Knowledge-Based Systems 100 (2016) 200–211

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Online fitted policy iteration based on extreme learning machines

Pablo Escandell-Montero[a,*], Delia Lorente[b], José M. Martínez-Martínez[a], Emilio Soria-Olivas[a], Joan Vila-Francés[a], José D. Martín-Guerrero[a]

[a] Intelligent Data Analysis Laboratory (IDAL), Electronics Engineering Department, University of Valencia, Av. de la Universitat s/n, Burjassot, 46100 Valencia, Spain

[b] Centro de Agroingeniería, Instituto Valenciano de Investigaciones Agrarias (IVIA), Crta. Moncada-Náquera km 5, Moncada, 46113 Valencia, Spain

ARTICLE INFO

Article history:
Received 2 July 2015
Revised 29 December 2015
Accepted 8 March 2016
Available online 14 March 2016

Keywords:
Reinforcement learning
Sequential decision-making
Fitted policy iteration
Extreme learning machine

ABSTRACT

Reinforcement learning (RL) is a learning paradigm that can be useful in a wide variety of real-world applications. However, its applicability to complex problems remains problematic due to different causes. Particularly important among these are the high quantity of data required by the agent to learn useful policies and the poor scalability to high-dimensional problems due to the use of local approximators. This paper presents a novel RL algorithm, called online fitted policy iteration (OFPI), that steps forward in both directions. OFPI is based on a semi-batch scheme that increases the convergence speed by reusing data and enables the use of global approximators by reformulating the value function approximation as a standard supervised problem. The proposed method has been empirically evaluated in three benchmark problems. During the experiments, OFPI has employed a neural network trained with the extreme learning machine algorithm to approximate the value functions. Results have demonstrated the stability of OFPI using a global function approximator and also performance improvements over two baseline algorithms (SARSA and Q-learning) combined with eligibility traces and a radial basis function network.

1. Introduction

Reinforcement learning (RL) is a learning paradigm in the field of machine learning for solving decision-making problems where decisions are made in stages [1]. This kind of problems appears in many fields, such as medicine [2,3], automatic control [4,5], artificial intelligence [6,7], or operations research [8,9]. The standard RL setting consists of an agent (or controller) in an environment (or system). Each decision (also called action) produces an immediate reward. The agent learns to perform actions in order to maximize the reward accrued over time. The goal is defined by the user through the reward function. Contrary to other approaches, RL does not require a mathematical model of the system, but it is based on experience (or data). The agent acquires experience interacting with the environment. Fig. 1 represents the main RL elements and how they interact. At each stage or discrete time-point k, the agent receives the environment's state, and on that basis it selects an action. As a consequence of its action, in the next time step, the agent receives a numerical reward and the environment evolves to a new state. The agent selects actions depending on the environment state using a policy that assigns an action to every state. Typically, the agent modifies the policy as a result of the interactions with the environment. The objective of the agent is to find an optimal policy.

Many RL algorithms rely on *value functions* to find optimal policies. Given a policy, the value function estimates the long-term reward obtained by the agent when it follows that policy. Classical RL methods are limited to discrete, small problems because they require exact representations of value functions. However, in most realistic problems the environment state space is large or infinite (e.g., if state variables are continuous). In such cases, value functions must be approximated.

Function approximators can be classified according to their generalization capabilities as global or local. In global approximators (e.g. multilayer perceptron or support vector machines) a change in the parameters induced by an update in a certain part of the input space might influence the values of any region of the output space. On the contrary, a change in the input space of a local approximator (e.g., radial basis network (RBF) or k-nearest-neighbor) only affects to a localized region of the output space [10]. Although global approximators a priori may have very positive effects in combination with RL algorithms [11], they usually lead to poor results even in very simple cases compared to local approximators [12,13]. This is due to the fact that, when the data acquired during the last

* Corresponding author. Tel.: +34 963543421; fax: +34 963544353.
E-mail address: pablo.escandell@uv.es (P. Escandell-Montero).

http://dx.doi.org/10.1016/j.knosys.2016.03.007
0950-7051/

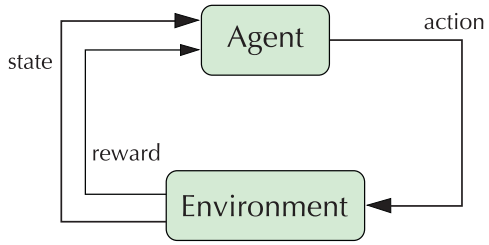

Fig. 1. Basic elements of RL and their flow of interaction.

agent-environment interaction are used to update the parameters of the approximator, they are adjusted to learn the value function at one particular region. However, unpredictable changes may also occur at other regions of the value function [11].

A possible solution to tackle the limitation of global approximators is to accumulate a set of experiences large enough to be representative of the complete state-space and update the entire value function at the same time. In this sense, Ernst et al. [14], based on previous works by Gordon [15] and Ormoneit and Sen [16], proposed a new reinforcement learning scheme where the determination of the value function was reformulated as a sequence of standard *batch-mode* supervised learning problems. Their algorithm, called fitted Q iteration (FQI), works *offline* and is based on *value iteration*, this being one of the two main classes of RL algorithms. The other central class of algorithms is *policy iteration*. Both classes of algorithms are topics of current research and widely used. Nevertheless, policy iteration typically converges using fewer iterations despite it is computationally more demanding than value iteration [1,17,18]. This paper extends the main ideas of FQI to an online, policy iteration algorithm, which can be reliably combined with global approximators and speeds up learning by reusing the acquired data.

In contrast to FQI, the proposed algorithm works online using a semi-batch approach. Therefore, the function approximator should be fast enough to fulfill the time restrictions imposed by online learning. In this context, during the experiments carried out to evaluate the proposed method, a neural network trained with the extreme learning machine (ELM) algorithm was employed to approximate the value functions. ELM is a method for training single-hidden layer feedforward neural networks (SLFNs) that provides an extremely fast learning speed [19].

The rest of this paper is organized as follows. Section 2 presents the required background in Markov decision processes and reinforcement learning. Section 3 introduces briefly extreme learning machine. The details of the proposed algorithm are described in Section 4. Section 5 discusses the relation of the proposed algorithm with similar methods. In Sections 6 and 7, the experiments and results are presented, respectively. Finally, the conclusions are drawn in Section 8.

2. Reinforcement learning

The reinforcement learning problem can be formalized using the Markov decision processes (MDPs) [20] framework. Firstly, this section introduces the RL problem using MDPs. Then, the classical algorithm policy iteration is described briefly.

2.1. Markov decision processes

An MDP is defined by the tuple $\{S, A, P, \rho\}$, where S is the state space of the process, A is the action space, P the transition probability function P: $S \times A \times S \to [0, 1)$ that gives the probability of the next state as a result of the chosen action, and $\rho : S \times A \times S \to \mathbb{R}$ is the bounded reward function [20]. Here, S models the possible states of the environment and A includes the actions that can be performed by the agent. Let k denote the current stage or time step. Once the action a_k is applied to the state s_k, the next state s_{k+1} is determined by the transition probability function P. The agent selects actions according to its policy $\pi : S \to A$ which drives the action selection process. Each transition generates an immediate reward $r_{k+1} = \rho(s_k, a_k, s_{k+1})$. The reward evaluates the immediate effect of the transition, but it does not provide information about its long-term effects [18].

The goal of the agent is to learn a policy that maximizes the return (e.g. the sum of rewards received over time). Such a maximization policy, denoted by π^*, is said to be optimal. For an initial state s_0 and under the policy π, the expected infinite-horizon discounted return[1] is [21]:

$$R^{\pi}(s_0) = \lim_{K \to \infty} E_{s_{k+1}|s_k, \pi(s_k)} \left\{ \sum_{k=0}^{K} \gamma^k \rho(s_k, \pi(s_k), s_{k+1}) \right\} \tag{1}$$

where $\gamma \in [0, 1)$ is the discount factor. This parameter can be intuitively interpreted as a way to balance the immediate reward and future rewards. Future rewards are more relevant for the calculation of the return when γ approaches 1.

The quality of a policy π can be measured using its state-action value function $Q^{\pi} : S \times A \to \mathbb{R}$ (commonly referred to as Q-function). The Q-function is defined as the total expected discounted reward that is encountered starting from state s, taking action a and thereafter following policy π [18]:

$$Q^{\pi}(s, a) = E_{s'|s,a} \left\{ \rho(s, a, s') + \gamma R^{\pi}(s') \right\} \tag{2}$$

where s' is the state reached after taking action a in the state s.

Due to the Markov property, the Q-function of a policy π satisfies the Bellman equation [18]:

$$Q^{\pi}(s, a) = E_{s'|s,a} \left\{ \rho(s, a, s') + \gamma Q^{\pi}(s', a') \right\} \tag{3}$$

The optimal Q-function is defined as $Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$, i.e., the best Q-function that can be obtained from any policy. From the optimal Q-function, an optimal policy can be easily derived choosing in each state the action that maximizes Q^*:

$$\pi^*(s) = \arg\max_{a} Q^*(s, a) \tag{4}$$

In general, for a given Q-function, a policy that maximizes Q in this way is said to be greedy in Q. Therefore, a given MDP can be solved (i.e. finding an optimal policy) by first finding Q^*, and then using Eq. (4) to compute a greedy policy in Q^*. This methods are collectively known as *value function methods*.

2.2. Policy iteration

Policy iteration (PI) is one of the major approaches used in RL to solve MDPs [23]. This section introduces some relevant theoretical results from the classical policy iteration algorithm, which will be used in Section 4 to derive the proposed algorithm.

Policy iteration consists in using an iterative process to construct a sequence of policies that are monotonically improved. Starting with an arbitrary policy π_0, at iteration $l > 0$, the algorithm computes the Q-function underlaying π_l (this step is called policy evaluation). Then, given Q^{π_l}, a new policy π_{l+1} that is greedy with respect to Q^{π_l} is found (this step is called policy improvement). Fig. 2 depicts schematically the policy iteration algorithm [24]. When the state-action space is finite and exact representations of the value function and policy are used (usually

[1] Although there are several types of returns, this work only focuses on infinite-horizon discounted return due to its useful theoretical properties. For a discussion of these properties and other types of returns, see [21,22].