Engineering Applications of Artificial Intelligence 65 (2017) 87–98

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies

Daniel Hein [a,b,*], Alexander Hentschel [c], Thomas Runkler [b], Steffen Udluft [b]

[a] Technische Universität Müchen, Department of Informatics, Boltzmannstr. 3, 85748 Garching, Germany
[b] Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, 81739 Munich, Germany
[c] AxiomZen, 980 Howe St #350, Vancouver, BC V6Z 1N9, Canada

A R T I C L E I N F O

Keywords:
Interpretable
Reinforcement learning
Fuzzy policy
Fuzzy controller
Particle swarm optimization

A B S T R A C T

Fuzzy controllers are efficient and interpretable system controllers for continuous state and action spaces. To date, such controllers have been constructed manually or trained automatically either using expert-generated problem-specific cost functions or incorporating detailed knowledge about the optimal control strategy. Both requirements for automatic training processes are not found in most real-world reinforcement learning (RL) problems. In such applications, online learning is often prohibited for safety reasons because it requires exploration of the problem's dynamics during policy training. We introduce a fuzzy particle swarm reinforcement learning (FPSRL) approach that can construct fuzzy RL policies solely by training parameters on world models that simulate real system dynamics. These world models are created by employing an autonomous machine learning technique that uses previously generated transition samples of a real system. To the best of our knowledge, this approach is the first to relate self-organizing fuzzy controllers to model-based batch RL. FPSRL is intended to solve problems in domains where online learning is prohibited, system dynamics are relatively easy to model from previously generated default policy transition samples, and it is expected that a relatively easily interpretable control policy exists. The efficiency of the proposed approach with problems from such domains is demonstrated using three standard RL benchmarks, i.e., mountain car, cart-pole balancing, and cart-pole swing-up. Our experimental results demonstrate high-performing, interpretable fuzzy policies.

1. Introduction

This work is motivated by typical industrial application scenarios. Complex industrial plants, like wind or gas turbines, have already been operated in the field for years. For these plants, low-level control is realized by dedicated expert-designed controllers, which guarantee safety and stability. Such low-level controllers are constructed with respect to the plant's subsystem dependencies which can be modeled by expert knowledge and complex mathematical abstractions, such as first principle models and finite element methods. Examples for low-level controllers include self-organizing fuzzy controllers, which are considered to be efficient and interpretable (Casillas et al., 2003) system controllers in control theory for decades (Procyk and Mamdani, 1979; Scharf and Mandve, 1985; Shao, 1988).

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 dedicated mathematical representations is a complicated and often infeasible approach. Further, modeling such representations by closed-form differentiable equations, as required by classical controller design, is even more complicated. Since in many real-world applications such equations cannot be found, training high-level controllers has to be performed on reward samples from the plant. Reinforcement learning (RL) (Sutton and Barto, 1998) is capable of yielding high-level controllers based solely on available system data.

Generally, RL is concerned with optimization of a policy for a system that can be modeled as a Markov decision process. 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 2). Learning such RL controllers in a way that produces interpretable high-level controllers

* Corresponding author.
E-mail address: daniel.hein@in.tum.de (D. Hein).

http://dx.doi.org/10.1016/j.engappai.2017.07.005
Received 20 October 2016; Received in revised form 4 July 2017; Accepted 7 July 2017
0952-1976/© 2017 Elsevier Ltd. All rights reserved.

is the scope of this paper and the proposed approach. Especially for real-world industry problems this is of high interest, since interpretable RL policies are expected to yield higher acceptance from domain experts than black-box solutions (Maes et al., 2012).

A fundamental difference between classical control theory and machine learning approaches, such as RL, lies in the way how these techniques address stability and reward function design. In classical control theory, stability is the central property of a closed-loop controller. For example, Lyapunov stability theory analyzes the stability of a solution near a point of equilibrium. It is widely used to design controllers for nonlinear systems (Lam and Zhou, 2007). Moreover, fault detection and robustness are of interest for fuzzy systems (Yang et al., 2013, 2014a,b). The problems addressed by classical fuzzy control theory, i.e., stability, fault detection, and robustness, make them well suited for serving as low-level system controllers. For such controllers, reward functions specifically designed for the purpose of parameter training are essential.

In contrast, the second view on defining reward functions, which is typically applied in high-level system control, is to sample from a system's latent underlying reward dynamic and subsequently use this data to perform machine learning. Herein, we apply this second view on defining reward functions, because RL is capable of utilizing sampled reward data for controller training. Note that the goal of RL is to find a policy that maximizes the trajectory's expected accumulated rewards, referred to as return value, without explicitly considering stability.

Several approaches for autonomous training of fuzzy controllers have proven to produce remarkable results on a wide range of problems. Jang (1993) introduced ANFIS, a fuzzy inference system implemented using an adaptive network framework. This approach has been frequently applied to develop fuzzy controllers. For example, ANFIS has been successfully applied to the cart-pole (CP) balancing problem (Saifizul et al., 2006; Hanafy, 2011; Kharola and Gupta, 2014). During the ANFIS training process, training data must represent the desired controller behavior, which makes this process a supervised machine learning approach. However, the optimal controller trajectories are unknown in many industry applications.

Feng (2005a, b) applied particle swarm optimization (PSO) to generate fuzzy systems to balance the CP system and approximate a nonlinear function (Debnath et al., 2013). optimized Gaussian membership function parameters for nonlinear problems and showed that parameter tuning is much easier with PSO than with conventional methods because knowledge about the derivative and complex mathematical equations are not required (Debnath et al., 2013). Kothandaraman and Ponnusamy (2012) applied PSO to tune adaptive neuro fuzzy controllers for a vehicle suspension system. However, similar to ANFIS, the PSO fitness functions in all these contributions have been dedicated expert formulas or mean-square error functions that depend on correctly classified samples.

To the best of our knowledge, self-organizing fuzzy rules have never been combined with a model-based batch RL approach. In the proposed fuzzy particle swarm reinforcement learning (FPSRL) approach, different fuzzy policy parameterizations are evaluated by testing the resulting policy on a world model using a 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 by the optimizer.

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 is trained using historic operational data comprising a set of four-tuples of the form (*observation*, *action*, *reward*, *next observation*), which is 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; Schneegass et 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.

FPSRL is a model-based 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 fuzzy policy offline using this model has several advantages. (1) In many real-world scenarios, data describing system dynamics is available in advance or is easily collected. (2) Policies are not evaluated on the real system, thereby avoiding the detrimental effects of executing a bad policy. (3) 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 using supervised machine learning.

The remainder of this paper is organized as follows. The methods employed in our framework are reviewed in Sections 2–4. Specifically, the problem of finding policies via RL is formalized as an optimization task. In addition, we review Gaussian-shaped membership functions and describe the proposed parameterization approach. Finally, PSO, an optimization heuristic we use for searching for optimal policy parameters, and its different extensions are presented. An overview of how the proposed FPSRL approach is derived from different methods is given in Section 5.

Experiments using three benchmark problems, i.e., the mountain car (MC) problem, the CP balancing (CPB) task, and the more complex CP swing-up (CPSU) challenge, are described in Section 6. In this section, we also explain the setup process of the world models and introduce the applied fuzzy policies.

Experimental results are discussed in Section 7. The results demonstrate that the proposed FPSRL approach can solve the benchmark problems and is human-readable and understandable. To benchmark FPSRL, we compare the obtained results to those of neural fitted Q iteration (NFQ) (Riedmiller, 2005a,b), an established RL technique. Note that this technique was chosen to describe the advantages and limitations of the proposed method compared to a well-known, widely available standard algorithm.

2. Model-based reinforcement learning

In biological learning, an animal interacts with its environment and attempts to find action strategies to maximize its perceived accumulated reward. This notion is formalized in RL, an area of machine learning where 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 responses to the agent's actions. For the most common (and most challenging) RL problems, an action affects both the next reward and subsequent rewards (Sutton and Barto, 1998). Examples for such delayed effects are nonlinear change in position when a force is applied to a body with mass or delayed heating in a combustion engine.

In the RL formalism, the agent interacts with the target system in discrete time steps, $t = 0, 1, 2, \ldots$. At each time step, the agent observes the system's state $s_t \in \mathcal{S}$ and applies an action $a_t \in \mathcal{A}$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. Depending on s_t and a_t, the system transitions to a new state and the agent receives a real-value reward $r_{t+1} \in \mathbb{R}$. Herein, we focus on deterministic systems where state transition g and reward r can be expressed as functions $g : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ with $g(s_t, a_t) = s_{t+1}$ and $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ with $r(s_t, a_t, s_{t+1}) = r_{t+1}$, respectively. The desired solution to an RL problem is an action strategy, i.e., a policy, that maximizes the expected cumulative reward, i.e., return $\mathcal{R}$.

In our proposed setup, the goal is to find the best policy among a set of policies that is spanned by a parameter vector $x \in \mathcal{X}$.