STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 2, 2011

REINFORCEMENT LEARNING ALGORITHMS IN ROBOTICS

BOTOND BÓCSI[(1)] AND LEHEL CSATÓ[(1)]

Abstract. Modern robots are not build to solve pre-determined tasks, rather they are designed to tackle a wider class of problems. Finding efficient control algorithms for a new problem within the class is not straightforward. Machine learning techniques, e.g., reinforcement learning (RL) proved to provide suitable methods in finding such control algorithms. Robotic control learning tasks share several common properties, thus, when selecting among RL methods one has to consider these properties. In this paper, we present the state-of-the-art RL algorithms from the perspective of robotic control. We highlight their advantages and drawbacks in conjunction with robotic control, hereby, analyzing their feasibility in this context. Our results are supported by simulated pole balancing control experiments.

1. Introduction

The aim of machine learning (ML) is to develop algorithms that improve their performance based on empirical observed data. We aim to use machine learning techniques in developing *intelligent* robots. Intelligent robots are defined in this context as instruments – or algorithms – that can adapt to new environments and new conditions *whilst* solving the problem they were designed for. Within the context of implementing adaptive behavior, a promising ML technique is the application of the *reinforcement learning* methods. Requiring limited knowledge about the environment, these methods are used with success in problems like optimal robot control [6], or various tasks involving unknown environments where agents must move.

Within the problems addressed by RL, robotic control learning tasks share several common properties, thus, when selecting among RL methods one has to consider these properties. For example, variables attached with the robotic

Received by the editors: March 14, 2011.

2010 *Mathematics Subject Classification.* 68T05, 68T40, 60J25.

1998 *CR Categories and Descriptors.* I.2.9 [**Artificial intelligence**]: Robotics – *Autonomous vehicles*; I.2.6 [**Artificial intelligence**]: Learning – *Kowledge aquisition.*

Key words and phrases. machine learning, reinforcement learning, robotics, policy gradient.

learning process – e.g., joint angles, joint torques – are continuous, therefore, methods which handle well continuous state spaces are preferred. Another requirement is the efficient handling of high dimensional data originated from robots with many degrees of freedom – e.g., humanoid robots. These features require a special selection of learning algorithms. In this paper, we analyze the state-of-the-art RL algorithms from the perspective of robotic control. Several attempts has been made to takle this problem [6, 1, 3], we present a comparative study of RL methods in conjunction with robotic control, analyzing their feasibility in this context.

The paper is organized as follows. In Section 2, we define RL and introduce the state-of-the-art RL algorithms, i.e., value based methods, policy gradient methods, and evolutionary algorithms. These algorithms form the base of our comparison. In Section 3, we presents a quantitative comparison of the presented methods based on a simulated pole balancing experiment. Conclusions are drawn is Section 4.

2. Reinforcement Learning Algorithms

Reinforcement learning (RL) is the learning process when an *agent* takes *actions* in an *environment* consisting of *states* and gets *reward* associated to state and action pairs [9]. The goal of the agent is to maximize its long-term reward. Formally RL problems are defined in terms of a *Markov Decision Process* (MDP) [7] consisting of a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \pi)$ where (1) $\mathcal{S}$ is the state space; (2) $\mathcal{A}$ is the action space; (3) $\mathcal{P}^a_{ss'} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \Re$, with $\mathcal{P}^a_{ss'} = P(s'|s,a)$ are the transition probabilities, i.e., the probability of going from state s to s' by taking action a; (4) $\mathcal{R}^a_{ss'} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \Re$ is the reward received when action a was taken in state s followed by state s'; (5) $\pi(s,a) : \mathcal{S} \times \mathcal{A} \to [0,1]$, $\pi(s,a) = P(a|s)$ is called the policy, that is the probability of taking action a in state s. A trajectory – a.k.a episode or roll-out – τ is a sequence of triplets $(s_t, a_t, r_t) \in \mathcal{S} \times \mathcal{A} \times \mathcal{R}$ with t the time index. The values of the triplets a_{t+1}, s_{t+1}, r_{t+1} are obtained from sampling based on the policy $\pi(s,a)$ and the transition probabilities.

By solving an MDP, we understand the finding of a policy π' that maximizes the long-term (discounted) reward along a trajectory generated by the respective policy

$$\pi' = \arg\max_{\pi} \; E_\pi \left[\sum_t \gamma^t r_t \right],$$

where $r_t \in \tau$, E_π denotes expectation conditioned on π, and $\gamma \in (0,1]$ is a discount factor. The objective of RL is to solve the MDP underlying the problem. Solving the MDP is not straightforward. To tackle the problem, different algorithms has been introduced, approaching the problem from different points of views. Next, we classify the learning approaches into three classes and present the appropriate methods in details.

2.1. **Value function based methods.** Value function based methods model the optimal policy indirectly via so called value functions. The key insight is that we measure the utility of states and actions respectively, and based on these values, the optimal policy chooses the action that has to higher utility.

Given an MDP, we define the *action-value function*, also known as Q *function*, that expresses the utility of action a in state s. The definition looks as follows[1]

$$Q(s_t, a_t) = \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q(s', a') \right].$$

The optimal action-value function Q is a fixed-point of the above equality. Then, the optimal policy can be easily computed by taking the most valuable action in every state, i.e., $\pi(s, a) \sim \arg\max_a Q(s, a)$.

The computation of Q is not trivial, different approaches have been proposed. A fundamental method is temporal-difference learning [9]. It is an iterative algorithm that updates the values of Q after every action taken based on the difference between expected and observed Q value. Q-learning [9] is the mostly used temporal-difference learning methods. It has the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right],$$

where $\alpha \in [0, 1]$ is a learning rate that expresses how much confidence we have in the observed value.

Is has been shown that temporal-difference learning converges asymptotically to the optimal policy when the accurate representation of the action value function is possible [9]. However, when the action-state space is continuous – e.g., joint angles, joint torques, in the case of robotic control – function approximation has to be applied to model the value function. As a consequence, all theoretical convergence guarantees are vanished [4]. When used in continuous domain, a tabular representation of the value function is advised. This representation is unfeasible in high dimensional action-state spaces.

2.2. **Policy gradient methods.** A different approach for solving the RL problem is to model the policy directly as a parametric function π_θ, e.g., by

[1]The definition of the *value function* $V(s) = \max_a Q(s, a)$ is also possible, however, the determination of the policy is harder based on $V(s)$.

a neural network, and update its parameters using steepest gradient descent [8], based on the gradient of the average expected reward:

$$J(\theta) = E_{\pi_\theta}\left[\sum_t \gamma^t r_t\right].$$

The computation of the gradient of $J(\theta)$ is not tractable, thus, to ease the difficulties related to calculating the expected return, several approximations of the gradient have been suggested. We present the three basic approaches.

Finite difference methods compute the gradient by making small perturbation in the parameters of the policy and observing the corresponding rewards. Then, the gradient is estimated using regression techniques. The generation of the parameter perturbation is difficult, since it depends on the parameter space induced by the policy. These methods suffer from slow convergence. For details about finite difference methods consult Peters and Schaal 2008 [6].

The family of *vanilla policy gradient* algorithms use the *log-ratio method* to compute the gradient [6]. They have several advantages over finite difference methods. Fewer roll-outs are needed to achieve convergence – it is possible that a single roll-out leads to an unbiased gradient estimate. Another benefit is that perturbation – representing the exploration – is not generated in the parameter space, rather in the action space that is much easier to handle.

To speed up the vanilla policy gradient algorithm, the use of *natural gradients* [6] has been suggested. The motivation behind natural policy gradient is that the first order gradient based policy update step does not take into account the structure of the parameter space. Kakade 2001 [2] introduced the extension by defining a metric based on the underlying parameter space.

Policy gradient methods can be used with different policy representations, thus, the policy can be chosen to handle well continuous state spaces and to scale acceptable with high dimensional data, as well. The major drawback of the policy gradient methods is that they can easily be stuck in local maxima. This is a direct cause of the steepest gradient descent learning [8].

2.3. **Evolutionary methods.** Evolutionary methods are black-box optimization algorithms. They optimize a parametric function by keeping a population of possible function parameters – called individuals –, and combining them based on the corresponding function values. In RL, individuals are policies and the function values are the corresponding average expected rewards [5]. Evolutionary algorithms are used with success in RL [1, 3], since they need no prior knowledge about the learning task.

As well as policy gradient methods, evolutionary algorithms model directly the policy, thus, share the advantage of good scalability to high dimensional and continuous state spaces. Although, note that high dimensional tasks may require a large population size. Evolutionary methods are less influenced by being stuck in local maxima solutions but they need significantly more evaluations to converge – see Section 3.

3. Experiments

In this section, we present the simulated robotic experiments we conducted, and highlight the advantages and drawbacks of the presented learning methods in a robotic control framework. We have conducted experiments to analyze the performance of the following algorithms: Q-learning, finite difference method, vanilla policy gradient, natural policy gradient, and evolutionary algorithms.

The experiments were conducted in a simulated 3D environment – using the ODE physics simulation library – on a pole balancing robot[2] – see Figure 1.(a). The task of the robot – a car with a pole attached on the top – is to learn how to prevent the pole from falling down by applying force to itself. For evolutionary methods and policy gradient learning we used neural networks with no hidden layer as policy. As a measure of performance we used the average number of episodes needed by the algorithms to find a good policy.

Results based on 393 experiments are shown on Figure 1.(b) where the variance of the convergence is displayed as well. From the simulations reveals that the policy gradient based methods outperformed the other algorithms. The finite difference method is rather slow and the time of convergence has a huge variance. The vanilla policy gradient algorithm produced better results than the natural policy gradient, however, it failed to converge in 10% of the simulations which did not happen in the case of natural policy gradient. Divergence occurred when the robot was close to the optimal policy and the magnitude of the gradient was too small, therefore, the update of the parameters had almost no effect. Q-learning happened to be stable but produced the worst results since it does not scale well with high dimensional continuous state spaces.

Note that all the algorithms are sensitive to parameter settings (e.g., state space segmentation – Q-learning – , convergence detection – gradient based methods –, population size – evolutionary algorithms), thus, careful parameter tuning is required to obtain good performance.

4. Conclusions

In this paper, we analyzed RL algorithms from the point of view of robotic control. We have shown that algorithms which use direct policy modeling provide better performance than value based methods. This behavior is a

[2]Code available at `http://cs.ubbcluj.ro/~bboti/downloads/RL_sim.tar.gz`.

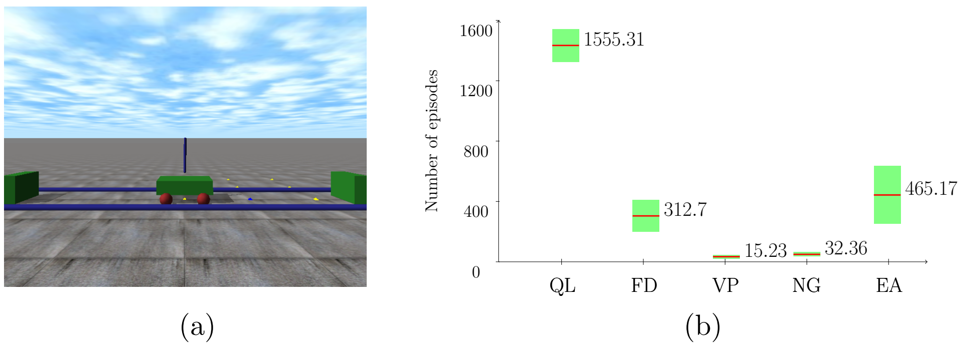

FIGURE 1. (a) Simulated pole balancing robot. (b) Experimental results, performance measured by the number of episodes until convergence, for Q-learning (QL), finite difference method (FD), vanilla policy gradient method (VP), natural gradient method (NG) and evolutionary algorithm (EA).

direct cause of the continuous action-state spaces induced by robotic control tasks. The results are based on both theoretical considerations and simulated robotic control experiments. As future work, we aim to improve the value based methods by finding suitable value function approximators – e.g., Gaussian processes. We also want to provide theoretical convergence guarantees when the aforementioned approximations are used.

ACKNOWLEDGMENTS

The authors wish to thank for the financial support provided from program: Investing in people! PhD scholarship, project co-financed by the European Social Fund, sectoral operational program, human resources development 2007 - 2013: POSDRU 88/1.5/S/60185 – "Innovative doctoral studies in a knowledge based society".

REFERENCES

1. F. Gomez, J. Schmidhuber, and R. Miikkulainen, *Accelerated neural evolution through cooperatively coevolved synapses*, Journal of Machine Learning Research **9** (2009), 937–965.
2. S. Kakade, *A natural policy gradient*, Advances in Neural Information Processing Systems (NIPS), 2001, pp. 1531–1538.
3. J. R. Koza and J. P. Rice, *Automatic programming of robots using genetic programming*, Proceedings of the Tenth National Conference on Artificial Intelligence, The MIT Press, 1992, pp. 194–201.
4. F. S. Melo and M. I. Ribeiro, *Q-learning with linear function approximation*, Proceedings of the 20th Annual Conference on Learning Theory, Springer-Verlag, 2007, pp. 308–322.

5. D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, *Evolutionary algorithms for reinforcement learning*, Journal of Artificial Intelligence Research **11** (1999), 241–276.
6. J. Peters and S. Schaal, *Reinforcement learning of motor skills with policy gradients*, Neural Networks **21** (2008), no. 4, 682–697.
7. M. L. Puterman, *Markov decision processes: Discrete stochastic dynamic programming*, Wiley-Interscience, April 1994.
8. J. A. Snyman, *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*, Applied Optimization, Vol. 97, Springer-Verlag New York, Inc., 2005.
9. R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, The MIT Press, March 1998.

(1) Babeş-Bolyai University, Faculty of Mathematics and Computer Science, 1 Kogalniceanu str., RO-400084 Cluj-Napoca, Romania

E-mail address: {bboti, lehel.csato}@cs.ubbcluj.ro