in: Proc. Int. Conf. on Artifical Neural Networks (ICANN) 2003, Istanbul, pp. 346-349, Bogaziai University Press

Is Reinforcement-Learning Able to Solve Real-World Challenges?

V. Stephan, M. Saupe, M. Bischoff, H. Reindanz, H.-M. Gross.
Ilmenau Technical University, Department of Neuroinformatics, P.O.B. 100565,
D-98684 Ilmenau, Germany
{vstephan,homi}@informatik.tu-ilmenau.de

Abstract

In this paper we compare several reinforcement-learning (RL) approaches with respect to their ability to solve problems arising in real-world. Thereto, we investigated various RL-agents, ranging from classical value-iteration like Q-learning to policy-iteration techniques like ADHDP. These agents try to solve several tasks within a simulated environment featuring properties, that usually characterize real-world problems like continuous state and action space, partial observability, changing goals or dead-times.

1 Introduction

There have already been a number of benchmarks in context of reinforcement learning, such as [1] and [2]. But these tests did not deal with real-world problems or were only focused on one special aspect arising in real-world. Therefore we have decided to compare various methods of reinforcement learning using a simple test environment. The tests are designed with respect to following problems of real-world: partially observable system properties, time changing goals, dead-time between control variables and observed inputs. In a separate study, R-learning and different Q-learning were investigated [3]. In this paper, we want to continue this work with emphasis to other real-world specific aspects and extend the tests to approaches based on Growing Neural Gas [4] and ADHDP [5].

2 Benchmark-Scenario

To investigate the performance of the presented RL-architectures in the framework of real-world control problems, we decided not to conduct experiments with a real problem, because of two major drawbacks: First, a benchmark usually requires many experiments, which would be very time consuming on a real platform or process and second, it is almost impossible to guarantee identical system properties for every single experiment in order to make the results comparable.

Because of this, we use a typical and well-known control problem, which was extended by the interesting real-world properties. The environment consists of a ball rolling in a mountain range, which is defined by the polynom $y = 32x^4 - 64x^3 + 42x^2 - 10x + 0.95$ with $x \in [0 \dots 1]$ (see figure 1). The dynamics of the

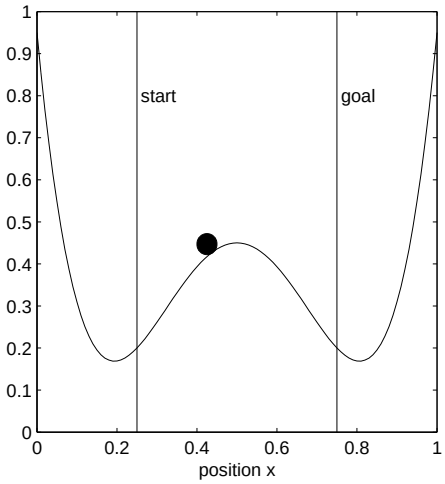

Figure 1: Simulation of a ball within a mountain range.

ball is described by equation 1, where $\ddot{x}$ denotes the acceleration, $\dot{x}$ the velocity, and x the position of the ball.

$$m\ddot{x} = f_A + f_N + f_C \quad (1)$$

$$f_A = -mg\sin(\alpha(x)) \quad (2)$$

The tangent force f_A, the unobservable disturbance f_N, and the force f_C, generated by the RL-agent to control the ball, define the acceleration $\ddot{x}$. $m = 1000g$ denotes the mass of the ball, $g = 9.81m/s^2$ the gravity and $\alpha(x)$ the steepness of the ground at position x.

For all the following experiments, the RL-agent receives information about the current position x and velocity $\dot{x}$ of the ball and generates actions $f_C \in [-4N \ldots +4N]$ to move the ball from its initial position $x_0 = 0.25$ to the target position $x^T = 0.75$. The reward is defined as $r_t = -|x_t - x^T|$.

3 RL-Agents

Standard Q-Learning:
Classical Q-learning [6] uses a predefined discrete set of states, a predefined discrete set of actions, and approximates via action value iteration a value for each state-action pair solving the Bellman-equation (eqn. 3).

$$\Delta Q(s_t, a_t) \quad = \quad r_t + \gamma \cdot max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \quad (3)$$

$Q(s, a)$ is the value of action a executed in state s, r_t denotes the received reward, and γ is a discount factor.

Q-Learning with Clustered State Space: A negative aspect of the standard Q-learning is its predefined set of states. For a more efficient state-representation a neural cluster called Neural Gas [7] is applied, which places the state-representing weights according to the real distribution of the systems states. By concentrating to these relevant regions of the systems state space, the limited state-resources can be used much more efficiently.

Q-Learning with Incrementally Clustered State Space: Both approaches explained above, use a fixed number of 25 states. However, in a real-world environment, which may change over time, it is difficult to define an appropriate number of states in advance to approximate the value function properly. By using incremental neural networks, like Growing Neural Gas [4], a RL-agent is able to adapt to changing environments by inserting new neurons in regions with high input density.

Action Dependent Heuristic Dynamic Programming: To handle problems with large state space and continuous action space in reinforcement-learning, policy iteration approaches using function approximators like MLP's were developed. In ADHDP [5] a multilayer perceptron (MLP), called actor, is used to implement the policy-mapping $X \mapsto A$ and a second MLP, called critic, learns to approximate the Bellman-equation. The actor is adapted by backpropagation through the critic network in order to maximize the critic output.

Random-Agent For comparison, we build a random-agent, which chooses its actions independently from the current input simply by generating equally distributed actions within the given action space. This random-agent of course is not used to solve the task, but its performance is a valuable measure for the complexity of the given problem and the performance of the RL-agents.

4 Results

To investigate these RL-agents described above with respect to the environment-properties introduced in section 1, we designed various experiments. For every experiment, the RL-agent started without any a-priori knowledge to explore the environment. This exploration was realized by adding a noise term to the generated action, which decreased over time (approx. 500 steps) down to zero. Each experiment consists of 100 trials, where for each trial the ball is initially placed at its starting position and is controlled by the RL-agent's actions for 500 steps. In order to become more robust against different weight initializations of the RL-agents, each experiment is independently repeated 10 times. Thus, every plot shows the mean result over 10 independent realizations of the same experiment.

4.1 Learning with Different Starting Points and Partial Observability

In a very first experiment (*default*), we inserted the ball at the starting position 0.25 depicted in figure 1. The RL-agents had to find out a strategy to move the ball out of the starting valley by swinging up and, after reaching the goal-valley, decelerate the ball to keep the ball as close as possible to the target position $x_T = 0.75$. In a second experiment, we inserted the ball in the target-valley (*goalstart*) in order to investigate, if any a-priori knowledge about the target position may help the RL-agents to develop an appropriate strategy. In a third test, we examined the influence of a partially observable environment to the performance of the RL-agents (*POMDP*). Thereto, in contrast to previous tests, the unobservable force $f_N \in [-2N \ldots 2N]$ with a sinusoidal shape was introduced to simulate an oscillating wind.

As can be seen in figure 2, the standard-Q, the NG+Q, and the GNG+Q approach achieve pretty good results. The lower average performance of the GNG+Q in the *goalstart* experiment is caused by a single unsuccessful run, during which the agent suddenly and completely forgot the strategy he learned. This can be caused by the deletion of a neuron (and the loss of the associated Q-values) through the GNG algorithm. The ADHDP-approach is unable to control the environment (compared to the performance of the RANDOM-agent).

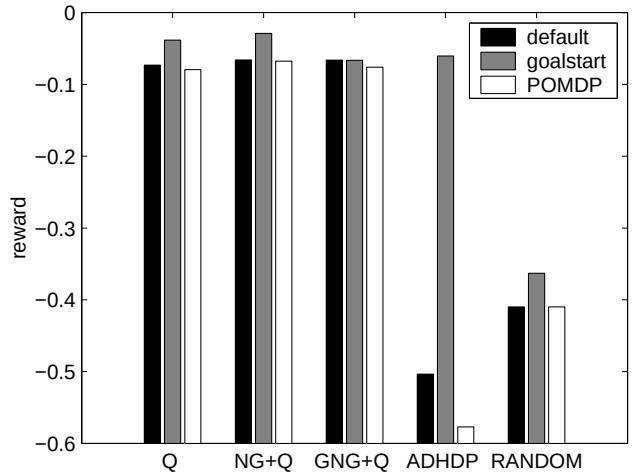

Figure 2: Mean reward of the last trial out of a learning experiment with 100 trials averaged over 10 independent experiments for the RL-agents standard-Q (Q), Q-learning with clustered state space (NG+Q), Q-learning with incrementally clustered state space (GNG+Q), Action Dependent Heuristic Dynamic Programming (ADHDP), and the random-agent (RANDOM) for the experiments *default, goalstart*, and *POMDP*. The higher the mean reward, the closer and faster the agent moved the ball to the target position (see section 2).

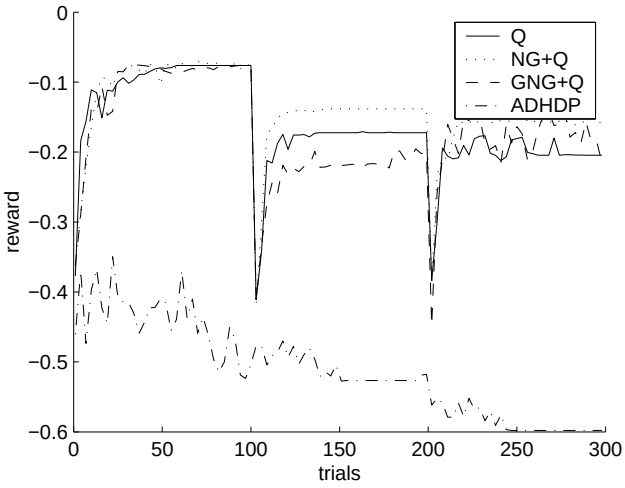

Figure 3: Development of mean reward over 300 trials. At trial 100, the target position was moved to 0.25, and at trial 200, the target was moved back to 0.75.

4.2 Learning with a Changing Goal

Changing system goals are of great importance for many real-world applications. Thereto, we investigated the ability of these RL-agents to perform this task by moving the target position from 0.75 to 0.25 after agent-training and afterwards back to 0.75 during one experiment. Figure 3 depicts clearly, that the well performing agents Q, NG+Q and GNG+Q experience a dramatical performance-reduction after moving the target position. Nevertheless, they are able to adapt to the new situation, where the NG+Q-approach is much more successful even if this plasticity is decreasing over time. Agent ADHDP poorly performs in the first *default* section, so a target change does not influence the initially bad results.

4.3 Learning with Dead-Times

In a real-world environment, usually there is no instantaneous reaction to the controllers action. To simulate this aspect, the generated actions were fed into the environment with a delay of 1 up to 5 time steps. For comparison, a well trained agent takes about 15 steps to move the ball from its starting position to the top of the hill in the middle. Again, the Q, NG+Q, and GNG+Q-agents perform well without dead-times between action generation and execution ($T_D = 0$). As expected, the performance decreases with increasing dead-time (see figure 4). A deadtime $T_D > 0$ prevents the agent from learning a valid mapping $state \times action \mapsto reward$, because the causal relationship between state, action, and reward is disturbed more and more with increasing dead-time. ADHDP again is unable to develop a problem-solving strategy with any dead-time.

4.4 Combination

Finally, we combined some of these real-world challenges investigated separately above in a complex experimental scenario. Thereto, the environment featured a dead-time $T_D = 2$, the unobservable disturbance $f_N \in [-2N \dots 2N]$, and a time changing target position. As to be seen in figure 5, agents Q, NG+Q, and GNG+Q are able to solve the problem in the first phase of the experiment despite of the present dead-time and partial observability. The major problem is the changing target position. The ability of these Agents, to adapt to this change is reduced by the other real-world properties (see also section 4.2). As in previous experiments ADHDP fails to control the environment appropriately.

Finally, it must be stated, that none of the agents investigated in this article, is able to solve the simulated real-world problems sufficiently. As expected, after several resettings of the target position even the best agents perform pretty poor to move the ball to the target position.

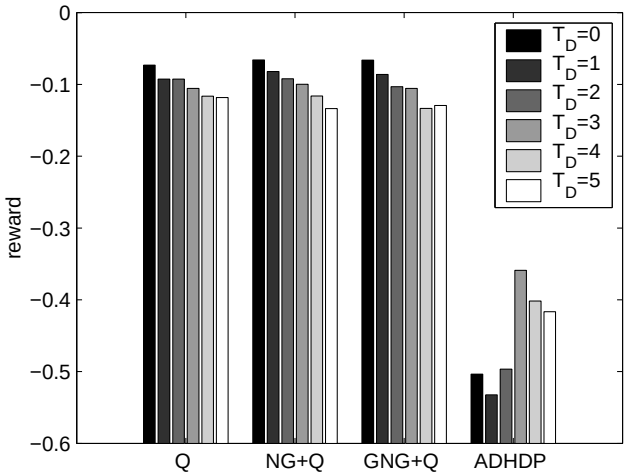

Figure 4: Mean reward of the last trial averaged over 10 independent runs for different dead-times.

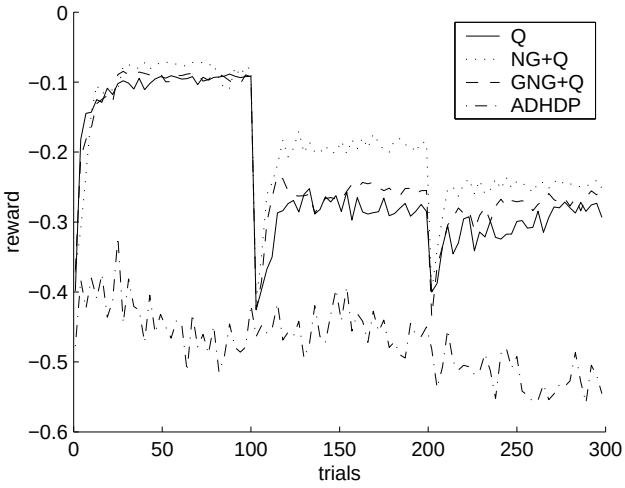

Figure 5: Development of mean reward over 300 trials for the combined scenario including dead-time, partial observability and changing goals. The target position is moved as well as in figure 3.

5 Conclusions and Future Work

Our experiments show that Q, NG+Q and GNG+Q produce the best results; the better performance of the NG+Q-agent is caused by the ability to place neurons according to input density and the fact that not only the best-matching neuron but also it's neighbours learn in each step. The incremental GNG+Q agent is able to remove rarely used neurons and reinsert them where needed. Of course, this agent benefits from that advantage, but also suffers from the fact, that sometimes the deletion of neurons causes loss of vital information that cannot always be relearned. In consequence, the performance of the GNG+Q is varying in a larger range, showing both the best and worst single runs of all successful agents. The ADHDP agent was not able to learn an appropriate control policy in our experiments with the ball-world scenario. The relatively high average reward in the goalstart experiment documents, that that agent benefits from the a-priori knowledge about the target position. As described in [5], the used ADHDP design depends on settings which start close to the desired target position and system dynamics, that produce sample trajectories leading away from the goal into failure states. We conclude that this ADHDP-approach is not applicable for this type of scenario.

Further work will address the stability-plasticity dilemma, which may be solved by an adaptive exploration, which is activated, if the learned values of the state-action-pairs become invalid. Although our scenario tries to simulate many aspects of the "real world" closely, final experiments must use a real environment.

References

[1] S. Mahadevan. To discount or not to discount in reinforcement learning: A case study comparing r-learning and q-learning. In *International Conference on Machine Learning*, pages 164–172, 1994.

[2] D. Surmeli and H.-M. Gross. Benchmarking reinforcement-learning based on neural function approximatros. In *Proc. of Fourth International Conference on Cognitive and Neural Systems*. Boston University Press, 2000.

[3] H. Renkewitz. Untersuchung verschiedener Reinforcement-Lernverfahren auf ihre Realwelttauglichkeit. Master's thesis, Ilmenau, Technical University, 9 2002.

[4] B. Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*. MIT-Press, 1995.

[5] J. Si and Y.-T. Wang. On-line learning control by association and reinforcement. *IEEE Transactions on Neural Networks*, 12:264–276, 2001.

[6] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[7] T.M. Martinetz and K. Schulten. A "neural gas" network learns topologies. In T. Kohonen, Mäkisara, K., O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 397–402. Elsevier Amsterdam, 1991.