Engineering Applications of Artificial Intelligence 76 (2018) 158–169

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Interpretable policies for reinforcement learning by genetic programming

Daniel Hein [a,b,*], Steffen Udluft [b], Thomas A. Runkler [a,b]

[a] Technical University of Munich, Department of Informatics, Boltzmannstr. 3, 85748 Garching, Germany
[b] Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany

ARTICLE INFO

Keywords:
Interpretable
Reinforcement learning
Genetic programming
Model-based
Symbolic regression
Industrial benchmark

ABSTRACT

The search for interpretable reinforcement learning policies is of high academic and industrial interest. Especially for industrial systems, domain experts are more likely to deploy autonomously learned controllers if they are understandable and convenient to evaluate. Basic algebraic equations are supposed to meet these requirements, as long as they are restricted to an adequate complexity. Here we introduce the *genetic programming for reinforcement learning* (GPRL) approach based on model-based batch reinforcement learning and genetic programming, which autonomously learns policy equations from pre-existing default state–action trajectory samples. GPRL is compared to a straightforward method which utilizes genetic programming for symbolic regression, yielding policies imitating an existing well-performing, but non-interpretable policy. Experiments on three reinforcement learning benchmarks, i.e., mountain car, cart–pole balancing, and industrial benchmark, demonstrate the superiority of our GPRL approach compared to the symbolic regression method. GPRL is capable of producing well-performing interpretable reinforcement learning policies from pre-existing default trajectory data.

1. Introduction

This work introduces a genetic programming (GP) approach for autonomously learning interpretable reinforcement learning (RL) policies from previously recorded state transitions. Despite the search of interpretable RL policies being of high academic and industrial interest, little has been published concerning human interpretable and understandable policies trained by data driven learning methods (Maes et al., 2012). Recent research results show that using fuzzy rules in batch RL settings can be considered an adequate solution to this task (Hein et al., 2017b, 2018b). However, in many cases the successful use of fuzzy rules requires prior knowledge about the shape of the membership functions, the number of fuzzy rules, the relevant state features, etc. Moreover, for some problems the policy representation as a set of fuzzy rules might be generally unfavorable by some domain experts. Our *genetic programming for reinforcement learning* (GPRL) approach learns policy representations which are represented by basic algebraic equations of low complexity.

The GPRL approach is motivated by typical industrial application scenarios like wind or gas turbines. For industrial systems, low-level control is realized by dedicated expert-designed controllers, which guarantee safety and stability. However, we observed that high-level control is usually implemented by default control strategies, provided by best practice approaches or domain experts who are maintaining the system, based on personal experience and knowledge about the system's dynamics. One reason for the lack of autonomously generated real-world controllers is that modeling system dependencies for high-level control by a first principle model is a complicated and often infeasible approach. Since in many real-world applications such representations cannot be found, training high-level controllers has to be performed on data samples from the system. RL is capable of yielding high-level controllers based solely on available system data.

RL is concerned with learning a policy for a system that can be modeled as a Markov decision process (Sutton and Barto, 1998). This policy maps from system states to actions in the system. Repeatedly applying an RL policy generates a trajectory in the state–action space (Section 3). Based on our experience, learning such RL controllers in a way that produces interpretable high-level controllers is of high interest, especially for real-world industry problems, since interpretable solutions are expected to yield higher acceptance from domain experts than black-box solutions.

In batch RL, we consider applications where online learning approaches, such as classical temporal-difference learning (Sutton, 1988), are prohibited for safety reasons, since these approaches require exploration of system dynamics. In contrast, batch RL algorithms generate a policy based on existing data and deploy this policy to the system after training. In this setting, either the value function or the system dynamics are trained using historic operational data comprising a set of four-tuples of the form (*observation*, *action*, *reward*, *next observation*), which is

* Corresponding author at: Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany.
E-mail address: daniel.hein@in.tum.de (D. Hein).

https://doi.org/10.1016/j.engappai.2018.09.007
Received 27 November 2017; Received in revised form 3 April 2018; Accepted 9 September 2018

referred to as a data batch. Research from the past two decades (Gordon, 1995; Ormoneit and Sen, 2002; Lagoudakis and Parr, 2003; Ernst et al., 2005) suggests that such batch RL algorithms satisfy real-world system requirements, particularly when involving neural networks (NNs) modeling either the state–action value function (Riedmiller, 2005a, b; Schneegaßet al., 2007a, b; Riedmiller et al., 2009) or system dynamics (Bakker, 2004; Schäfer, 2008; Depeweg et al., 2016). Moreover, batch RL algorithms are data-efficient (Riedmiller, 2005a; Schäfer et al., 2007) because batch data is utilized repeatedly during the training phase.

To the best of our knowledge, GP-generated policies have never been combined with a model-based batch RL approach (Section 2). In the proposed GPRL approach, the performance of a population of basic algebraic equations is evaluated by testing the individuals on a world model using the Monte Carlo method (Sutton and Barto, 1998). The combined return value of a number of action sequences is the fitness value that is maximized iteratively from GP generation to generation.

GPRL is a novel model-based RL approach, i.e., training is conducted on an environment approximation referred to as world model. Generating a world model from real system data in advance and training a GP policy using this model has several advantages. (i) In many real-world scenarios, data describing system dynamics is available in advance or is easily collected. (ii) Policies are not evaluated on the real system, thereby avoiding the detrimental effects of executing a bad policy. (iii) Expert-driven reward function engineering, yielding a closed-form differentiable equation, utilized during policy training is not required, i.e., it is sufficient to sample from the system's reward function and model the underlying dependencies by using supervised machine learning.

The remainder of this paper is organized as follows. The RL and GP methods employed in our framework are reviewed in Sections 3 and 4. Specifically, the problem of finding policies via RL is formalized as an optimization task. In addition, GP in general and the specific implementation that we used for experiments are motivated and presented. An overview of how the proposed GPRL approach is derived from different methods is given in Section 5. Experiments using three benchmark problems, i.e., the mountain car (MC) problem, the cart–pole balancing (CPB) task, and the industrial benchmark (IB), are described in Section 6. Experimental results are discussed in Section 7. The results demonstrate that the proposed GPRL approach can solve the benchmark problems and is able to produce interpretable RL policies. To benchmark GPRL, we compare the obtained results to an alternative approach in which GP is used to mimic an existing non-interpretable NN policy by symbolic regression.

2. Related work

GP has been utilized for creating rule-based policies since its introduction by Koza (1992). Since then, the field of GP has grown significantly and has produced numerous results that can compete with human-produced results, including controllers, game playing, and robotics (Koza, 2010). Keane et al. (2002) automatically synthesized a controller by using GP, outperforming conventional PID controllers for an industrially representative set of plants. Another approach using genetic algorithms for RL policy design is to learn a set of fuzzy "if-then" rules, by modifying membership functions, rule sets and consequent types (Juang et al., 2000). Recently, Koshiyama et al. (2014) introduced GPFIS, a genetic fuzzy controller based on multi-gene GP, and demonstrated the superiority in relation to other genetic fuzzy controllers on the cart-centering and the inverted pendulum problems. On the same benchmark, a movable inverted pendulum, Shimooka and Fujimoto (1999) applied GP to generate equations for calculating the control force by evaluating the individuals' performances on predefined fitness functions.

A fundamental drawback with all of the former methods is that in many real-world scenarios such dedicated expert generated fitness functions do not exist. In RL the goal is to derive well-performing policies only by (i) interacting with the environment, or by (ii) extracting knowledge out of pre-generated data, running the system with an arbitrary policy (Sutton and Barto, 1998). (i) is referred to as the online RL problem, for which Q-learning methods are known to produce excellent results. For (ii), the offline RL problem, model-based algorithms are usually more stable and yield better performing policies (Hein et al., 2017b).

GP in conjunction with online RL Q-learning has been used in Downing (2001) on standard maze search problems and in Kamio and Iba (2005) to enable a real robot to adapt its action to a real environment. Katagiri et al. (2002) introduced genetic network programming (GNP), which has been applied to online RL in Mabu et al. (2002) and improved by Q-tables in Mabu et al. (2004). In these publications, the efficiency of GNP for generating RL policies has been discussed. This performance gain, in comparison to standard GP, comes at the cost of interpretability, since complex network graphs have to be traversed to compute the policy outputs.

Gearhart (2003) examined GP as a policy search technique for Markov Decision Processes. Given a simulation of the Freecraft tactical problem, he performed Monte Carlo simulations to evaluate the fitness of each individual. Note that such exact simulations are usually not available in industry. Similarly, in Maes et al. (2012) Monte Carlo simulations have been drawn in order to identify the best policies. However, the policy search itself has been performed by formalizing a search over a space of simple closed-form formulas as a multi-armed bandit problem. This means that all policy candidates have to be created in an initial step at once and are subsequently evaluated. The computational effort to follow this approach combinatorially explodes as soon as more complex solutions are required to solve more complicated control problems.

3. Model-based reinforcement learning

Inspired by behaviorist psychology, RL is concerned with how software agents ought to take actions in an environment in order to maximize their received accumulated rewards. In RL, the acting agent is not explicitly told which actions to implement. Instead, the agent must learn the best action strategy from the observed environment's rewards in response to the agent's actions. Generally, such actions affect both the next reward and subsequent rewards (Sutton and Barto, 1998).

In RL formalism, at each discrete time step $t = 0, 1, 2, \dots$, the agent observes the system's state $\mathbf{s}_t \in S$ and applies an action $\mathbf{a}_t \in \mathcal{A}$, where S is the state space and $\mathcal{A}$ is the action space. Depending on $\mathbf{s}_t$ and $\mathbf{a}_t$, the system transitions to the next state $\mathbf{s}_{t+1}$ and the agent receives a real-value reward $r_{t+1} \in \mathbb{R}$. In deterministic systems the state transition can be expressed as a function $g : S \times \mathcal{A} \to S$ with $g(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$. The related reward is given by a reward function $r : S \times \mathcal{A} \times S \to \mathbb{R}$ with $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) = r_{t+1}$. Hence, the desired solution to an RL problem is a policy that maximizes the expected accumulated rewards.

In our proposed setup, the goal is to find the best policy π among Π the set of all possible equations which can be built from a pre-defined set of function building blocks, with respect to a certain maximum complexity. For every state $\mathbf{s}_t$, the policy outputs an action, i.e., $\pi(\mathbf{s}_t) = \mathbf{a}_t$. The policy's performance, when starting from $\mathbf{s}_t$, is measured by the return $\mathcal{R}(\mathbf{s}_t, \pi)$, i.e., the accumulated future rewards obtained by executing the policy π. To account for increasing uncertainties when accumulating future rewards, the reward r_{t+k} for k future time steps is weighted by γ^k, where $\gamma \in [0, 1]$. Furthermore, adopting a common approach, we include only a finite number of $T > 1$ future rewards in the return (Sutton and Barto, 1998), which is expressed as follows:

$$\mathcal{R}(\mathbf{s}_t, \pi) = \sum_{k=0}^{T-1} \gamma^k r(\mathbf{s}_{t+k}, \pi(\mathbf{s}_{t+k}), \mathbf{s}_{t+k+1}), \quad \text{with} \quad \mathbf{s}_{t+k+1} = g(\mathbf{s}_{t+k}, \mathbf{a}_{t+k}). \tag{1}$$

Herein, we select the discount factor γ such that, at the end of time horizon T, the last reward accounted for is weighted by $q \in [0, 1]$, yielding $\gamma = q^{1/(T-1)}$. The overall state-independent policy performance