An Adaptive Clustering Method for Model-free Reinforcement Learning

Andreas Matt and Georg Regensburger [*]
Institute of Mathematics
University of Innsbruck, Austria
{andreas.matt, georg.regensburger}@uibk.ac.at

Abstract

Machine Learning for real world applications is a complex task due to the huge state and action sets they deal with and the a priori unknown dynamics of the environment involved. Reinforcement Learning offers very efficient model-free methods which are often combined with approximation architectures to overcome these problems. We present a Q-learning implementation that uses a new adaptive clustering method to approximate state and actions sets. Experimental results for an obstacle avoidance behavior with the mobile robot Khepera are given.

1. Introduction

Reinforcement learning (RL) is a research field within Machine Learning and Artificial Intelligence. It addresses problems of sequential decision making and stochastic control and is strongly connected to dynamic programming and Markov decision processes.

RL is based on the idea of learning by trial and error while interacting with an environment. At each step the agent performs an action and receives a reward depending on the starting state, the action and the environment. The agent learns to choose actions that maximize the sum of all rewards in the long run. The resulting choice of actions for each state is called a policy.

Several methods to find optimal or suboptimal policies are known, Bertsekas and Tsitsiklis [1], Kaelbling, Littman and Moore [2], Sutton and Barto [3]. Combined with approximation architectures to deal with huge state and action sets model-free methods have become a powerful tool which resulted in impressive applications, see for example [1] and [3].

* We would like to thank Prof. Ulrich Oberst for numerous discussions and substantial improvements on the presented method.

2. Markov Decision Processes

The term Markov decision process (MDP) was introduced by Bellman [4, 5]. For further information on MDPs see for example Puterman [6].

An *environment* is given by

- A finite set S. The set S is interpreted as the set of all possible *states*.
- A family $\mathbf{A} = (A(s))_{s\in S}$ of finite sets. The set $A(s)$ is interpreted as the set of available *actions* in state s.
- A family $\mathbf{P} = (P(- \mid a, s))_{s\in S, a\in A(s)}$ of probabilities $P(- \mid a, s)$ on S. We interpret $P(s' \mid a, s)$ as the *transition probability* that performing action a in state s leads to the *successor state* s'.

Let $E = (S, \mathbf{A}, \mathbf{P})$ be an environment. A *policy* for E is given by a family $\pi = (\pi(- \mid s))_{s\in S}$ of probabilities $\pi(- \mid s)$ on $A(s)$. We interpret $\pi(a \mid s)$ as the probability that action a is chosen in state s. A policy is called *deterministic* if $\pi(- \mid s)$ is deterministic for all $s \in S$. We use the notation $\pi(- \mid s) = a \in A(s)$.

A *finite Markov decision process (MDP)* is given by an environment $E = (S, \mathbf{A}, \mathbf{P})$ and a family

$$\mathbf{R} = (R(s', a, s))_{s', s\in S, a\in A(s)} \quad \text{with } R(s', a, s) \in \mathbb{R}.$$

The value $R(s', a, s)$ represents the *reward* if performing action a in state s leads to the successor state s'.

The goal of RL is to find policies that maximize the sum of the expected rewards. Let $(E, \mathbf{R})$ be an MDP and $0 \leq \gamma < 1$ be a *discount rate*. The *value function* for a policy π for E is the unique solution of the *Bellman equation*

$$V^\pi(s) = \sum_{a\in A(s)} \left(R(a, s) + \gamma \sum_{s'} V^\pi(s') P(s' \mid a, s) \right) \cdot \pi(a \mid s), \quad \text{for } s \in S,$$

where

$$R(a, s) = \sum\nolimits_{s'} R(s', a, s) P(s' \mid a, s)$$

is the *expected reward* of action a in state s. The value function $V^{\pi}(s)$ is the expected discounted sum of rewards starting in state s and following policy π. The Bellman equation suggests defining the *action-value* of action $a \in A(s)$ in state s for policy π

$$Q^{\pi}(a,s) = R(a,s) + \gamma \sum_{s' \in S} V^{\pi}(s') P(s' \mid a,s)$$

as the average reward if action a is chosen in state s and afterwards the policy π is followed.

A policy π^* is *optimal* if $V^{\pi}(s) \leq V^{\pi^*}(s)$ for all $s \in S$ and all policies π. There exists at least one deterministic optimal policy and all optimal policies share the same value function which we denote by V^*. The *optimal action-value* for action $a \in A(s)$ in state $s \in S$ is defined by

$$Q^*(a,s) = R(a,s) + \gamma \sum_{s' \in S} V^*(s') P(s' \mid a,s). \quad (1)$$

The optimal value function and action-values satisfy

$$V^*(s) = \max_{a \in A(s)} Q^*(a,s), \quad \text{for } s \in S. \quad (2)$$

A *greedy policy* π^* for the optimal action-values Q^*,

$$\pi^*(- \mid s) = a \in \arg\max_{a \in A(s)} Q^*(a,s), \quad \text{for } s \in S,$$

is an optimal deterministic policy. A greedy policy for an approximation of the optimal action-values is an approximation of an optimal policy.

3. Model-free Methods and Q-learning

Methods that approximate the value function and action-values or find (sub)optimal policies without an explicit model of the environment are called *model-free* methods. They are important in real world applications where in general the transition probabilities of the environment can only be observed.

Figure 1 summarizes the interaction between an agent and the (real world) environment or simulation which provide the observations. We start in state s, apply an action a and observe the successor state s' and the reward $r = R(s', a, s) \in \mathbb{R}$. The resulting triple (s', a, s) and the reward r are then used to estimate the value function or action-values.

Q-learning, introduced by Watkins [7], directly approximates the optimal action-values Q^*. The following considerations motivate the algorithm. Let $(E, \mathbf{R}, \gamma)$ be an MDP. By substituting (2) into (1) we obtain

$$Q^*(a,s) = R(a,s) + \gamma \sum_{s'} \max_{a'} Q^*(a', s') P(s' \mid a,s).$$

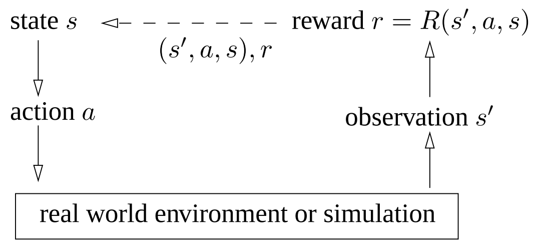

Figure 1. Interaction between agent and environment

Suppose that Q is an approximation of Q^*. Then r is an estimate of $R(a,s)$ and

$$r + \gamma \max_{a' \in A(s')} Q(a', s')$$

is a new estimate of $Q^*(a,s)$ based on the current observations (s', a, s), r and Q. We obtain the *update rule*

$$Q(a,s) \leftarrow Q(a,s) + \alpha(r + \gamma \max_{a'} Q(a', s') - Q(a,s)), \quad (3)$$

where α is a positive *step-size parameter*. The resulting algorithm is given in Algorithm 1.

```
Output: an approximation of $Q^*$
  initialize $Q$ arbitrarily
  repeat
    choose $s \in S$                        ($s \leftarrow s'$)
    choose $a \in A(s)$                     (derived from $Q(-, s)$)
    apply action $a$, observe $s'$ and obtain $r$
    $Q(a,s) \leftarrow Q(a,s)$
                $+\alpha(r + \gamma \max_{a'} Q(a', s') - Q(a,s))$.
```

Algorithm 1: Q-learning

The random vectors related to the approximations Q generated by the algorithm converge to the optimal action-values Q^* with probability one, provided that each state and each action is chosen with nonzero probability and the step-size parameter decreases to zero under conditions explained below. We refer to Tsitsiklis [8] and [1, pp. 247] for a proof based on the fact that Q^* is the fixed point of a contraction mapping.

The conditions for the decreasing step-size parameters α_t at iteration t are

$$\sum\nolimits_{t=1}^{\infty} \alpha_t = \infty \quad \text{and} \quad \sum\nolimits_{t=1}^{\infty} \alpha_t^2 < \infty. \quad (4)$$

The first condition allows the change in the update to be big enough to overcome any initial bias and the second ensures convergence, compare [1, p. 135] and [3, p. 39]. A usual choice for the step-size parameters that satisfies (4) is $\alpha_t = c/t$, where c is a positive constant.

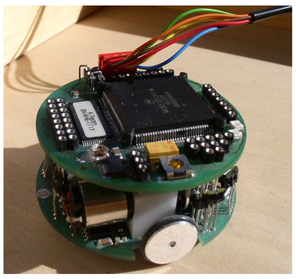
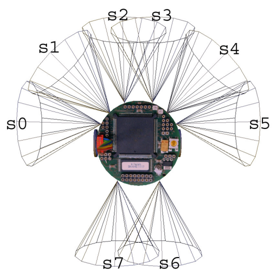

Figure 2. *left:* The robot Khepera *right:* Range and distribution of its sensors

In real world applications usually a variant of Q-learning is applied where the states are chosen according to the successor states obtained from the previous iteration, see ($s \leftarrow s'$) in Algorithm 1. Thus the chosen action influences the next state. Since every state has to be visited with nonzero probability to guarantee convergence the actions should be chosen so that the whole state space is explored. Moreover, the robot should choose actions using knowledge already gathered. This dilemma is referred to as *exploration-exploitation* and plays an important role in online learning methods, see [1, p. 251] and Thrun [9].

For *exploitation*, that is, using knowledge gained during learning, the actions are often selected according to a greedy policy for the current approximation of the action-values. Then *exploration* is applied, a technique that adds a random component to the actions selected. We call the resulting action *derived* from the current action-values, see (derived from $Q(-, s)$) in Algorithm 1. See Section 5 for a specific implementation.

4. The Robot and its Environment

Khepera is a miniature mobile robot produced by the Swiss company *K-Team*, see Figure 2 *left*. It has a cylindric shape, is 55 mm in diameter and 30 mm high. Two wheels allow the robot to move around with a maximum speed of one meter per second. It is equipped with eight infrared proximity sensors. See Mondada, Franzi and Ienne [10] for technical details.

For our experiments we developed the program `RealRobo` which controls the robot in real-time via the `Rs232` serial port, see [11] for further details. The robot is put into a wooden box as shown in Figure 3. The values received by the proximity sensors are between 0 and 1023. We normalize them between 0 and 1 and thus the set of possible sensor values is $\tilde{S} = \{0, 1/1023, \dots, 1022/1023, 1\}$, where 0 means that there is no obstacle in sight and 1 that there is an obstacle right in front. Figure 2 *right* shows the distribution and enumeration of the six front sensors and the two back sensors.

We describe how we model the robot and its environment. The possible values of the eight proximity sensors of the robot in a wooden box define the set of states, that is a subset S of all possible values

$$\{(s_0, ..., s_7) : s_i \in \tilde{S}, \quad \text{for } i \in 0 \dots 7\} \subset [0, 1]^8,$$

In every state we consider the same actions, that consist of turning by an integer angle between -90 and 90 degrees, and then moving forward a fixed distance of 4 mm. Hence the set of actions is

$$A(s) = \{a : a \in -90 \dots 90\} \quad \text{for all } s \in S.$$

In theory, if we fix a probability on all physical positions of the robot then transition probabilities can be derived from the wooden box. However, this process turns out to be infeasible since it is difficult to measure the robot's physical position and there are approximately 10^{24} states and 181 actions in each state.

We consider obstacle avoidance to be learned by the robot, see Santos [12, p. 90]. If the robot moves away from obstacles it will be rewarded, if it gets too close to an obstacle it will be punished, and otherwise it gets a neutral reinforcement. The rewards depend only on state s and successor state s' and are given by:

$$R(s', s) = \quad +1, \quad \text{if } \textstyle\sum_{i=0}^{7}(s_i - s'_i) > 0.04,$$

and in any other case

$$R(s', s) = \quad -1, \quad \text{if } \textstyle\sum_{i=0}^{7} s'_i > 0.94,$$

and 0 otherwise.

5. Q-learning Implementation

In the following two sections we discuss a method to apply Q-learning for large state and action spaces. We describe a specific variant of Q-learning for the robot but the considerations can be applied generally in the model-free case. We call the approximation of Q^* the *learning phase*. One experimental run of the robot is described by a theoretically infinite sequence

$$(\dots, a_{t+1}, s_{t+1}, a_t, s_t, \dots, s_1, a_0, s_0)$$

of states $s_t = (s_0^t, \dots, s_7^t) \in S$ and actions $a_t \in A(s_t)$. At the discrete time step t action a_t is applied to state s_t to yield the next state s_{t+1}.

Let $\tilde{\pi}_{t+1}(- \mid s_{t+1})$ be a greedy policy for the current approximation Q_{t+1}. For exploration, we add a random number $b \in [-b_t, b_t]$ to a greedy action a

of $\tilde{\pi}_{t+1}$ where $b_t > 0$ is a zero-sequence. We then choose the action a_{t+1} which is nearest to $a + b$. We obtain a probability $\pi_{t+1}(- \mid s_{t+1})$ on $A(s_{t+1})$ according to which the action a_{t+1} is selected. Note that $\pi_{t+1}(- \mid s_{t+1})$ becomes a greedy policy for the approximation Q_{t+1} for large t.

Assuming the convergence of the Q-learning variant we have

$$\lim_{t \to \infty} \max_{a \in A(s)} Q_{t+1}(a, s) = \max_{a \in A(s)} Q^*(a, s) = V^*(s),$$

for $s \in S$, and a greedy policy with respect to Q_{t+1} becomes an optimal policy for large t. Since action a_{t+1} is chosen greedily for Q_{t+1} we have $a_{t+1} \in A^*(s_{t+1})$ for large t, where

$$A^*(s) = \{a \in A(s) : Q^*(a, s) = V^*(s)\}$$

denotes the set of *optimal actions* in a state s. In other terms, for large t

$$(s_t, a_t, Q_t(a_t, s_t)) \sim (s_t, a^*, V^*(s_t)),$$

with $a^* \in A^*(s_t)$. Thus if we run the experiment for a long time the chosen actions become optimal actions. After learning we apply the obtained policy in the so called *execution phase*.

6. An Adaptive Clustering Method

Approximation methods are needed to implement learning methods for large state and action spaces. See Bertsekas and Tsitsiklis [1, pp. 59] for an introduction and overview. In our experiment we have $2^{80} \sim 10^{24}$ states and 181 actions, such that the vectors Q are contained in $\mathbb{R}^{2^{80} \cdot 181}$. It is obvious that vectors of such a high dimension cannot be computed and stored. This problem is often referred to as "the curse of dimensionality" [5, p. ix]. To implement the Q-learning method discussed in the previous section we propose a new *clustering method* based on [12, pp. 85]. Compare also Mahadevan and Connell [13]. The clustering method can be interpreted as a variant of *vector quantization* as introduced by Kohonen [14, ch. 6].

We map $A(s)$ into the unit interval $[0, 1]$ and identify $A = A(s) = \{i/180 : i = 0, \dots, 180\}$. Recall that in an experimental run of the robot we obtain, for each time step t, a triple

$$(s_t, a_t, Q_t(a_t, s_t)) \in S \times A \times \mathbb{R} \subset \mathbb{R}^{10}.$$

We encode the wanted data s, a^* and V^* in a matrix $W \in \mathbb{R}^{m \times 10}$ with a suitable number m of rows $w(j)$, for $j = 1, \dots, m$, of the form

$$w(j) = (w_s(j), w_a(j), w_q(j)) \in [0, 1]^{8+1} \times \mathbb{R}$$

and the following desired property. If s is a state then there is a row j with $w_s(j)$ nearby, $w_q(j)$ is a good approximation of $V^*(s)$ and $w_a(j)$ is close to an optimal action $a^* \in A^*(s)$. Or, in more precise terms,

$$\min_j \parallel s - w_s(j) \parallel \ll 1,$$

and for

$$j^* = \arg\min_j \parallel s - w_s(j) \parallel$$

we have $|V^*(s) - w_q(j^*)| \ll 1$ and

$$\arg\min_a |a - w_a(j^*)| \in A^*(s).$$

Here $\parallel - \parallel$ is any norm and in our implementation the *L_2-norm*, that is $\|x\|_2 = (\sum_i x_i^2)^{\frac{1}{2}}$, for $x \in \mathbb{R}^n$. In neural science the matrix W is called a *network* with m *units* $j = 1, \dots, m$ with *center* $w(j)$.

Learning Phase

The network W is determined by one or more experimental runs $(\dots, a_t, s_t, \dots, a_0, s_0)$ of the robot as the limit of networks W_t, $t = 0, 1, \dots$, with m_t units. The m_t form an increasing but not strictly increasing sequence of row dimensions. One talks about an *adaptive network* W since the number of units can change.

The sequence $(\dots, a_t, s_t, \dots, a_0, s_0)$ and the $m_t \times 10$-matrices W_t with rows $w_t(j)$, $j = 1, \dots, m_t$, are obtained as follows. We start with a randomly chosen state-action pair (a_0, s_0) and the 1×10-matrix W_0 with the row $w_0(1) = (s_0, a_0, 0)$.

After t iterations the history $(a_t, s_t, \dots, s_0)$ has been observed and the network $W_t \in \mathbb{R}^{m_t \times 10}$ has been obtained. The rows of this matrix W_t are interpreted as a good approximation of the vectors

$$(s_k, a_k, Q_k(a_k, s_k)), \quad k = 0, \dots, t,$$

where, however, $m_t \ll t$ in general. The matrix W_t is updated as explained below.

For the update we need to select greedy actions based on the current approximation. In a state s we choose a row j^* with $w_s(j^*)$ close to s and $w_q(j^*)$ as high as possible. We choose a "distance function" $d_{s,q}(x, y)$ of a nonnegative real variable x for the distance between s and w_s and a real variable y for the approximate action-value w_q, with the property that it is strictly monotonically increasing in x and decreasing in y. In our actual implementation action-values q are kept in the unit interval by cutting them off below zero and above one, and we use the distance function

$$d_{s,q}(x, q) = x^2 + \frac{(1 - q)^2}{2}.$$

Moreover, we need to select a row k^* for the current state-action pair (s, a) with $w_s(k^*)$ close to s and

$w_a(k^*)$ close to a. Again we choose a distance, in our case the L_2-norm. To decide when to add a new unit during the update we additionally choose two small constants $\delta_{s,a} > 0$ and $\delta_{s,q} > 0$. The choice of the distances and constants have to be made appropriately. The actual update proceeds in the following five steps:

Step 1. Apply action a_t to the state s_t, observe the new state s_{t+1} and compute the reward $r_t = R(s_{t+1}, a_t, s_t)$.

Step 2. Compute

$$j^* = \arg\min_j d_{s,q}(\| s_{t+1} - w_{t,s}(j) \|, w_{t,q}(j)).$$

Due the properties of $d_{s,q}$ we can use the vector $(w_{t,s}(j^*), w_{t,q}(j^*))$ as an approximation of $(s_{t+1}, \max_{a'} Q_t(a', s_{t+1}))$ in the update rule (3).

Step 3. Compute

$$\delta_1 = \min_k \|(s_t, a_t) - (w_{t,s}(k), w_{t,a}(k))\| .$$

If $\delta_1 \le \delta_{s,a}$ then compute

$$k^* = \arg\min_k \|(s_t, a_t) - (w_{t,s}(k), w_{t,a}(k))\| .$$

If $\delta_1 > \delta_{s,a}$ then add to W_t a new row or unit

$$w_t(m_t + 1) = (s_t, a_t, r_t + \gamma w_{t,q}(j^*))$$

and set $k^* = m_t + 1$. We use $w_t(k^*)$ as an approximation of $(s_t, a_t, Q_t(s_t, a_t))$ in the update rule (3).

Step 4. Define the new network W_{t+1} by

$$w_{t+1}(k) = w_t(k), \quad \text{for } k \neq k^*,$$

and update the row

$$w_t(k^*) = (w_{t,s}(k^*), w_{t,a}(k^*), w_{t,q}(k^*))$$

by

$$w_{t+1,q}(k^*) = w_{t,q}(k^*) + \alpha_t(r_t + \gamma w_{t,q}(j^*) - w_{t,q}(k^*)),$$

and

$$\begin{aligned} w_{t+1,s}(k^*) &= w_{t,s}(k^*) + \eta_{t,s}(s_t - w_{t,s}(k^*)), \\ w_{t+1,a}(k^*) &= w_{t,a}(k^*) + \eta_{t,a}(a_t - w_{t,a}(k^*)). \end{aligned}$$

The sequence α_t denotes step-size parameters for Q-learning and $\eta_{t,s}, \eta_{t,a}$ are zero-sequences of positive step-size parameters for the network.

The first update is determined by the update rule (3) with the approximate values from the preceding steps. The other two updates are typical for clustering or vector quantization. Since the vector $(w_{t+1,s}(k^*), w_{t+1,a}(k^*))$ is intended to be a better approximation of the observed state action pair (s_t, a_t) than $(w_{t,s}(k^*), w_{t,a}(k^*))$ it is chosen in the interior of the line segment between the two latter vectors. Hence the updated center moves towards (s_t, a_t). In the robot experiments we choose the constant parameters $\alpha_t = 0.7$, $\eta_{t,s} = 0.01, \eta_{t,q} = 0.2$.

Step 5. Finally the action a_{t+1} is chosen. Compute

$$\delta_2 = \min_i d_{s,q}(\| s_{t+1} - w_{t+1,s}(i) \|, w_{t+1,q}(i)).$$

If $\delta_2 \le \delta_{s,q}$ then compute

$$i^* = \arg\min_i d_{s,q}(\| s_{t+1} - w_{t+1,s}(i) \|, w_{t+1,q}(i))$$

and use $w_{t+1,a}(i^*)$ as an approximation of the greedy action a. For exploration we draw a random number $b \in [-b_t, b_t]$ and choose

$$a_{t+1} = \arg\min_a |a - w_{t+1,a}(i^*) - b|.$$

If $\delta_2 > \delta_{s,q}$ choose a_{t+1} randomly and add to the matrix W_{t+1} the new row or unit

$$w_{t+1}(m_{t+1} + 1) = (s_{t+1}, a_{t+1}, 0).$$

In our experiment we fix a number of iterations T for the learning phase. For exploration we choose $b_0 = 1$ and $b_{t+1} = b_t - 1/T$.

Execution Phase

After T iteration steps where T is sufficiently large the matrix $W = W_T$ with $m = m_T$ rows is used to construct an approximation of an optimal deterministic policy. The rows $w(j)$ in the network approximate the optimal action-values for large T. Thus a greedy policy π with respect to the network which can be computed for each state as follows is a (sub)optimal policy. For state s we select the row

$$i^* = \arg\min_i d_{s,q}(\| s - w_s(i) \|, w_q(i))$$

and choose action $a = \arg\min_a |a - w_a(i^*)|$.

7. Experiments

The robot is put in the center of the wooden box, Figure 3. During the learning phase the robot is provided with two reflexes. If it gets too close to an obstacle

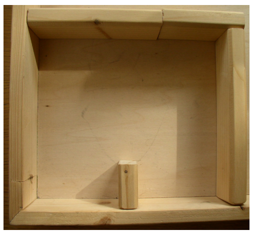

Figure 3. Environment for the robot

it undoes the last action executed. If there is no obstacle in sight it moves straight forward until it again perceives an obstacle. The robot acts randomly at the beginning due to the used exploration technique. With increasing iterations the actions of the robot become the greedy actions with respect to the current network. Figure 4 shows the increasing number of units of the network during the learning phase for an experiment with 430 iterations.

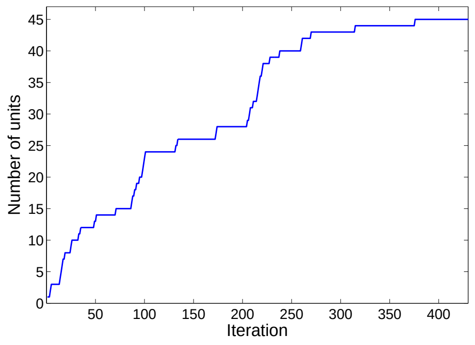

Figure 4. Number of units of the network

To compare the obtained policies we use estimates of the value function derived from sample runs, see Monte Carlo methods in [3, pp. 113] and [1, pp. 181]. The robot is put to a start position. Then the policy is executed for 100 steps and the rewards are observed. The average sum of the rewards is computed. For the random policy, a policy obtained after 150 iterations and a policy obtained after 300 iterations we conduct three sample runs each and average the obtained estimates. In Figure 5 these averages are displayed. We observe that the average sum of rewards raises with the number of iterations.

The robot performs obstacle avoidance applying the obtained policy. It moves around while remaining far away from obstacles. Once the parameters are set the proposed method obtains good policies by learning a reduced representation of the state and action sets and an approximation of optimal action-values for this representation. The learned policies turn out to be flexible and can successfully be applied to different environments.

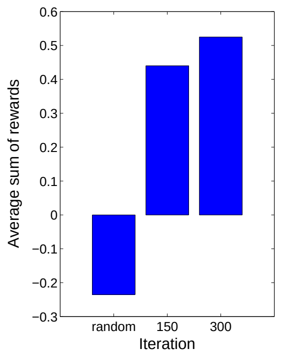

Figure 5. Average sum of rewards

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific, 1996.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[4] R. Bellman, "The theory of dynamic programming," *Bull. Amer. Math. Soc.*, vol. 60, pp. 503–515, 1954.

[5] R. Bellman, *Dynamic programming*. Princeton University Press, Princeton, N. J., 1957.

[6] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics, New York: John Wiley & Sons Inc., 1994.

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

[8] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Machine Learning*, vol. 16, no. 3, pp. 185–202, 1994.

[9] S. B. Thrun, "Efficient exploration in reinforcement learning," Tech. Rep. CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1992.

[10] F. Mondada, E. Franzi, and P. Ienne, "Mobile robot miniaturisation: A tool for investigation in control algorithms," in *Experimental Robotics III*, (Kyoto), pp. 501–513, Springer-Verlag, 1994.

[11] A. Matt and G. Regensburger, *Reinforcement Learning for Several Environments: Theory and Applications*. PhD thesis, University of Innsbruck, 2004. URL: `http://mathematik.uibk.ac.at/users/rl`.

[12] J. M. Santos, *Contribution to the study and the design of reinforcement functions*. PhD thesis, Universidad de Buenos Aires and Universit d'Aix-Marseille III, 1999.

[13] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial Intelligence*, vol. 55, pp. 311–365, June 1992.

[14] T. Kohonen, *Self-organizing maps*, vol. 30 of *Springer Series in Information Sciences*. Berlin: Springer-Verlag, 1995.