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 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

ABSTRACT

Todo

Todo

1 INTRODUCTION

This is an introduction.

2

PRELIMINARIES ON POLICY GRADIENT

In this chapter we provide basic knowledge on the reinforcement learning framework and on the policy gradient approach. The aim is to introduce concepts that will be used extensively in the following chapters. For a complete treatment of reinforcement learning, refer to [12].

2.1 CONTINUOUS 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). Given the scope of our work, we will focus on continuous MDPs.

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, basing its decision on s_t and any other source of knowledge, 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 the Markov property: s_{t+1} must depend only on s_t and a_t. 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 draw 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 taken action is a.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to [-R, R]$ 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 of the environment is drawn.

The behavior of the agent can be defined 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.

2.1.2 *Problem formulation*

The goal of RL is to find an optimal policy π^* by exploiting the information obtained through the interaction with the environment. The objective is to maximize the total discounted reward, called return v:

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

To evaluate how good a policy π is, we need to compute the expected value of the return over π and the initial state distribution μ:

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

It is convenient to introduce a new distribution, called stationary distribution or future-state distribution d_μ^π:

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

which allows to 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 ζ between d_μ^π and π, we can express the expected return even more compactly:

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

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

$$\pi^* \in \arg\max_\pi J(\pi).$$

2.1.3 *Value functions*

Value functions provide a measure of how good 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:

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

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:

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

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.

2.2 THE POLICY GRADIENT APPROACH

In this section we describe in general terms a special class of RL algorithms known as policy gradient methods, and we report some theoretical results that are of key importance for any treatment of such methods.

2.2.1 *Definition*

Direct policy search is an approach to RL consisting in exploring a subset of policy space directly to find an optimal policy. It is opposed to the value function approach, in which the optimal policy is derived from an estimate of the action value function Q^π. The advantages of direct policy search over value function methods for continuous MDPs are discussed in the following chapter. For a recent survey on policy search methods refer to [3].

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

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

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 rename $J_\mu(\theta)$ for 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.

2.2.2 *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. With the term trajectory we mean an episode of the task starting from an initial state s_0, drawn from μ, and stopping after a fixed number H of time steps. 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 $nicefrac\pi2$.

2.2.3 *Policy Gradient Theorem and eligibility vector*

The Policy Gradient Theorem is a result by Sutton [13] that relates the policy gradient to the value function Q^π:

Theorem 2.1 (Continuous version of Theorem 1 from [13])**.** For any MDP

$$\nabla_\theta J_\mu(\theta) = \int_S d_\mu^\pi(s) \int_A \nabla_\theta \pi_\theta(a \mid s) Q^\pi(s,a) da\, ds.$$

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

2.2.4 *Common policy classes*

Although there are no restriction 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 $\boldsymbol{\phi}(s)$:

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

having eligibility vector:

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

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

$$\pi_\theta(a \mid s) = \frac{e^{\theta^\top \boldsymbol{\phi}(s,a)}}{\sum_{a' \in \mathcal{A}} e^{\theta^\top \boldsymbol{\phi}(s,a')}},$$

having eligibility vector:

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

2.2.5 *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 θ
while not converged **do**
  perform N trajectories following policy π_θ
  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 section we will see actual policy gradient algorithms, all adding details or introducing small variations to this general schema.

3 STATE OF THE ART

In this chapter we provide an overview of the most commonly used policy gradient methods, with a focus on safe approaches.

3.1 POLICY GRADIENT IN GENERAL

In this section we present common policy gradient algorithms. The aim is not to provide an exhaustive survey of policy gradient methods, but to give a proper context to the main topic of this and the following chapter: safe policy gradients.

3.1.1 *Why policy gradient*

Traditional RL methods rely on the computation (or the estimation) of the action-value function Q^π. These so-called value function methods were first proposed for solving simple MDPs characterized by small, discrete state and action spaces. They are also known as tabular methods, since the value function can be stored in a finite table. In this limited scenario, tabular algorithms have strong convergence guarantees and perform well in practice. However, they become unfeasible when the number of possible states and actions is too big or even infinite, such as in continuous MDPs. The natural extension of tabular methods is to employ the function approximation tools known from supervised learning, such as neural networks, to represent Q^π. However, this approach have many problems, especially when applied to control tasks:

- Value function approximation methods have no guarantee of convergence, not even to locally optimal policies. In some applications, divergence may even damage the system.
- Greedy policies based on approximated value functions can be highly sensitive to small perturbations in the state, being 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 potentially dangerous behaviors.

Policy gradient methods bypass these problems by searching directly for the optimal policy. Convergence to a local optimum is guaranteed

(also for stochastic methods), uncertainty in the state have controllable effects and policies can be made safe by design. Moreover, prior domain knowledge can be exploited to design policies suited for the specific tasks.

Indeed, policy gradient methods have been successfully applied to complex control tasks. In [11] 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 [6] 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 [8] for a baseball swing task. In [7] policy search was used to play simulated robot table tennis.

3.1.2 *From finite differences to likelihood ratio*

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:

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

where $\boldsymbol{\Delta\Theta}^\top = [\Delta\theta_1, \dots, \Delta\theta_K]^\top$ and $\boldsymbol{\Delta\hat{J}} = [\Delta\hat{J}_1, \dots, \Delta\hat{J}_K]^\top$. 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, gradient estimation requires a large number of trajectories.

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 [15], but is better understood from the perspective of the Policy Gradient Theorem [13]. By applying trivial differentiation rules to the expression of the policy gradient (see Theorem 2.1), we have:

$$\nabla_\theta J_\mu(\theta) = \int_S d_\mu^\pi(s) \int_A \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a) da\, ds.$$

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, policy score and, of course, likelihood ratio. In terms of the joint distribution ζ, the policy gradient is just:

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

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:

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

3.1.3 *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 [15]:

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

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:

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

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

3.1.4 *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 [15] 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 [9]:

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

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.

3.1.5 *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 [14]:

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

and the G(PO)MDP gradient estimator [2]:

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

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 [9]. 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 [16] under mild assumptions. A variance-minimizing baseline for PGT is provided in [9]. Differently from the REINFORCE case, a separate baseline b_l is used for each time step l:

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

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

3.1.6 *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 [5]:

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

where F_θ is the Fisher information matrix:

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

The parameter update is in the direction of a rotated policy gradient. Since the angular difference is never more than $\pi/2$ [1], 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 [8].

3.1.7 *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 critic-only. The "critic" is an estimate of the action-value function $\hat{Q}_w$, where w is a parameter vector. The approximate value-function methods briefly mentioned in Section 3.1.1 can be thought as critic-only. The RL literature provides a great number of methods to learn a value function estimate from experience [12]. We provide an example of episodic value-function parameter update:

$$w \leftarrow w + \beta \langle (R_t - \hat{Q}_w(s_t, a_t)) \nabla_w \hat{Q}_w(s_t, a_t) \rangle_H,$$

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

$$\theta \leftarrow \theta + \langle \nabla_\theta \log \pi_\theta(a \mid s) \hat{Q}_w(s, a) \rangle_H. \tag{3.1}$$

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 [13]), the Policy Gradient Theorem is still valid when $\hat{Q}_w$ is used in place of the true value function Q^π:

$$\nabla_\theta J_\mu(\theta) = \int_S d_\mu^\pi(s) \int_A \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) \hat{Q}_w(s,a) da ds.$$

This means that parameter update (3.1) is still in the direction of the policy gradient, guaranteeing convergence to a local optimum. For a recent survey of actor-critic methods refer to [4], which reports also examples of the usage of natural gradients in combination with the actor-critic approach.

3.2 SAFE POLICY GRADIENT METHODS

In this section we present the state of the art on monotonically improving policy gradient methods, which are also known as conservative or safe approaches. In the following we will refer to them as safe policy gradient methods.

3.2.1 *The need for safe policy gradients*

Exploration is a key component of RL algorithms. Through exploration, new states are discovered and new actions are tried and evaluated, increasing the agent's knowledge of the environment. The information gain, though, is paid in terms of performance, since gathering new knowledge takes precious time that could be spent in performing a good policy, based on the knowledge that has already been acquired. This is known as the exploration-exploitation dilemma. In policy gradient methods, exploration can be regulated at two levels: the intrinsic stochasticity of the policy (e.g. by changing the standard deviation of a Gaussian policy) and the parameter updates. We will focus on the latter, although the two problems are not entirely independent.

In terms of the policy performance $J_\mu(\theta)$, exploratory updates can produce oscillations. If we could plot the exact value of $J_\mu(\theta)$ over learning iterations, the policy gradient methods described in the previous subsections would show positive-trend zig-zag progressions. When learning is performed on a simulated task, it is natural to favor exploration, since all that counts is how good the final policy is. This is no longer true when learning is performed on a real system, like in the scenarios presented in the introduction. In this case, uncontrolled exploratory behavior can cause one or both of the following issues:

- 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: when learning is performed simultaneously with production, e.g. to automatically adjust an existing optimal policy to small changes in the environment, oscillations in the per-episode performance are paid in term of long-term performance when they are not justified by a relevant enough policy improvement.
- 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.

From these problems comes the necessity of policy updates that are monotonically increasing, i.e. safe policy gradients.

3.2.2 *The step-size problem*

Most of the safe policy gradient methods that have been proposed 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. To face this trade-off, the step size can be made adaptive, e.g. by making it a function of the policy gradient. Many of the methods described in the remaining part of this section follow this approach, which is also the starting point of our main contribution.

3.2.3 *State of the art*

3.2.4 *What to do next*

4

JOINT STEP-SIZE AND BATCH-SIZE OPTIMIZATION

5 NUMERICAL SIMULATIONS

6 CONCLUSION

This is a conclusion.

BIBLIOGRAPHY

[1] 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. 13).

[2] 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. 12).

[3] 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. 5).

[4] 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. 14).

[5] 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. 13).

[6] 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).

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

[8] Jan Peters and Stefan Schaal. "Natural Actor-Critic." In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190 (cit. on pp. 10, 13).

[9] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients." In: *Neural Networks* 21.4 (2008), pp. 682–697 (cit. on p. 12).

[10] 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. 26).

[11] 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).

[12] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981 (cit. on pp. 3, 13).

[13] 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. 6, 10, 14).

[14] 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. 12).

[15] 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. 10, 11).

[16] 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. 12).

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.

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

■

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

$$\pi_{\theta'}(a \mid s) - \pi_\theta(a \mid s) \geqslant \nabla_\theta \pi_\theta(a \mid s)^\top \Lambda \nabla_\theta J_\mu(\theta) - 6M_\phi^2 \left\|\Lambda\nabla_\theta J_\mu(\theta)\right\|_1^2$$

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:

$$\begin{aligned}H(\pi_{\theta'}, \pi_\theta) &= \sum_{a\in\mathcal{A}} \frac{e^{\theta'^\top \phi(s,a)}}{\sum_{a'\in\mathcal{A}} e^{\theta'^\top \phi(s,a')}} \log \frac{e^{\theta'^\top \phi(s,a)} \sum_{a'\in\mathcal{A}} e^{\theta^\top \phi(s,a')}}{e^{\theta^\top \phi(s,a)} \sum_{a'\in\mathcal{A}} e^{\theta'^\top \phi(s,a')}} \\ &= \mathop{\mathbf{E}}_{a\sim\pi_{\theta'}}\left[\Delta\theta^\top \phi(s,a) + \log \frac{\sum_{a'\in\mathcal{A}} e^{\theta^\top \phi(s,a')}}{\sum_{a'\in\mathcal{A}} e^{\theta'^\top \phi(s,a')}}\right] \\ &= \mathop{\mathbf{E}}_{a\sim\pi_{\theta'}}\left[\Delta\theta^\top \phi(s,a) + \log \frac{\sum_{a'\in\mathcal{A}} e^{\theta^\top \phi(s,a')}}{\sum_{a'\in\mathcal{A}} e^{\theta^\top \phi(s,a')} e^{\Delta\theta^\top \phi(s,a')}}\right] \\ &\leqslant \mathop{\mathbf{E}}_{a\sim\pi_{\theta'}}\left[\left\|\Delta\theta\right\|_1 M_\phi + \log \frac{\cancel{\sum_{a'\in\mathcal{A}} e^{\theta^\top \phi(s,a')}}}{e^{-\left\|\Delta\theta\right\|_1 M_\phi} \cancel{\sum_{a'\in\mathcal{A}} e^{\theta^\top \phi(s,a')}}}\right] \\ &= 2\left\|\Delta\theta\right\|_1 M_\phi = 2M_\phi \left\|\Lambda\nabla_\theta J_\mu(\theta)\right\|_1\end{aligned}$$

The rest of the proof is trivial. ■

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

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \nabla_\theta J_\mu(\theta)^\top \Lambda \nabla_\theta J_\mu(\theta) - \frac{6M_\Phi^2 |\mathcal{A}| R \, \|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2}{(1-\gamma)^2} - \frac{2\gamma M_\Phi R \, \|\Lambda \nabla_\theta J_\mu(\theta)\|_1}{(1-\gamma)^3}$$

Proof. It's enough to plug the results from Lemmas A.2 and A.1 into Lemma 3.1 from [10]. ■