*

Both of the policy oscillation causes that have been described pose a trade-off between monotonic improvement and speed of convergence. Using small step sizes results in more conservative policy updates, reducing performance oscillation. On the other hand, a small step size can heavily affect convergence speed. Similarly, computing more accurate policy gradient estimates typically requires a large number of samples. Even if the batch size does not affect directly the number of iterations to convergence, it can affect the actual learning time, especially when collecting sample trajectories is time-wise expensive. The resulting trade-off is similar to the well known exploration-exploitation dilemma, and is particularly evident when learning is performed on real systems. In a simulated task with no time limitations, neither convergence speed nor oscillation are major issues, unless convergence is compromised. Instead, in more realistic scenarios, oscillation itself may cause the following problems:

- Dangerous behavior: low performance peaks may correspond to large deviations from the initial policy. Even if the initial policy has been designed to be safe, such deviations may result in behavior able to damage the system.

- Poor long-term performance: Consider the scenario in which learning is performed simultaneously with production, e.g. to automatically adjust an existing optimal policy to small changes in the environment, either in a real-time manner or by temporary stopping production. Oscillations in the per-episode performance are paid in term of long-term performance when they are not justified by a relevant enough overall policy improvement. This loss is similar to the concept of regret caused by exploration.

- Selection of suboptimal policies: if the learning process has a fixed duration or must be stopped for some reason, oscillations may lead to the selection of a policy that is not the best seen so far, or is even worse than the initial policy. In highly stochastic environments it can be very difficult to manually reject such sub-par policies, and testing all the policies seen so far may be prohibitively expensive.

From all these problems comes the necessity of policy updates that are monotonically increasing.

3.2 SAFE POLICY ITERATION

In policy iteration, policy oscillation can arise each time approximation is used either in the evaluation step or in the improvement step. In this section we present the main safe approaches approximate to policy iteration, starting from the seminal work by Kakade and Langford [8].

3.2.1 *Conservative Policy Iteration*

In [8], Kakade and Langford address the policy oscillation problem by limiting the magnitude of the policy updates. Let π be the current policy and π' a greedy policy w.r.t. the latest action-value estimate, i.e. :

$$\pi'(a \mid s) \in \arg\max_{a \in \mathcal{A}} \hat{Q}^{\pi}(s, a)$$

The traditional approximate policy update is:

$$\pi_{new}(a \mid s) = \pi'(a \mid s).$$

Instead, Kakade and Langford proposed to update π with a convex combination of the greedy and the current policy:

$$\pi_{new}(a \mid s) = (1-\alpha)\pi(a \mid s) + \alpha\pi'(a \mid s), \tag{3.1}$$

with step size $\alpha \in [0, 1]$. When a step size $\alpha < 1$ is used, this is more conservative than a pure greedy improvement, producing a new policy that is more similar to the original one. The authors were able to compute a lower bound on the performance improvement:

$$J_\mu(\pi_{new}) - J_\mu(\pi) \geqslant \frac{\alpha}{1-\gamma}\left(\mathbb{A}_{\pi,\mu}(\pi') - \frac{2\alpha\gamma\epsilon}{1-\gamma(1-\alpha)}\right),$$

where $\epsilon = \max_{s \in \mathcal{S}} | \mathbf{E}_{a \sim \pi'}[A^{\pi}(s, a)] |$. While the positive term obviously depends on the advantage of the greedy policy, the negative one must be intended as a penalty for large policy updates. By maximizing this bound w.r.t. α, a step size is obtained that guarantees positive performance improvement as long as the advantage is positive, without totally sacrificing the speed of convergence:

$$\alpha^* = \frac{(1-\gamma)\mathbb{A}_{\pi,\mu}(\pi')}{4R} \tag{3.2}$$

$$J_\mu(\pi_{new}) - J_\mu(\pi) \geqslant \frac{\mathbb{A}^2_{\pi,\mu}(\pi')}{8R} \tag{3.3}$$

When the true advantage cannot be computed and an estimate must be used, monotonic improvement can still be guaranteed with high probability. The resulting algorithm is called Conservative Policy Iteration (CPI). This methodology inspired the development of several later safe methods.

3.2.2 *Unique and Multiple-Parameter Safe Policy Improvement*

Pirotta et al. [17] derived a more general bound on the performance improvement of a policy π' over π:

$$J_\mu(\pi_{new}) - J_\mu(\pi) \geqslant \mathbb{A}_{\pi,\mu}(\pi') - \frac{\gamma}{(1-\gamma)^2}\left\|\pi' - \pi\right\|^2_\infty \frac{\|Q^{\pi}\|_\infty}{2}, \tag{3.4}$$

where the penalty induced by large policy updates is made more evident by the term $\|\pi' - \pi\|_\infty^2$. Starting from this bound, a new algorithm called Unique-parameter Safe Policy Improvement (USPI) was developed. USPI uses the same update as CPI (3.1), but with a step size based on a better improvement bound, derived from 3.4. This results in better guaranteed performance improvements. In the same paper, the authors also propose a more general, state dependent policy update:

$$\pi_{new}(a \mid s) = (1 - \alpha(s))\pi(a \mid s) + \alpha(s)\pi'(a \mid s).$$

Using state-dependent step-sizes, it is possible to limit the update to the states in which the average advantage $\mathbf{E}_{a \sim \pi'}[A^\pi(s, a)]$ is positive, by setting $\alpha(s) = 0$ for all other states. This is the idea behind Multiple-parameter Safe Policy Improvement (MSPI). Although more general, this algorithm is not guaranteed to provide better performance improvements than USPI, although it is never worse than CPI. Moreover, the computation of the optimal step-sizes for 'good' states can only be done iteratively, limiting efficiency.

3.2.3 *Linearized Policy Improvement*

Bartlett et al. [1] propose a different policy update employing state-dependent mixture coefficients, which in this case can be computed efficiently. The policy update is:

$$\pi_{new}(a \mid s) = \nu(a \mid s)(1 + \alpha\Delta_\pi(s, a)),$$

where:

- $\nu(\cdot \mid s)$ is a policy such that:

  $$\mathop{\mathbf{E}}_{a \sim \pi}\left[\hat{Q}^\pi(s \mid a)\right] = \mathop{\mathbf{E}}_{a \sim \nu}\left[\hat{Q}^\pi(s \mid a)\right] + \mathop{\mathbf{Var}}_{a \sim \nu}\left[\hat{Q}^\pi(s \mid a)\right], \quad (3.5)$$

  which can be computed for each visited state with a binary search.

- $\Delta_\pi(s, a)$ is a normalized action-value function:

  $$\Delta_\pi(s, a) = \hat{Q}^\pi(s, a) - \mathop{\mathbf{E}}_{a \sim \nu}\left[\hat{Q}^\pi(s, a)\right], \quad (3.6)$$

- α is a step size.

The idea is the following: the original policy π is reshaped into:

$$\overline{\pi}(a \mid s) = \nu(a \mid s)(1 + \Delta_\pi(s, a)),$$

which has the same value function (in the absence of estimation error), i.e. $V^{\overline{\pi}} = V^\pi$. The negative term in Equation 3.6 ensures that $\overline{\pi}$ is a well defined distribution. Introducing a step size α allows to assign higher probability to actions with positive normalized value function Δ_π. A conservative choice for α is $1/\|\Delta_\pi\|_\infty$, but the authors also propose more practical alternatives. The resulting algorithm is called Linearized Policy Improvement (LPI). The authors show that the performance improvement guarantee of LPI is better than the one of CPI, at the same time allowing for bigger policy updates, hence faster convergence.

3.3 SAFE POLICY GRADIENT

3.3.1 *The role of the step size*

Most of the safe policy gradient methods that have been proposed so far focus on the selection of the proper step size α. The step size regulates the magnitude of the parameter updates. Although the relationship between $\Delta\theta$ and the actual change in the policy is parametrization-dependent (at least for vanilla policy gradients) and may be non-trivial, in general small step sizes produce small changes in the policy, hence small performance oscillations. However, reducing the step-size also reduces the convergence speed, which is a major drawback in most applications. The traditional solution is to employ line search to select the optimal step size for each problem. Although effective in supervised learning [10], this approach can be prohibitively expensive in RL. An alternative is to make the step size adaptive, i.e. to compute it at each learning iteration from collected data.

3.3.2 *Adaptive step-size*

Pirotta et al. [14] apply the approach of Kakade and Langford [8] to policy gradient. Starting from a continuous version of Bound 3.4, they derive a series of bounds on performance improvement which can be maximized w.r.t. the step size α. We report here a bound for Gaussian policies which does not require to compute the action-value function:

$$\begin{aligned} J_\mu(\theta') - J_\mu(\theta) \geqslant \alpha \|\nabla_\theta J_\mu(\theta)\|_2^2 - \\ \alpha^2 \frac{RM_\phi^2 \|\nabla_\theta J_\mu(\theta)\|_1^2}{(1-\gamma)^2\sigma^2} \left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)} \right), \end{aligned} \tag{3.7}$$

where M_ϕ is a bound on state features $\phi(s)$. At each learning iteration, given $\nabla_\theta J_\mu(\theta)$, the step size that optimizes this bound can be computed:

$$\alpha^* = \frac{\|\nabla_\theta J_\mu(\theta)\|_2^2 (1-\gamma)^3 \sigma^3 \sqrt{2\pi}}{RM_\phi^2 \|\nabla_\theta J_\mu(\theta)\|_1^2 (2|\mathcal{A}|(1-\gamma) + \sqrt{2\pi}\sigma\gamma)}$$

Updating the policy parameters with α^* guarantees a constant performance improvement:

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \frac{\|\nabla_\theta J_\mu(\theta)\|_2^4 (1-\gamma)^3 \sigma^3 \sqrt{2\pi}}{2RM_\phi^2 \|\nabla_\theta J_\mu(\theta)\|_1^2 (2|\mathcal{A}|(1-\gamma) + \sqrt{2\pi}\sigma\gamma)}$$

When the policy gradient cannot be computed exactly, monotonicity can be guaranteed with high probability, depending on the entity of the estimation error. The authors provide methods, based on Chebyshev's inequality, to select the number of samples necessary to achieve a desired improvement probability δ, for the cases of REINFORCE and PGT gradient estimators.

The same authors [16] follow a similar approach to derive tighter bounds for the case of Lipschitz-continuous MDPs. A Lipschitz MDP is a MDP with Lipschitz-continuous transition model $\mathcal{P}$ and reward function $\mathcal{R}$. We omit the mathematical details, but intuitively the Lipschitz hypothesis is reasonable when the dynamics of the environment are sufficiently smooth. Starting from this hypothesis, and constraining the policy to be Lipschitz-continuous as well, the authors are able to prove the Lipschitz continuity of the policy gradient $\nabla_\theta J_\mu(\theta)$ and provide an efficient algorithm to compute an optimal step size based on this fact.

3.3.3 *Beyond the step size*

Although the step size is crucial in controlling policy oscillation, we saw that also the entity of the estimation error plays a role. In likelihood ratio methods, the variance of the policy gradient estimate $\hat{\nabla}_\theta J_\mu(\theta)$ can be limited by averaging over a large number of trajectories. In this case, the batch size N can affect policy improvement. In [14] the effect of N is recognized but not exploited: it is still a metaparameter that must be selected beforehand by a human expert. As in the case of the step size, line search algorithm are usually too expensive given the time that is typically required to collect a sample trajectory. For the same reason, an adaptive batch size should take into consideration the cost of collecting more samples, which cannot be neglected in practical applications. In the following chapter we address this problems to derive a cost-sensitive adaptive batch size that guarantees monotonic improvement.

3.4 OTHER SAFE METHODS

In this section we describe other safe approaches to reinforcement learning.

3.4.1 *Trust Region Policy Optimization*

A problem with the methods proposed so far is that they tend to be over conservative, negatively affecting convergence speed. Schulman et al. [18] revisit the CPI approach to develop an effective, although strongly heuristic, safe direct policy search algorithm. Given a parametrized policy π_θ, said θ the current parameter and θ' a generic updated parameter, the following is shown to hold (adapted to our notation):

$$J_\mu(\theta) \geqslant L_\theta(\theta') - \alpha H_{max}(\theta, \theta'),$$

where $L_\theta(\theta') = J_\mu(\theta) + \underset{s \sim d_\mu^{\pi_\theta}, a \sim \pi_{\theta'}}{\mathbf{E}} [A_{\pi_\theta}(s, a)]$ is a first order approximation of the performance improvement, $\alpha = \frac{2 \max_{s \in \mathcal{S}} \mathbf{E}_{a \sim \pi_{\theta'}}[A^\pi(s,a)]}{(1-\gamma)^2}$ is a penalty coefficient and $H_{max}(\theta, \theta') = \max_{s \in \mathcal{S}} H(\pi_\theta, \pi\theta)$, where $H(\cdot, \cdot)$ is the Kullback-Liebler divergence between two distributions. The traditional approach would be to maximize the bound:

$$\theta' = \arg\max_{\tilde{\theta} \in \Theta} L_\theta(\tilde{\theta}) - \alpha H_{max}(\theta, \tilde{\theta}),$$

but the theoretically justified penalty coefficient α leads to updates that are too conservative. Instead, the authors propose to resolve to the following constrained optimization problem:

$$\theta' = \arg\max_{\tilde{\theta} \in \Theta} L_\theta(\tilde{\theta}) \tag{3.8}$$

$$\text{subject to } H_{max}(\theta, \tilde{\theta}) \leqslant \delta. \tag{3.9}$$

Constraint 3.9 is called trust region constraint and its intuitive role is to limit the update to a neighborhood of the current parameter θ to avoid oscillation. A series of approximations to this exact method lead to the Trust Region Policy Optimization (TRPO) algorithm, which shows to be effective on complex tasks.

3.4.2 *Thomas*

3.4.3 *Petrik*

4

ADAPTIVE BATCH SIZE FOR POLICY GRADIENT

5 NUMERICAL SIMULATIONS

6 CONCLUSION

This is a conclusion.

BIBLIOGRAPHY

[1] Yasin Abbasi-Yadkori, Peter L Bartlett, and Stephen J Wright. "A Fast and Reliable Policy Improvement Algorithm." In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. 2016, pp. 1338–1346 (cit. on p. 20).

[2] Shun-Ichi Amari. "Natural gradient works efficiently in learning." In: *Neural Computation* 10.2 (Feb. 1998), pp. 251–276. ISSN: 0899-7667 (cit. on p. 14).

[3] Jonathan Baxter and Peter L. Bartlett. "Infinite-Horizon Policy-Gradient Estimation." In: *Journal of Artificial Intelligence Research* 15 (2001), pp. 319–350 (cit. on p. 14).

[4] DimitriP. Bertsekas. "Approximate policy iteration: a survey and some new methods." English. In: *Journal of Control Theory and Applications* 9.3 (2011), pp. 310–335. ISSN: 1672-6340 (cit. on p. 17).

[5] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. "A Survey on Policy Search for Robotics." In: *Foundations and Trends in Robotics* 2.1-2 (2013), pp. 1–142 (cit. on p. 9).

[6] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients." In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 42.6 (2012), pp. 1291–1307 (cit. on p. 15).

[7] Sham Kakade. "A Natural Policy Gradient." In: *Advances in Neural Information Processing Systems 14 (NIPS 2001)*. Ed. by Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani. MIT Press, 2001, pp. 1531–1538 (cit. on p. 14).

[8] Sham Kakade and John Langford. "Approximately Optimal Approximate Reinforcement Learning." In: *ICML*. Morgan Kaufmann, 2002, pp. 267–274 (cit. on pp. 18, 19, 21).

[9] Jens Kober and Jan Peters. "Policy Search for Motor Primitives in Robotics." In: *Advances in Neural Information Processing Systems 21*. Vol. 21. 2008, pp. 849–856 (cit. on p. 10).

[10] Jorge J. Moré and David J. Thuente. "Line Search Algorithms with Guaranteed Sufficient Decrease." In: *ACM Trans. Math. Softw.* 20.3 (Sept. 1994), pp. 286–307. ISSN: 0098-3500 (cit. on pp. 9, 21).

[11] Jan Peters, Katharina Mülling, and Yasemin Altun. "Relative Entropy Policy Search." In: *AAAI*. AAAI Press, 2010 (cit. on p. 10).

[12] Jan Peters and Stefan Schaal. “Natural Actor-Critic.” In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190 (cit. on pp. 10, 14).

[13] Jan Peters and Stefan Schaal. “Reinforcement learning of motor skills with policy gradients.” In: *Neural Networks* 21.4 (2008), pp. 682–697 (cit. on pp. 13, 14).

[14] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. “Adaptive Step-Size for Policy Gradient Methods.” In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Curran Associates, Inc., 2013, pp. 1394–1402 (cit. on pp. 21, 22).

[15] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. “Adaptive Step-Size for Policy Gradient Methods.” In: *Advances in Neural Information Processing Systems 26*. 2013, pp. 1394–1402 (cit. on p. 36).

[16] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. “Policy gradient in Lipschitz Markov Decision Processes.” In: *Machine Learning* 100.2-3 (2015), pp. 255–283 (cit. on p. 22).

[17] Matteo Pirotta, Marcello Restelli, Alessio Pecorino, and Daniele Calandriello. “Safe Policy Iteration.” In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. 3. JMLR Workshop and Conference Proceedings, 2013, pp. 307–315 (cit. on p. 19).

[18] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. “Trust Region Policy Optimization.” In: *ICML*. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 1889–1897 (cit. on p. 23).

[19] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. “Policy Gradients with Parameter-Based Exploration for Control.” In: *ICANN (1)*. Ed. by Vera Kurková, Roman Neruda, and Jan Koutník. Vol. 5163. Lecture Notes in Computer Science. Springer, 2008, pp. 387–396. ISBN: 978-3-540-87535-2 (cit. on p. 10).

[20] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981 (cit. on pp. 3, 8, 15).

[21] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. “Policy Gradient Methods for Reinforcement Learning with Function Approximation.” In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. Ed. by Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller. The MIT Press, 1999, pp. 1057–1063. ISBN: 0-262-19450-3 (cit. on pp. 10, 12, 15).

[22] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. “Policy Gradient Methods for Reinforcement Learning with Function Approximation.” In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. 1999, pp. 1057–1063 (cit. on p. 13).

[23] Paul Wagner. “A reinterpretation of the policy oscillation phenomenon in approximate policy iteration.” In: *Advances in Neural Information Processing Systems 24*. 2011, pp. 2573–2581 (cit. on p. 17).

[24] Ronald J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning.” In: *Machine Learning* 8.3 (1992), pp. 229–256. ISSN: 1573-0565 (cit. on pp. 12, 13).

[25] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. “Analysis and Improvement of Policy Gradient Estimation.” In: *NIPS*. Ed. by John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger. 2011, pp. 262–270 (cit. on p. 14).

A EXTENSION TO SOFTMAX POLICIES

Lemma A.1. *If π_θ is a softmax policy:*

$$\left|\frac{\partial^2 \pi_\theta(a \mid s)}{\partial\theta_i \partial\theta_j}\right| \leqslant 6M_\phi^2$$

Proof.

$$\begin{aligned}\left|\frac{\partial^2 \pi_\theta(a \mid s)}{\partial\theta_i \partial\theta_j}\right| &= \Bigg|\pi_\theta(a \mid s)\Bigg(\phi_j(s,\cdot)\phi_i(s,\cdot) - \phi_i(s,\cdot)\underset{a'\sim\pi_\theta}{\mathbf{E}}[\phi_j(s,\cdot)] - \phi_j(s,\cdot)\underset{a'\sim\pi_\theta}{\mathbf{E}}[\phi_i(s,\cdot)] \\ &\quad +2\underset{a'\sim\pi_\theta}{\mathbf{E}}[\phi_i(s,\cdot)]\underset{a'\sim\pi_\theta}{\mathbf{E}}[\phi_j(s,\cdot)] - \underset{a'\sim\pi_\theta}{\mathbf{E}}[\phi_i(s,\cdot)\phi_j(s,\cdot)]\Bigg)\Bigg| \\ &\leqslant 6M_\phi^2\end{aligned}$$

■

Lemma A.2. *If π_θ is a softmax policy and $\theta' = \theta + \Lambda\nabla_\theta J_\mu(\theta)$:*

$$\pi_{\theta'}(a \mid s) - \pi_\theta(a \mid s) \geqslant \nabla_\theta\pi_\theta(a \mid s)^T \Lambda\nabla_\theta J_\mu(\theta) - 6M_\phi^2 \left\|\Lambda\nabla_\theta J_\mu(\theta)\right\|_1^2$$

Proof. It follows from the application of Lemma A.1 to Taylor's expansion. ■

Lemma A.3. *If π_θ is a softmax policy and $\theta' = \theta + \Lambda\nabla_\theta J_\mu(\theta)$:*

$$\left\|\pi_{\theta'} - \pi_\theta\right\|_\infty^2 \leqslant 4M_\phi \left\|\Lambda\nabla_\theta J_\mu(\theta)\right\|_1$$

Proof. By Pinsker's inequality:

$$\left\|\pi_{\theta'} - \pi_\theta\right\|_\infty^2 = \sup_s \left\|\pi_{\theta'} - \pi_\theta\right\|_1^2 \leqslant \sup_s \left(2H(\pi_{\theta'} \,\|\, \pi_\theta)\right),$$

where $H(\cdot \,\|\, \cdot)$ is the Kullback-Liebler divergence, which in this case can be bounded as:

$$\begin{aligned} H(\pi_{\theta'}, \pi_\theta) &= \sum_{a\in\mathcal{A}} \frac{e^{\theta'^T\phi(s,a)}}{\sum_{a'\in\mathcal{A}} e^{\theta'^T\phi(s,a')}} \log \frac{e^{\theta'^T\phi(s,a)}\sum_{a'\in\mathcal{A}} e^{\theta^T\phi(s,a')}}{e^{\theta^T\phi(s,a)}\sum_{a'\in\mathcal{A}} e^{\theta'^T\phi(s,a')}} \\ &= \underset{a\sim\pi_{\theta'}}{\mathbf{E}}\left[\Delta\theta^T\phi(s,a) + \log\frac{\sum_{a'\in\mathcal{A}} e^{\theta^T\phi(s,a')}}{\sum_{a'\in\mathcal{A}} e^{\theta'^T\phi(s,a')}}\right] \\ &= \underset{a\sim\pi_{\theta'}}{\mathbf{E}}\left[\Delta\theta^T\phi(s,a) + \log\frac{\sum_{a'\in\mathcal{A}} e^{\theta^T\phi(s,a')}}{\sum_{a'\in\mathcal{A}} e^{\theta^T\phi(s,a')}e^{\Delta\theta^T\phi(s,a')}}\right] \\ &\leqslant \underset{a\sim\pi_{\theta'}}{\mathbf{E}}\left[\left\|\Delta\theta\right\|_1 M_\phi + \log\frac{\cancel{\sum_{a'\in\mathcal{A}} e^{\theta^T\phi(s,a')}}}{e^{-\left\|\Delta\theta\right\|_1 M_\phi}\cancel{\sum_{a'\in\mathcal{A}} e^{\theta^T\phi(s,a')}}}\right] \\ &= 2\left\|\Delta\theta\right\|_1 M_\phi = 2M_\phi \left\|\Lambda\nabla_\theta J_\mu(\theta)\right\|_1 \end{aligned}$$

The rest of the proof is trivial. ■

Theorem A.4. If π_θ is a softmax policy and $\theta' = \theta + \Lambda\nabla_\theta J_\mu(\theta)$:

$$\begin{aligned} J_\mu(\theta') - J_\mu(\theta) \geqslant{} & \nabla_\theta J_\mu(\theta)^T \Lambda \nabla_\theta J_\mu(\theta) \\ & - \frac{6M_\phi^2 |\mathcal{A}| R \left\| \Lambda \nabla_\theta J_\mu(\theta) \right\|_1^2}{(1-\gamma)^2} - \frac{2\gamma M_\phi R \left\| \Lambda \nabla_\theta J_\mu(\theta) \right\|_1}{(1-\gamma)^3} \end{aligned}$$

Proof. It's enough to plug the results from Lemmas A.2 and A.1 into Lemma 3.1 from [15]. ∎