Reinforcement Learning Competition: Helicopter Hovering with Controllability and Kernel-Based Stochastic Factorization

Anwarissa Asbah ANWARISSA.ASBAH@MAIL.MCGILL.CA
André M. S. Barreto AMSB@CS.MCGILL.CA
Clement Gehring CLEMENT.GEHRING@MAIL.MCGILL.CA
Joelle Pineau JPINEAU@CS.MCGILL.CA
Doina Precup DPRECUP@CS.MCGILL.CA
School of Computer Science, McGill University, Montreal, Canada[†]

1. Introduction

The Reinforcement Learning (RL) Competition offers researchers and practitioners from the field an opportunity to empirically evaluate algorithms under a well-defined protocol. In its 2013 edition the competition is composed of three tasks: invasive species, polyathlon, and helicopter hovering.[1] This paper describes our efforts to tackle the latter problem using two value-based approaches.

The first approach is a smart exploration strategy developed by Gehring & Precup (2013), and the second one is a kernel-based algorithm recently presented by Barreto et al. (2011; 2012). We refer the reader to the original papers for a detailed description of the algorithms. Here we only provide overviews followed by a brief discussion of the algorithms's performance on the helicopter task. We start by presenting some common background material and then we proceed to discuss each approach in particular.

2. Background

The RL Competition assumes the standard framework of reinforcement learning, in which an agent interacts with an environment and tries to maximize the amount of reward collected in the long run (Sutton & Barto, 1998). The interaction between agent and environment happens at discrete time step $t = 1, 2, \ldots$. At each time step the agent observes the state of the environment, $s_t \in S$, and chooses an action $a_t \in A$. One time step later, the agent receives a reward r_{t+1} and observes a new state s_{t+1}.

A stochastic Markovian policy $\pi : S \times A \to \mathbb{R}$ defines a probability distribution for actions in each state: $\pi(s, a) = Pr(a_t = a | s_t = s)$. This choice of action is typically driven by a *value function*, which estimates the expected long-term return of each state or state-action pair. We mainly focus on the state-action value function, $Q^\pi : S \times A \to \mathbb{R}$, defined as $Q^\pi(s, a) = \mathbf{E}_\pi [r_{t+1} + \gamma r_{t+2} + \ldots | s_t = s, a_t = a]$, where $\gamma \in (0, 1)$ is a discount factor. The goal of reinforcement learning is to obtain an *optimal policy*, π^*, which has maximal value in all states.

The interaction between agent and environment can be modeled as a *Markov decision process* (MDP, Puterman, 1994). An MDP is a tuple $M \equiv (S, A, P^a, R^a, \gamma)$, where $R^a(s)$ is the expected value of the reward that will be received when a is taken in s, and $P^a(\cdot|s)$ is the next state distribution. Once the interaction between agent and environment has been modeled as an MDP, a natural way of searching for an optimal policy is to resort to *dynamic programming* (Puterman, 1994).

3. Smart Exploration

Exploration is still one of the crucial problems in reinforcement learning, especially for agents acting in safety-critical situations. Intuitively, if an agent wants to stay safe, it should seek out states where the effects of its actions are easier to predict. From the point of view of value estimation, if a particular state yields a lot of variability in the temporal-difference error signal, it is less controllable. Based on this idea, Gehring & Precup (2013) propose to use the mean absolute deviation of the temporal difference errors as a measure of controllability.

3.1. Temporal-difference learning

For a Markovian problem in which the state and action spaces are discrete and small enough, Q^π can be represented by a table with one entry for each (s, a)

[†]Authors are listed in alphabetical order.
[1]http://2013.rl-competition.org

pair. The values can be learned incrementally by the following update: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t$, where Q denotes the estimate of Q^π, $\alpha \in (0, 1)$ is the learning rate, and

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \qquad (1)$$

is the *temporal-difference (TD) error* produced as a result of the transition. Note that both a_t and a_{t+1} are assumed to be chosen according to π. It has been shown that this update process converges to correct value estimates in the limit (see references in Sutton & Barto's book). Hence, $\lim_{t\to\infty} \mathbf{E}_\pi(\delta_t) = 0$.

If the state space is very large or continuous, function approximation is used to represent Q. A common approach is to use a linear approximator of the form $Q(s_t, a_t) = \theta_{a_t}^T \phi_{s_t}$, where θ_{a_t} is a parameter vector associated with action a and ϕ_{s_t} is a feature vector corresponding to the current state. Then, the parameter vector is updated as: $\theta_{a_t} \leftarrow \theta_{a_t} + \alpha\delta_t\phi_{s_t}$, where δ_t is still computed according to (1).

3.2. Controllability

Exploration methods aim to improve the procedure for choosing actions. In particular, in *directed exploration* actions are chosen greedily with respect to a function of the form $Q(s_t, a_t) + \omega B(s_t, a_t)$, where $B(s_t, a_t)$ is a bonus that encourages visiting unknown areas.

It is common to define the exploration bonus based on the variability of the value function estimates (Brafman & Tennenholtz, 2002). Gehring & Precup (2013) propose to use the mean absolute TD error as an indication of how controllable a state is. More specifically, the controllability of a state-action pair, given a fixed policy π, is defined as: $C^\pi(s, a) = -\mathbf{E}_\pi[\ |\delta_t|\ |s_t = s, a_t = a]$. The higher the variability of the TD errors is in a given state, the lower the controllability will be. Controllability can be estimated using the TD errors in a straightforward way. At each time step t,

$$C(s_t, a_t) \leftarrow C(s_t, a_t) - \alpha'(|\delta_t| + C(s_t, a_t)), \qquad (2)$$

where $\alpha' = \beta\alpha$ is a learning rate, with $\beta < 1$. If a linear function approximation is used, controllability can be represented by a parameter vector $\mathbf{w}$. The parametrization of C should be the same as for Q, and the update of $\mathbf{w}$ representing C becomes:

$$\mathbf{w}_{a_t} \leftarrow \mathbf{w}_{a_t} - \alpha'(|\delta_t| + \mathbf{w}_{a_t}^T \phi_{s_t})\phi_{s_t}. \qquad (3)$$

Gehring & Precup (2013) show that the sequence of estimates generated by (2) or (3) is convergent.

The exploration algorithm uses controllability as an exploration bonus, picking actions greedily according to $Q(s_t, a_t) + \omega C(s_t, a_t)$, where ω is a parameter used to trade off between the desire to obtain high returns and the minimization of the TD error signals. If $\omega = 0$, the algorithm relies only on the values. If ω is high, the emphasis is on visiting areas of high controllability.

Gehring & Precup (2013) have addressed the helicopter task using a "safe" version of Sarsa that uses controllability to guide exploration (see Figure 1). In their experiments the authors used the helicopter environment provided as part of the RL library.[2] Although this version of the task is not the same used in the RL competition, the improvement on Sarsa's performance provided by smart exploration suggests that controllability might be a promising approach.

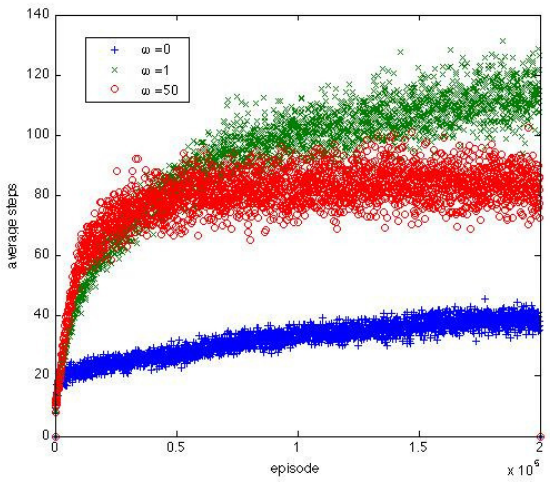


Figure 1. Results on the helicopter task achieved by Sarsa algorithm using controllability. Each point correspond to an average over 50 runs. Originally published by Gehring & Precup, 2013 (see paper for details of the experiment).

4. Practical Kernel-Based Learning

Kernel-based reinforcement learning (KBRL,Ormoneit & Sen, 2002) stands out among approximate reinforcement learning algorithms for its nice theoretical guarantees. Unfortunately, the size of KBRL's approximator grows with the number of sample transitions, which makes the approach impractical for large problems. This section describes an algorithm that turns KBRL into a practical tool.

4.1. Kernel-based reinforcement learning

Consider an MDP M with continuous state space $\mathbb{S} \subset [0, 1]^d$. KBRL uses sample transitions to derive a finite MDP that approximates the continu-

[2] http://library.rl-community.org/wiki/Helicopter(Java)

ous model (Ormoneit & Sen, 2002). Let $S^a = \{(s_k^a, r_k^a, \hat{s}_k^a) | k = 1, 2, ..., n_a\}$ be sample transitions associated with action $a \in A$, where $s_k^a, \hat{s}_k^a \in \mathbb{S}$ and $r_k^a \in \mathbb{R}$. Let $\phi : \mathbb{R}^+ \mapsto \mathbb{R}^+$ be a Lipschitz continuous function and let $\mathrm{k}_\tau(s, s')$ be a kernel function defined as $\mathrm{k}_\tau(s, s') = \phi(\|s - s'\|/\tau)$, where $\|\cdot\|$ is a norm in $\mathbb{R}^d$ and $\tau > 0$. Finally, define the normalized kernel function associated with action a as $\kappa_\tau^a(s, s_i^a) = \mathrm{k}_\tau(s, s_i^a) \Big/ \sum_{j=1}^{n_a} \mathrm{k}_\tau(s, s_j^a)$. KBRL uses κ_τ^a to build a finite MDP $\hat{M}$ whose state space $\hat{S}$ is composed solely of the $n = \sum_a n_a$ states $\hat{s}_i^a$. The transition and reward functions of KBRL's model are given by:

$$\hat{P}^a(s'|s) = \begin{cases} \kappa_\tau^a(s, s_i^a), \text{ if } s' = \hat{s}_i^a, \\ 0, \text{ otherwise} \end{cases} \quad \text{and} \quad \hat{R}^a(s, s') = \begin{cases} r_i^a, \text{ if } s' = \hat{s}_i^a, \\ 0, \text{ otherwise.} \end{cases} \tag{4}$$

After $\hat{Q}^*$, the optimal action-value function of $\hat{M}$, has been computed, the value of any state-action pair can be determined as: $Q(s, a) = \sum_{i=1}^{n_a} \kappa_\tau^a(s, s_i^a) \left[r_i^a + \gamma \max_a \hat{Q}^*(\hat{s}_i^a, a) \right]$, where $s \in \mathbb{S}$ and $a \in A$. Ormoneit & Sen (2002) proved that, if $n_a \to \infty$ for all $a \in A$ and the widths of the kernels τ shrink at an "admissible" rate, the probability of acting sub-optimally based on $Q(s, a)$ goes to zero.

Using dynamic programming, one can compute the optimal value function of $\hat{M}$, but the time and space required to do so grow fast with the number of states n (Puterman, 1994).

4.2. Kernel-based stochastic factorization

KBSF compresses the information contained in KBRL's model $\hat{M}$ in an MDP $\bar{M}$ whose size is independent of the number of transitions n. The core idea behind KBSF is the "stochastic-factorization trick", which we now summarize. Let $\mathbf{P} \in \mathbb{R}^{n \times n}$ be a transition probability matrix and let $\mathbf{P} = \mathbf{DK}$ be a factorization such that $\mathbf{D} \in \mathbb{R}^{n \times m}$ and $\mathbf{K} \in \mathbb{R}^{m \times n}$ are stochastic matrices. Then, *swapping* the factors $\mathbf{D}$ and $\mathbf{K}$ yields another transition matrix, $\bar{\mathbf{P}} = \mathbf{KD}$, which is potentially much smaller than $\mathbf{P}$ but retains some of its fundamental properties (Barreto & Fragoso, 2011).

KBSF emerges from the application of the stochastic-factorization trick to KBRL's MDP. In particular, one replaces the MDP $\hat{M} = \{\mathbb{S}, A, \hat{\mathbf{P}}^a, \hat{\mathbf{r}}^a, \gamma\}$ with a smaller model $\bar{M} = \{\bar{S}, A, \bar{\mathbf{P}}^a, \bar{\mathbf{r}}^a, \gamma\}$, where $\bar{S}$ is a set of representative states (*e.g.* sampled via trajectories, or from clustering), and $\bar{\mathbf{P}}^a = \mathbf{K}^a \mathbf{D}^a$. The matrix $\mathbf{K}^a \in \mathbb{R}^{m \times n_a}$ has elements $k_{ij}^a = \kappa_\tau^a(\bar{s}_i, s_j^a)$ and $\mathbf{D}^a \in \mathbb{R}^{n_a \times m}$ has elements $d_{ij}^a = \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_j)$, where $\bar{\kappa}_{\bar{\tau}}$ is defined as $\bar{\kappa}_{\bar{\tau}}(s, \bar{s}_i) = \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_i) / \sum_{j=1}^{m} \bar{\mathrm{k}}_{\bar{\tau}}(s, \bar{s}_j)$. The entries of the vectors $\bar{\mathbf{r}}^a$ are given by $\bar{r}_i^a = \sum_j k_{ij}^a r_j^a$.

Instead of solving an MDP with n states, KBSF solves a model with m states only. This results in an algorithm that can be orders of magnitude faster than KBRL but still converges to a unique solution. KBSF is also supported by theoretical guarantees, since it can be shown that the distance between KBRL's solution and KBSF's solution is bounded (Barreto et al., 2011).

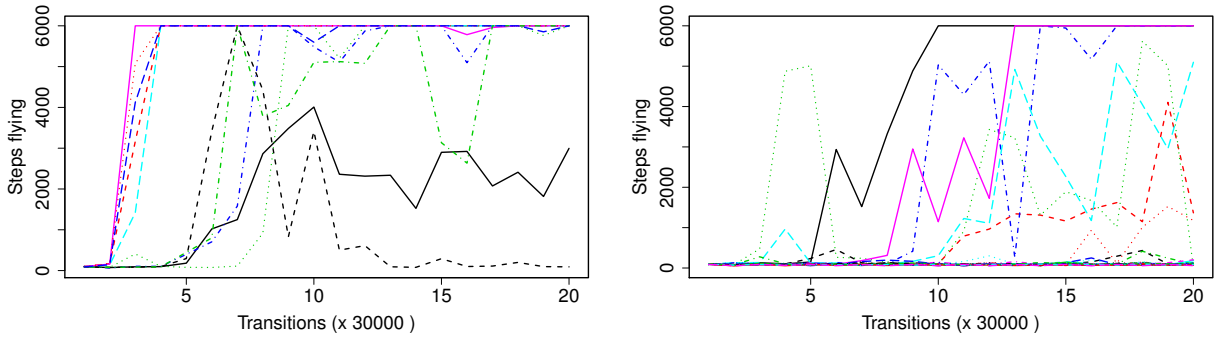
Despite these nice properties, KBSF's memory usage grows linearly with the number of transitions, precluding its application in scenarios where a very large amount of data must be processed. Barreto et al. (2012) have shown that it is possible to construct the KBSF solution in a fully incremental way. The incremental version of KBSF (*i*KBSF) can be used to solve large continuous MDPs on-line.

5. Results

Helicopter hovering is a difficult task. Part of this difficulty is due to the complex dynamics of the helicopter, which is not only non-linear, noisy, and asymmetric, but also counterintuitive in some aspects (Ng et al., 2003). The problem is also high-dimensional, with a 12-dimensional continuous state space and a 4-dimensional continuous action space.

To make things even more complicated, in the RL Competition one must tackle a generalized version of the problem. A generalized task is not a single MDP, but a distribution over related MDPs that vary along specific dimensions (for example, in previous editions of the competition the intensity of the wind in the helicopter task changed from MDP to MDP, as explained by Koppejan & Whiteson, 2011). In the 2013 edition of the competition 10 MDPs were provided for "training", and the final performance was evaluated on another sample of 30 MDPs.

Despite the difficulty of the helicopter problem, there exist simple policies that perform relatively well. One example is the "weak controller" provided with the competition software: by determining the value of each variable of the action space as a linear combination of a subset of the state variables, such controller is able to fly the helicopter without crashing on most instances of the task. Not surprisingly, all approaches that have been successful on the helicopter task are policy search methods. Among these, some are able to obtain impressive performances by exploiting prior knowledge about the problem, such as using demonstration data coming from an expert (see Koppejan & Whiteson's article and references therein).



(a) KBSF's results on 10 "training" MDPs

(b) KBSF's results on 30 "test" MDPs

Figure 2. KBSF's results on the helicopter task. Each curve corresponds to a different MDP. All results were generated with the same configuration of the algorithm.

Here we want to address the helicopter task using the value-based methods described in the previous sections, relying as much as possible on data—that is, keeping the use of prior knowledge to a minimum. Note that the fact that simple policies can perform well on the helicopter problem does not imply that either the value function or the transition kernels of the underlying MDP are simple as well.

Both Sarsa and KBSF assume a finite number of actions, and thus we had to discretize the problem's action space. In order to do so, we started with a high-resolution discretization and slowly decreased the number of break points until the performance of the weak controller degenerated. By doing so we eliminated one dimension of the action space, ending up with 18 actions ($3 \times 3 \times 2$).

Using the action-space discretization described above Sarsa combined with controllability generated results slightly better than those shown in Figure 1. However, the algorithm was unable to consistently fly the helicopter for more than 1000 steps.

The "naive" version of KBSF was also unable to fly the helicopter for more than 1000 steps, so we turned to a specialized version that addressed a decomposed version of the problem. In particular, instead of a single agent, we used three agents, each responsible for one active dimension of the action space. Each agent only had access to a subset of the state variables; these sets were defined based on the weak controller.

Figure 2 shows KBSF's results on the helicopter task. As shown in Figure 2a, KBSF performs well on the training phase, being able to fly the helicopter for the maximum number of 6000 steps in 8 out of 10 MDPs. To the best of our knowledge, these are the best results obtained by a value-based method on the helicopter task. Unfortunately, the good performance was not repeated on the testing phase, as shown in Figure 2b. The reason for this is still unclear; one possible explanation is that the test MDPs are very different from the training models, either by chance or because they were sampled from a different distribution. We are currently investigating this phenomenon with the objective of improving KBSF's ability to generalize.

References

Barreto, A. and Fragoso, M. Computing the stationary distribution of a finite Markov chain through stochastic factorization. *SIAM Journal on Matrix Analysis and Applications*, 32:1513–1523, 2011.

Barreto, A., Precup, D., and Pineau, J. Reinforcement learning using kernel-based stochastic factorization. *NIPS*, 2011.

Barreto, A., Precup, D., and Pineau, J. On-line reinforcement learning using incremental kernel-based stochastic factorization. *NIPS*, 2012.

Brafman, R. and Tennenholtz, M. R-max, a general polynomial time algorithm for near-optimal reinforcement learning. *JMLR*, pp. 213–231, 2002.

Gehring, C. and Precup, D. Smart exploration in reinforcement learning using absolute temporal difference errors. *AAMAS*, 2013.

Koppejan, R. and Whiteson, S. Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. *Evol. Intell.*, 4(4):219–241, 2011.

Ng, A., Kim, H., Jordan, M., and Sastry, S. Autonomous helicopter flight via reinforcement learning. *NIPS*, 2003.

Ormoneit, D. and Sen, S. Kernel-based reinforcement learning. *Machine Learning*, 49 (2–3):161–178, 2002.

Puterman, M. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley, 1994.

Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction*. MIT Press, 1998.