Neurocomputing 72 (2009) 1508–1524

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Gaussian process dynamic programming

Marc Peter Deisenroth [a,b,*], Carl Edward Rasmussen [a,c], Jan Peters [c,d]

[a] Department of Engineering, University of Cambridge, Cambridge, UK
[b] Faculty of Informatics, Universität Karlsruhe (TH), Germany
[c] Max Planck Institute for Biological Cybernetics, Tübingen, Germany
[d] University of Southern California, Los Angeles, CA, USA

ARTICLE INFO

Available online 12 January 2009

Keywords:
Reinforcement learning
Optimal control
Dynamic programming
Gaussian processes
Bayesian active learning
Policy learning

ABSTRACT

Reinforcement learning (RL) and optimal control of systems with continuous states and actions require approximation techniques in most interesting cases. In this article, we introduce Gaussian process dynamic programming (GPDP), an approximate value function-based RL algorithm. We consider both a classic optimal control problem, where problem-specific prior knowledge is available, and a classic RL problem, where only very general priors can be used. For the classic optimal control problem, GPDP models the unknown value functions with Gaussian processes and generalizes dynamic programming to continuous-valued states and actions. For the RL problem, GPDP starts from a given initial state and explores the state space using Bayesian active learning. To design a fast learner, available data have to be used efficiently. Hence, we propose to learn probabilistic models of the a priori unknown transition dynamics and the value functions on the fly. In both cases, we successfully apply the resulting continuous-valued controllers to the under-actuated pendulum swing up and analyze the performances of the suggested algorithms. It turns out that GPDP uses data very efficiently and can be applied to problems, where classic dynamic programming would be cumbersome.

1. Introduction

Reinforcement learning (RL) is based on the principle of experience-based, goal-directed learning. In contrast to supervised learning, where labels are provided from an external supervisor, an RL algorithm must be able to learn from experience collected through interaction with the surrounding world. The objective in RL is to find a strategy, which optimizes a long-term performance measure, such as cumulative reward or cost. RL is similar to the field of optimal control although the fields are traditionally separate. In contrast to optimal control, RL does not necessarily assume problem-specific prior knowledge or an intricate understanding of the world. However, if we call the RL algorithm "controller" and identify actions with the "control signal" we have a one-to-one mapping from RL to optimal control if the surrounding world is fully known. In a general setting, however, an RL algorithm has to explore the world and collect information about it. Since RL is inherently based on collected experience, it provides an intuitive setup for sequential decision-making under uncertainty in autonomous learning.

The RL setup requires to automatically extract information and to *learn* structure from collected data. Learning is important when data sets are very complex or simply too large to find an underlying structure by hand. The learned structure is captured in the form of a statistical model that compactly represents the data. Bayesian data analysis aims to make inferences for quantities about which we wish to learn by using probabilistic models for quantities we observe. The essential characteristic of Bayesian methods is their explicit use of probability theory for quantifying uncertainty in inferences based on statistical data analysis. Without any notion of uncertainty, the RL algorithm would be too confident and claim exact knowledge, which it actually does not have. Representation and incorporation of uncertainties in RL is particularly important in the early stages of learning when the data set is still very sparse. Algorithms based on over-confident models can fail to yield good results due to model bias as reported by Atkeson and Santamaría [2] and Atkeson and Schaal [3]. Hence, it is important to quantify current knowledge appropriately. However, a major drawback of Bayesian methods is that they are computationally costly and the posterior distribution is often not analytically tractable.

Dynamic programming (DP) is a general and efficient method of solving sequential optimization problems under uncertainty. Due to the work of Bellman [4], Howard [19], Kalman [22], and many others, DP became a standard approach to solve optimal control problems. However, only in case of linear systems with

* Corresponding author at: Department of Engineering, University of Cambridge, Trumpington Street, Cambridge, UK.
E-mail address: mpd37@cam.ac.uk (M.P. Deisenroth).

0925-2312/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2008.12.019

quadratic cost and Gaussian noise, exact global solutions are known [5]. Similarly, many RL algorithms are based on DP techniques comprising value iteration and policy iteration methods, details of which are given by Sutton and Barto [53], Bertsekas and Tsitsiklis [7]. However, solving a nonlinear optimal control or RL problem for continuous-valued states and actions is challenging and requires approximation techniques in general.

In continuous-valued state and action domains, discretization is commonly used for approximations if required computations are no longer analytically tractable. However, the number of cells in a discretized space does not only depend on the dimensionality and the difficulty of the problem, but also on the time-sampling frequency. The higher the sampling rate, the smaller the size and the larger the number of cells required. Therefore, even low-dimensional problems can be infeasible to solve in discretized spaces. Function approximators address discretization problems and generalize to continuous-valued domains as described for instance by Bertsekas and Tsitsiklis [7] or Sutton and Barto [53]. The key idea is to model the DP value function in a function space rather than representing this function as a table of values at discrete input locations. Parametric function approximators, such as polynomials or radial basis function networks often used for this purpose, but they are only capable of modeling the unknown function within their corresponding model classes. A fundamental problem of parametric function approximators is that the model class is fixed before having observed any data. Often, it is hard to know ahead of time which class of functions will be appropriate. In general, the restriction to a wrong class of functions may result in diverging RL algorithms as shown by Gordon [18], Ormoneit and Sen [38]. Non-parametric regression techniques are generally more flexible than parametric models. "Non-parametric" does not imply that the model is parameter-free, but that the number and nature of the parameters are flexible and not fixed in advance.

Gaussian processes (GPs) combine both flexible non-parametric modeling and tractable Bayesian inference as described by Rasmussen and Williams [48]. The basic idea of non-parametric inference is to use data to infer an unknown quantity based on general prior assumptions. Often, this means using statistical models that are infinite dimensional [55]. Matheron [33] and others introduced GPs to geostatistics decades ago under the name *kriging*. They became popular in the machine learning community in the 1990s through work by Williams and Rasmussen [56] and the thesis by Rasmussen [44]. Recently they got introduced to the control community by Murray-Smith and Sbarbaro [35], Murray-Smith et al. [36] or Kocijan et al. [26], for instance.

GP regression allows for an appropriate uncertainty treatment in RL when approximating unknown functions. For value function and model learning the use of GPs in RL has for instance been discussed for model-free policy iteration by Engel et al. [13,14], for model-based control by Rasmussen and Kuss [47], Murray-Smith and Sbarbaro [35], Rasmussen and Deisenroth [45], and model-based value iteration by Deisenroth et al. [10,11]. Furthermore Ghavamzadeh and Engel [16] discussed GPs in the context of actor-critic methods.

In this article, we introduce and analyze the Gaussian process dynamic programming (GPDP) algorithm. GPDP is a value function-based RL algorithm that generalizes DP to continuous state and action spaces and which belongs to the family of fitted value iteration algorithms [18]. The central idea of GPDP is to utilize non-parametric, Bayesian GP models to describe the value functions in the DP recursion. We will consider both a classic optimal control problem, where much problem-specific prior knowledge is available, and a classic RL problem, where only very general assumptions can be made a priori. In particular, the transition dynamics will be unknown. To solve the RL problem, we will introduce a novel online algorithm that interleaves dynamics learning and value function learning. Moreover, Bayesian active learning is used to deal with the exploration–exploitation trade-off.

The structure of this article is as follows. Section 2 briefly introduces optimal control, RL, and GPs. In Section 3, GPDP is introduced in the context of an optimal control setting, where the transition dynamics are fully known. Furthermore, it will be discussed how a discontinuous, globally optimal policy can be learned if sufficient problem-specific knowledge is available. GPDP will be applied to the illustrative under-actuated pendulum swing up, a nonlinear optimal control problem introduced by Atkeson [1]. In Section 4, we consider a general RL setting, where only very general priors are available. We introduce a very data-efficient, fast learning RL algorithm, which builds probabilistic models of the transition dynamics and value functions on the fly. Bayesian active learning is utilized to efficiently explore the state space. We compare this novel algorithm to the neural fitted Q (NFQ) iteration by Riedmiller [50]. Section 5 summarizes the article.

2. Background

Throughout this article, we consider discrete-time systems

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}, \tag{1}$$

where $\mathbf{x}$ denotes the state, $\mathbf{u}$ the control signal (action), and $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_w)$ a Gaussian distributed noise random variable, where $\mathbf{\Sigma}_w$ is diagonal. Moreover, k is a discrete-time index. The transition function f mapping a state–action pair to a successor state is assumed to evolve smoothly over time.

2.1. Optimal control and RL

Both optimal control and RL aim to find a policy that optimizes a long-term performance measure. A policy π is a mapping from a state space $\mathbb{R}^{n_x}$ into a control space $\mathbb{R}^{n_u}$ that assigns a control signal to each state. In many cases, the performance measure is defined as the expected cumulative cost over a certain time interval. For an initial state $\mathbf{x}_0 \in \mathbb{R}^{n_x}$ and a policy π, the (discounted) expected cumulative cost of a finite N-step optimization horizon is

$$V^\pi(\mathbf{x}_0) := \mathrm{E}\left[\gamma^N g_{\mathrm{term}}(\mathbf{x}_N) + \sum_{k=0}^{N-1} \gamma^k g(\mathbf{x}_k, \mathbf{u}_k)\right], \tag{2}$$

where k indexes discrete time. Here, $\mathbf{u} := \pi(\mathbf{x})$ is the control signal assigned by policy π. The function g_{term} is a control-independent terminal cost that incurs at the last time step N. The immediate cost is denoted by $g(\mathbf{x}_k, \mathbf{u}_k)$. The discount factor $\gamma \in (0, 1]$ weights future cost. An optimal policy π^* for the N-step problem minimizes Eq. (2) for any initial state $\mathbf{x}_0$. The associated state-value function V^* satisfies Bellman's equation

$$V^*(\mathbf{x}) = \min_{\mathbf{u}} (g(\mathbf{x}, \mathbf{u}) + \gamma \mathrm{E}_{\mathbf{x}'}[V^*(\mathbf{x}') | \mathbf{x}, \mathbf{u}]) \tag{3}$$

for all states $\mathbf{x}$. The successor state for a given state–action pair $(\mathbf{x}, \mathbf{u})$ is denoted by $\mathbf{x}'$. The state–action value function Q^* is defined by

$$Q^*(\mathbf{x}, \mathbf{u}) = g(\mathbf{x}, \mathbf{u}) + \gamma \mathrm{E}_{\mathbf{x}'}[V^*(\mathbf{x}') | \mathbf{x}, \mathbf{u}], \tag{4}$$

such that $V^*(\mathbf{x}) = \min_{\mathbf{u}} Q^*(\mathbf{x}, \mathbf{u})$ for all $\mathbf{x}$. In general, finding an optimal policy π^* that leads to Eq. (3) is hard. Assuming time-additive cost and Markovian transitions,[1] the minimal expected

[1] The successor state $\mathbf{x}'$ only depends on the current state–action pair $(\mathbf{x}, \mathbf{u})$.