Kernel-Based Stochastic Factorization for Batch Reinforcement Learning

André M. S. Barreto
Laboratório Nacional de Computação Científica
Petrópolis, RJ, Brazil
amsb@lncc.br

Doina Precup
Computer Science, McGill University
Montreal, QC, Canada
dprecup@cs.mcgill.ca

Recent years have witnessed the emergence of several reinforcement-learning techniques that make it possible to learn a decision policy from a batch of sample transitions. Among them, *kernel-based reinforcement learning* (KBRL [5]) stands out for two reasons. First, unlike other approximation schemes KBRL always converges to a unique solution. Second, KBRL is consistent in the statistical sense, meaning that additional data always improve the quality of the resulting policy and eventually lead to optimal performance.

Despite its nice theoretical properties, KBRL has not been widely adopted by the reinforcement-learning community. One possible explanation for this is the high computational cost of this approach. As discussed by Jong and Stone [3], KBRL can be seen as the derivation of a finite Markov decision process (MDP) with n states, where n is the number of sample transitions used in the approximation. This dependence on n gives rise to a dilemma: on the one hand one wants as much data as possible to describe the dynamics of the system, but on the other hand the number of transitions should be small enough to allow for the numerical solution of the resulting model. In this short note we describe an algorithm that provides a practical way of weighting the relative importance of these two conflicting objectives. The idea is to fix the size of the MDP generated by KBRL and to incorporate to this model all the information contained in the data.

The approach proposed here is based on a special decomposition of a transition matrix called *stochastic factorization* [2, 1]:

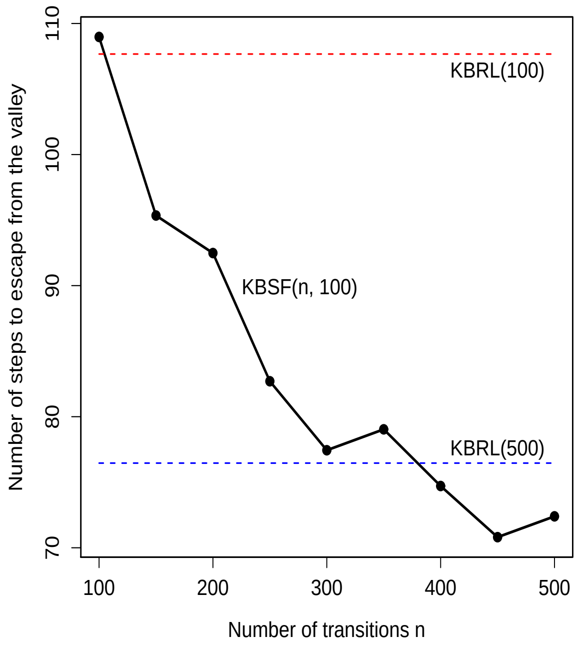
Definition. *Given a stochastic matrix* $\mathbf{P} \in \mathbb{R}^{n \times p}$*, the relation* $\mathbf{P} = \mathbf{DK}$ *is called a* stochastic factorization *of* $\mathbf{P}$ *if* $\mathbf{D} \in \mathbb{R}^{n \times m}$ *and* $\mathbf{K} \in \mathbb{R}^{m \times p}$ *are also stochastic matrices.*

The stochastic factorization is a particular case of nonnegative matrix factorization. Its importance lies on the following intriguing property: given a stochastic factorization of a square matrix, $\mathbf{P} = \mathbf{DK}$, one can *swap* the factors of the factorization to obtain another transition matrix $\bar{\mathbf{P}} = \mathbf{KD}$ which retains some fundamental characteristics of $\mathbf{P}$. In particular, it is possible to show that each recurrent class in $\mathbf{P}$ has a corresponding such class in $\bar{\mathbf{P}}$ with the same reducibility and same period [2]. Therefore, depending on the application of $\mathbf{P}$, one can replace this matrix with $\bar{\mathbf{P}}$, which may result in substantial computational gains when $m \ll n$. For example, the stochastic factorization can be used to reduce the number of operations involved in the computation of a Markov chain's stationary distribution or in the determination of a decision policy's value function [2, 1].

In the case of KBRL, the stochastic factorization "trick" can be applied to reduce the number of states in the MDP generated by this algorithm. Let $X^a = \{(x_i^a, r_i^a, y_i^a) | i = 1, 2, ..., n_a\}$ be a set of sample transitions associated with action $a \in A$, where $x_i^a, y_i^a \in S$ and $r_i^a \in \mathbb{R}$. The MDP constructed by KBRL has the following transition and reward functions [3]:

$$P^a(x_j|x_i) = \begin{cases} \kappa^a(x_i, x_k^a), \text{ if } x_j = y_k^a, \\ 0, \text{ otherwise} \end{cases} \quad \text{and} \quad R^a(x_i, x_j) = \begin{cases} r_k^a, \text{ if } x_j = y_k^a, \\ 0, \text{ otherwise,} \end{cases}$$

where κ^a is an appropriately defined kernel [5]. Since only transitions ending in states y_i^a have a nonzero probability of occurrence, one can solve a finite MDP composed by these states only. After the optimal value function of such MDP has been found, the values of the original states can be computed as $Q(x, a) = \sum_{i=1}^{n_a} \kappa^a(x, x_i^a) \left[r_i^a + \gamma V^*(y_i^a) \right]$, where γ is the discount factor [6]. The strategy of *kernel-based stochastic factorization* (KBSF) is to select a set of representative states $\{q_1, q_2, ..., q_m\}$ and then create matrices $\mathbf{D}^a$ and $\mathbf{K}^a$ whose elements are given by: $d_{ij}^a = \kappa^q(y_i^a, q_j)$ and $k_{ij}^a = \kappa^a(q_i, x_j^a)$, where κ^q is also a kernel. Depending on how the states q_i and the kernels κ^q are defined, one has $\mathbf{D}^a \mathbf{K}^a \approx \mathbf{P}^a$ for all $a \in A$. The important point is that the matrices $\mathbf{P}^a$ are never actually computed, but instead one solves an MDP with m states whose dynamics are given by $\bar{\mathbf{P}}^a = \mathbf{K}^a\mathbf{D}^a$ and $\bar{\mathbf{r}}^a$, with $\bar{r}_i^a = \sum_j k_{ij}^a r_j^a$. Notice that, depending on the value of m, this strategy may result in a significant economy in terms of computational effort. As for theoretical guarantees, it is possible to bound $\| \mathbf{v}^* - \mathbf{D}\bar{\mathbf{v}}^* \|$, but unfortunately space limitation precludes a full presentation of the proof here. Empirically, it has been observed that KBSF is able to find very good decision policies using only a fraction of the arithmetic operations performed by other algorithms. Figure 1 shows a small sample of the computational experiments carried out so far.



Algorithm	Ep.[*]	Steps Mean	Steps S.D.[†]
SINGLE POLE			
KBRL(600)	26.36	1069.61	1243.83
KBRL(900)	62.28	1931.43	1380.41
KBRL(1200)	74.57	2360.12	1145.55
KBRL(1500)	55.19	1866.84	1319.84
KBSF(94787, 25)	92.59	2779.15	781.07
KBSF(94787, 50)	99.20	2976.09	265.87
KBSF(94787, 75)	100.00	3000.00	0.00
KBSF(94787, 100)	100.00	3000.00	0.00
DOUBLE POLE			
LSPI(17319,100)	4.44	296.93	646.01
LSPI(17319,150)	9.01	396.77	841.03
LSPI(17319,200)	0.00	71.01	79.91
LSPI(17319,250)	1.11	120.59	359.44
KBSF(17319,100)	42.47	1568.57	1321.06
KBSF(17319,150)	61.98	2128.85	1246.90
KBSF(17319,200)	71.23	2348.42	1140.52
KBSF(17319,250)	75.06	2381.21	1161.26

[*]Percentage of episodes in which the pole was balanced for 3000 steps; [†]Standard deviation

Mountain car [6]: transitions uniformly sampled from S using a generative model; states q_i evenly distributed over the state space S; test set composed by 25 states equally spaced on $[-1, -0.07] \times [0.15, 0.02]$.

Pole balancing [7]: transitions collected by a random exploration policy in 1000 episodes; states q_i defined by k-means; test set composed by 81 states equally spaced in the region defined by $\pm[1.8, 1.8, 3\pi/20, 3\pi/20]$, for the single pole case, and by $\pm[0.72, 0.72, 3\pi/50, 3\pi/50, 0, 0]$ for the double pole version of the problem. LSPI [4] and KBSF used the same Gaussian kernels.

Figure 1: Results of computational experiments. The first number after the algorithms' names is n, the number of sample transitions used in the approximation, and the second one is m, the size of the model generated by each method (in the case of KBRL n and m coincide). The decision policies were evaluated in test sets composed by states from which the task could not be trivially solved (see above). The results correspond to an average computed over 20 independent runs of each algorithm.

References

[1] A. M. S. Barreto. Reducing the dimension of a Markov decision process through stochastic factorization. 2010. Submitted.

[2] A. M. S. Barreto and M. D. Fragoso. Computing the stationary distribution of a finite Markov chain through stochastic factorization. 2010. Submitted.

[3] N. Jong and P. Stone. Kernel-based models for reinforcement learning in continuous state spaces. In *Proceedings of ICML—Workshop on Kernel Machines and Reinforcement Learning*, 2006.

[4] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

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

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

[7] A. P. Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of IJCNN*, volume 2, pages 667–673, 1991.