Goal-Driven Dimensionality Reduction for Reinforcement Learning

Simone Parisi[1], Simon Ramstedt[1] and Jan Peters[1,2]

Abstract— **Defining a state representation on which optimal control can perform well is a tedious but crucial process. It typically requires expert knowledge, does not generalize straightforwardly over different tasks and strongly influences the quality of the learned controller. In this paper, we present an autonomous feature construction method for learning low-dimensional manifolds of goal-relevant features jointly with an optimal controller using reinforcement learning. Our method combines information-theoretic algorithms with principal component analysis to performs a return-weighted reduction of the state representation. The method does not require any preprocessing of the data, does not assume strong restrictions on the state representation, and substantially improves the performance of learning by reducing the number of samples required. We show that our method can learn high quality controller in redundant spaces, even from pixels, and outperforms both classical and state-of-the-art deep learning approaches.**

I. Introduction

In recent years reinforcement learning (RL) has become increasingly important in the field of robot learning [1], [2]. RL algorithms search for optimal controllers, also called policies, in a subset of the controller space using the experienced reward from the sampled trajectories as quality assessment for the controller. To perform the search, they often rely on concise and dense, but informative, state descriptions and are particularly well-suited for solving tasks in continuous state-action spaces. However, the quality of the learned controller strongly depends on the state representation and a poor representation is likely to result in an unsuccessful controller. Nonetheless, complex tasks often result in large sparse spaces. Such high-dimensional representations can contain features which are redundant or even irrelevant for both the description of the state and the goal of the agent. Scaling up RL by learning a parsimonious state representation is thus an important step towards more autonomous and widely applicable algorithms on real-world robotics problem.

Furthermore, learning a state representation for an RL algorithm jointly with an optimal controller introduces a circular dependency into the learning process. To learn goal-relevant features a large part of the state space needs to be explored using a goal-achieving controller. At the same time, learning a locally or globally optimal controller requires a concise feature representation. For this reason, common approaches separate the problem of learning the state representation from the RL task by preprocessing the feature space [3], [4] (Figure 2a). As it precedes the RL procedure, such a separated dimensionality reduction (DR) step requires additional state samples that are frequently not helpful for the learning itself. The state representation resulting from a preceding DR step, in fact, does not consider the importance of features for optimal controller but only for general controller. Thus, the state representation will not optimally support the RL algorithm as it may establish features that are only relevant for suboptimal or even poor controller.

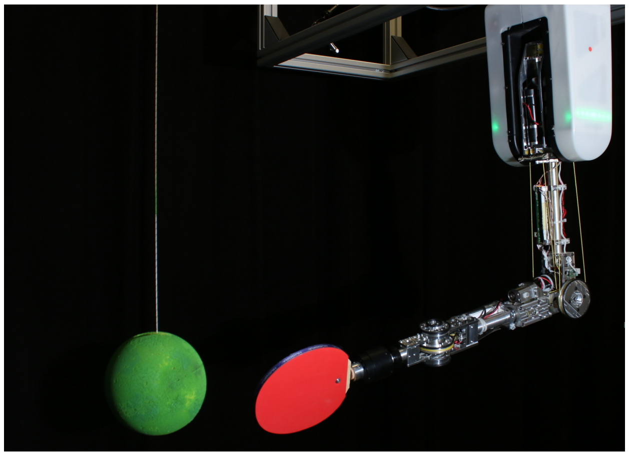


Fig. 1: A real-world example of learning with high-dimensional redundant input. The robot has to hit the green ball given pictures of its surroundings. However, only few pixels (the ball and the paddle) are relevant for the task.

Temporal difference approaches, on the other hand, do not require additional steps and reduce the dimensionality of the features while learning, e.g., by regularizing the states value function approximation or by random projections [5], [6], [7], [8], [9], [10], [11], [12], [13]. However, the quality of these approaches depends on the quality of the value function approximation, which is challenging in high-dimensional continuous spaces. A notable exception is deep learning algorithms, which use deep neural networks as approximators [14], [15]. These algorithms have achieved impressive results, being able to learn complex controllers from raw pixels input, but are computationally demanding and require many samples.

On the other hand, RL has been very favorable in many domains such as robotics as it allows task-appropriate pre-structured policies to be integrated straightforwardly and expert's knowledge can be incorporated with ease. By selecting a suitable controller parametrization, the learning problem can be simplified and stability, as well as robustness, can frequently be ensured [16].

This work was funded by a DFG grant within the priority program "Autonomous learning" (SPP1527).

[1]Autonomous Systems Labs, Technical University of Darmstadt, 64289 Darmstadt, Germany {last_name}@ias.tu-darmstadt.de

[2]Max Planck Institute for Intelligent Systems, 72076 Tübingen, Germany

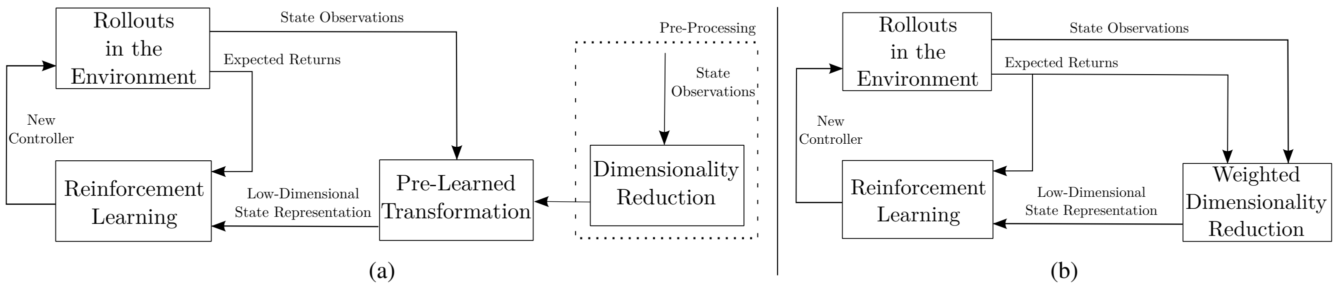


Fig. 2: In classical robot learning approaches (a), the learning of a low-dimensional state representation is decoupled from the learning of an optimal controller. On the contrary, our approach (b) directly integrates DR with RL to lean goal-relevant features jointly with a goal-achieving controller.

In this paper, we present a novel approach addressing the limitations of current state-of-the-art in feature construction and RL for optimal control. First, unlike temporal difference approaches, our method performs return-based feature construction *without relying on the value function to select the optimal action*. It is therefore applicable to problems with continuous state-action spaces. Second, unlike subspace invariant approaches, it is *goal-directed* and learns an approximate state representation which is relevant for the RL task. Third, it is *sample-efficient*, as it does not require any additional samples for preprocessing the state representation.

The remainder of the paper is organized as follows. We start by introducing the notation used in this paper and discuss the problem we aim to solve. We then present our approach, showing how to combine the learning of an optimal controller with the learning of a low-dimensional state representation. Subsequently, we evaluate our approach on two problems and compare it to related approaches in RL. Finally, we summarize the contributions of our paper and propose possible avenues of investigation for future research.

II. Problem Statement and Notation

We consider finite-horizon Markov Decision Processes (MDPs), a mathematical framework described by the tuple $\langle \mathcal{X}, \mathcal{U}, \mathcal{P}, \mathcal{R}, \mu \rangle$, where $\mathcal{X} \subseteq \mathbb{R}^{d_\text{x}}$ is the continuous state space and $\mathcal{U} \subseteq \mathbb{R}^{d_\text{U}}$ is the continuous action space. $\mathcal{P}$ is a Markovian transition model and $\mathcal{P}(x'|x,u)$ defines the transition density between states x and x' under action u. The mapping $\mathcal{R} : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the reward function and μ the initial state distribution ($x_1 \sim \mu$). The goal of the agent is to learn a controller π maximizing the expected cumulative reward (or expected return). The controller is typically described by a conditional distribution $\pi(u|x)$ specifying the probability of taking action u in state x. The optimal control problem we want to solve is

$$\max_{\pi} \iint \mu_\pi(x)\pi(u|x)\mathcal{R}(x,u)\,\mathrm{d}x\,\mathrm{d}u, \qquad (1)$$

where the stationary state distribution μ_π represents the probability of visiting state x when following π. Here, we consider *parametric* controllers $\pi \in \mathit{\Pi}^{\boldsymbol{\theta}} = \{\pi^{\boldsymbol{\theta}} | \boldsymbol{\theta} \in \boldsymbol{\Theta} \subseteq \mathbb{R}^{d_{\boldsymbol{\theta}}}\}$ and we want to learn the parameters $\boldsymbol{\theta}$ representing a locally or globally optimal controller. For simplicity, in the remainder of the paper we write just π or $\boldsymbol{\theta}$ instead of $\pi^{\boldsymbol{\theta}}$.

RL algorithms approximate the integrals above by sample trajectories $\boldsymbol{\tau} = \{x_t, u_t\}_{t=1}^{H}$ of length H drawn from the conditional distribution $\mu_\pi(x)\pi(u|x)$. However, their performance is highly coupled to how proficiently they explore the state space. Usually, the controller relies on some basis functions (or features) $\phi : \mathcal{X} \to \mathcal{F} \in \mathbb{R}^m$ to represent relevant state information. We denote this dependence by π_ϕ. Nonetheless, using generic basis functions (e.g., polynomials, radial basis functions) often results in large feature spaces, potentially containing redundant information for the agent's goal. Similarly, large rich state input (e.g., full images) can contain irrelevant information for the task (e.g., if the goal of the agent is to hit a ball, it may be irrelevant to know other objects positions). In the light of the above, we assume that, given a generic state representation ϕ, the features of interest for the RL problem lie on an embedded lower-dimensional manifold $\widetilde{\mathcal{F}} \in \mathbb{R}^n$ with $n < m$. We reformulate Eq. (1) as

$$\max_{\pi, \widetilde{\phi}} \iint \mu_\pi(x)\pi_{\widetilde{\phi}}(u|x)\mathcal{R}(x,u)\,\mathrm{d}x\,\mathrm{d}u. \qquad (2)$$

where $\widetilde{\phi} : \mathcal{X} \to \widetilde{\mathcal{F}}$. That is, we want to optimize the state representation and the controller at the same time. Consequently, the state space is explored by a goal-achieving controller, which simultaneously is optimized with the aid of parsimonious features. Therefore, we combine goal-directed DR techniques with RL. The idea of our method, summarized in Figure 2b, is to use the experienced return R to update both the low-dimensional state representation $\widetilde{\phi}$ and the controller π at each iteration k, i.e.,

$$\widetilde{\phi}_{k+1} = \mathrm{DR}\left(\widetilde{\phi}_k, \boldsymbol{d}_k\right), \qquad \pi_{k+1} = \mathrm{RL}\left(\pi_k, \widetilde{\phi}_{k+1}\right),$$

where DR is a dimensionality reduction algorithm with return-based weights $\boldsymbol{d}$, and RL is a reinforcement learning algorithm. Although this formulation does not rely on any specific algorithm, some controller update strategies might be inapplicable, e.g., when the number of parameters $\boldsymbol{\theta}$ is fixed, as in gradient-based approaches. Furthermore, the quality of the state representation strongly depends on the weights $\boldsymbol{d}$. As the goal of the agent is to maximize the cumulative sum of the rewards, weighting a sample by the corresponding immediate reward would not result in a state representation $\widetilde{\phi}$ helpful for maximizing the expected return. In the next sections, we address these issues and present suitable algorithms for our approach.

III. Reinforcement Learning with Integrated Dimensionality Reduction

We begin by discussing our method in the classical RL scenario introduced in Section II. We formulate an algorithm allowing the controller to change its number of parameters at each iteration. This trait is essential, as the low-dimensional state representation learned at each iteration influences the number of controller parameters. Subsequently, we identify a return-weighted procedure to construct the low-dimensional features exploited by the RL algorithm.

A. Relative Entropy Policy Search

Expectation-maximization (EM) is a well-know algorithm for finding the maximum likelihood (ML) solution of a probabilistic latent variable model. The problem of learning the optimal controller parameters can be formulated as an EM problem [17]: the goal is to find the controller parameters $\boldsymbol{\theta}$ (the model) maximizing the probability $p(R|\boldsymbol{\tau})$ of experiencing rewards (the observed variable) during a trajectory (the latent variable). That is, we want to find the parametric distribution $p_{\boldsymbol{\theta}}(R) = \int_{\boldsymbol{\tau}} p_{\boldsymbol{\theta}}(R, \boldsymbol{\tau}) \, \mathrm{d}\boldsymbol{\tau}$ maximizing the log-likelihood of the observed data. This update strategy has several advantages. First, many types of controller, including non-parametric ones, can be fitted. Second, for controllers belonging to the exponential distribution family the ML can be determined in closed form. Third, EM updates would particularly benefit from dimensionality reduction, as the number of the controller parameters depends on the number of basis functions used to represent the state. Therefore, less parameters require less samples for an accurate fit. Finally, as we will show below, this update strategy also provides reasonable return-based weights applicable for the DR step.

One of the most prominent EM-based algorithms is Relative Entropy Policy Search (REPS) [18]. REPS performs EM while keeping a sufficient level of statistical information w.r.t. a reference distribution q — typically the previous sampling distribution — in order to balance exploration and exploitation. The level of statistical information is measured using the KL divergence and the corresponding optimization problem is

$$\begin{aligned}
\max_{\pi} & \iint \mu_\pi(x)\pi(u|x)\mathcal{R}(x,u) \, \mathrm{d}x \, \mathrm{d}u \\
\text{s.t.} & \iint \mu_\pi(x)\pi(u|x) \, \mathrm{d}x \, \mathrm{d}u = 1 \\
& \iint \mu_\pi(x)\pi(u|x) \log \frac{\mu_\pi(x)\pi(u|x)}{q(x,u)} \, \mathrm{d}x \, \mathrm{d}u \leq \epsilon \\
& \iint \mu_\pi(x)\pi(u|x)\mathcal{P}(x'|x,u) \, \mathrm{d}x \, \mathrm{d}u = \mu_\pi(x'), \ \forall x' \quad (3)
\end{aligned}$$

where the first constraint ensures that π is a distribution, the second constrain the KL divergence between the controller and the reference distribution, and the third guarantees that the stationary state distribution μ_π comply with the given state dynamics and the controller. However, in continuous spaces, this formulation results in infinitely many constraints. To keep the problem tractable, we require that the distributions only match on their expected features $\phi(x)$. Therefore, we need to resort to matching feature expectations instead of matching single probabilities.

$$\iiint \mu_\pi(x)\pi(u|x)\phi(x')\mathcal{P}(x'|x,u) \, \mathrm{d}x \, \mathrm{d}u \, \mathrm{d}x' = \int \mu_\pi(x')\phi(x') \, \mathrm{d}x'.$$

Using the method of Lagrangian multipliers, we can solve the problem above and obtain an analytical solution for the new controller

$$\pi(u|x) \propto q(u|x) \exp\left(\frac{\delta(x,u,V)}{\eta}\right), \quad (4)$$

$$\delta(x,u,V) = \mathcal{R}(x,u) + \int V(x')\mathcal{P}(x'|x,u) \, \mathrm{d}x' - V(x) \quad (5)$$

where η and $V(x)$ are the Lagrangian multipliers and $\delta(x,u,V)$ denotes the Bellman error. The Lagrangian multipliers are obtained by minimizing the dual function of the original optimization problem

$$g(\eta,V) = \eta\epsilon + \eta \log \iint q(x,u) \exp\left(\frac{\delta(x,u,V)}{\eta}\right) \mathrm{d}x \, \mathrm{d}u. \quad (6)$$

Since we have access only to trajectories sampled by q, we update our controller π by fitting a parametric distribution to our samples. This parametric distribution is obtained by a weighted ML estimate on the samples $\{x^{[i]}, u^{[i]}\}$ with weightings $d^{[i]} = \exp\left(\delta(x^{[i]}, u^{[i]}, V)/\eta\right)$.

Therefore, REPS reduces RL to an iterative schema where at each step the new controller parameters are derived from the weighted ML estimate on samples collected by the previous controller. If the controller relies on features ϕ, samples $\phi(x^{[i]})$ are used for the ML estimate as well. Furthermore, since the weights $d^{[i]}$ account for the Bellman error — not the immediate reward — they are suitable for our purpose of samples weighting.

B. Constructing Goal-Relevant Features

Given a set of observed features $\mathit{\Phi} = \{\phi(x^{[i]})\}_{i=1}^{N}$, the DR problem consists of finding a lower-dimensional set $\widetilde{\mathit{\Phi}} = \{\widetilde{\phi}(x^{[i]})\}_{i=1}^{N}$ by learning a mapping from $\phi(x)$ to $\widetilde{\phi}(x)$ which preserves as much as information as possible about the state. However, we are concerned with transformations that aid the learning process, i.e., we aim to find a mapping that keeps only information relevant for the RL task. As the goal of the agent is to maximize the expected return, we want to perform a weighted DR taking into account the previously learned weights $d^{[i]}$. For this purpose, building on [19], [20] we develop *return-weighted principal component analysis (rwPCA)*. rwPCA solves the eigenproblem $\mathrm{Cov}(\boldsymbol{D}\boldsymbol{\mathit{\Phi}})\boldsymbol{T} = \lambda\boldsymbol{T}$, where $\boldsymbol{D}\boldsymbol{\mathit{\Phi}}$ is the return-weighted observation matrix, $\boldsymbol{D}$ is the diagonal matrix of the weights $\boldsymbol{d}$, λ are the eigenvalues and $\boldsymbol{T}$ the matrix of the eigenvectors.

Algorithm 1 Relative Entropy Policy Search with Return-Weighted Dimensionality Reduction

```
Initialize $\pi_{\widetilde{\phi}}(u|x)$, $\widetilde{\phi} \leftarrow \phi$
Repeat until convergence
    Execute $i = 1 \dots N$ trajectories and collect samples
        Observe state: $X = \left\{ x^{[i]}_{1:H}, \right\}$
        Draw action: $U = \left\{ u^{[i]}_{1:H-1} \right\}$, where $u \sim \pi_{\widetilde{\phi}}(\cdot|x)$
        Store complete features: $\Phi = \left\{ \phi\left( x^{[i]}_{1:H-1} \right) \right\}$
    Solve $g(\eta, V)$ and compute Lagrangians $\eta, V$
    Compute sample weights $d^{[i]} \propto \exp\left( \delta\left( x^{[i]}, u^{[i]}, V \right) / \eta \right)$
    Update feature mapping $\widetilde{\phi} \leftarrow \mathrm{DR}(\Phi, d)$
    Compute reduced features $\widetilde{\Phi} \leftarrow \widetilde{\phi}(\Phi)$
    Update controller $\boldsymbol{\theta} \leftarrow \mathrm{ML}(U, \widetilde{\Phi}, d)$
```

By considering the first n largest eigenvalues λ we obtain the desired mapping. The resulting set of basis functions is orthogonal and uncorrelated, describing as much of the variance in the original space as possible. The choice of n is performed by using the *explained variance* ν, a measure used in statistics to calculate the fraction of variance in the original data accounted for by the low-dimensional model. In the case of PCA, the variance explained by each component is expressed by the eigenvalues λ, i.e., $\nu_i = \lambda_i / \sum_{j=1}^{n} \lambda_j$. Finally, our approach can be extended to kernel PCA [21] by reformulating the problem in a higher-dimensional space using a return-weighted kernel function. The corresponding eigenproblem is $\mathrm{Cov}(\boldsymbol{K_d})\boldsymbol{T} = \lambda \boldsymbol{T}$, where $\boldsymbol{K_d} = \boldsymbol{DK}$ is the return-weighted kernel matrix, $\boldsymbol{K}$ is a matrix of entries $k_{ij} = \kappa(\phi(x_i), \phi(x_j))$, and κ is a kernel.

Algorithm 1 shows the complete procedure. At each iteration the current controller draws new samples by exploiting the features $\widetilde{\phi}$ obtained at the previous step. Subsequently, the samples are used both to update the controller and the feature mapping.

IV. Evaluation

In this section, we evaluate our algorithm and compare its performance in different versions: vanilla (without any DR), preprocessing-aided (we first collect samples to learn a low-dimensional state representation and subsequently we learn the controller), PCA-aided and rwPCA-aided. We also compare with Deep Deterministic Policy Gradient (DDPG) [14], a state-of-the-art deep RL algorithm, and with dimensionality reduction based on conditional mutual information (CMI) [22]. The latter consists of filter-type feature selection that evaluates the independence between return and state-feature sequences using their CMI. At each iteration the features with the highest CMI are selected for learning. For DDPG, we used the non-convolutional network architecture described in the original paper without dropout but with the same hyperparameters. For PCA and rwPCA, the threshold on the explained variance is set to 0.99 in all tasks.

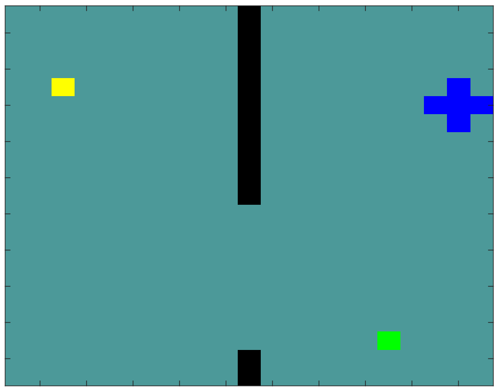


Fig. 3: The pixelworld environment as observed by the agent. It consists in a 21×21 pixel image where the blue cross is the agent, the yellow dot is the coin and the green dot is the goal. The wall (black line) moves along the y-axis with a constant clock time at half of the agent speed, and its hole width is twice the agent size. The agent has to collect the coin and to deliver it to the goal, while not touching the wall.

We evaluate the quality of the learned noise-free policies, i.e., with zero variance, and the sample efficiency. In the first problem, we present a continuous gridworld-like task, where the agent needs to learn optimal actions by receiving only a noisy pixelated representation of the world. In the second problem, we study the application of our method on a simulated robot tetherball game. For each case study, first, the experiments are presented and then the results are reported and discussed. In both problems, Algorithm 1 converges when the KL divergence between the current and the new controller is less than 0.1 or when the maximum number of iterations is reached.

A. Pixelworld

Description. In the pixelworld, shown in Figure 3, the blue agent has to pick the yellow coin and place it on the green goal while avoiding the moving black wall. The agent position is defined in $[0, 1]^2$, its action in $[-0.05, 0.05]^2$. Both the initial agent position and the center of the wall hole are drawn from a uniform distribution. At each time step the agent receives a penalty equal to its distance from the coin (if not collected yet) or from the goal (otherwise). Additionally, it receives a bonus of +1 for collecting the coin and for placing it on the goal, and a penalty of -1 for bumping into the wall or into the environment boundaries. However, the agent does not know the true state of the environment, i.e., the position of itself, the coin, the goal and the wall. Instead, it receives 21×21 pixels images of the environment (Figure 3). Using this representation has two major consequences. First, being the true space continuous, the pixel discretization entails a loss of information, as the agent cannot distinguish between similar states. Second, some pixels, although relevant for representing the environment, are non-informative for the task. For instance, it is irrelevant to identify the whole wall, as knowing just the hole center is sufficient for a full representation of the true state.

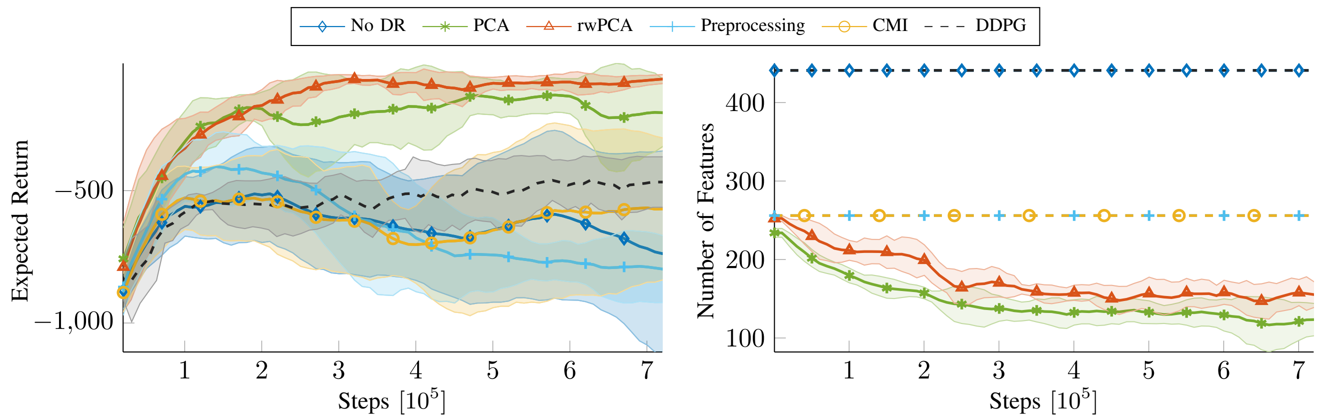


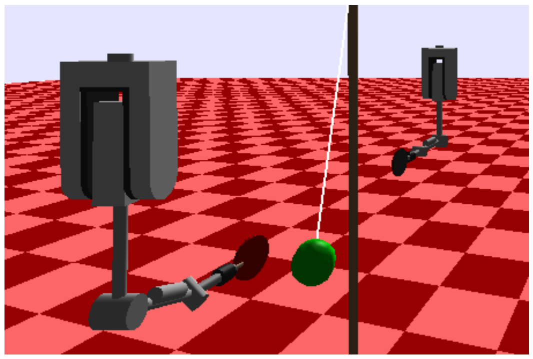
Fig. 4: Pixelworld results averaged over 20 trials (shaded area denotes standard deviation). Left Figure shows that only interleaving PCA and RL leads to high quality policies. The return-weighting step further increases the performance, as rwPCA-aided RWR trend is the most stable and converges quickly to the best controller.

Results. We use a Gaussian controller with diagonal covariance $\pi(u|x) \sim \mathcal{N}(\boldsymbol{k} + \boldsymbol{K}\phi(x), \boldsymbol{\Sigma})$, where $\phi(x)$ are the pixels representing the world. The controller is initialized with zero mean and identity covariance. The evaluation is performed over 1,000 episodes of at most 1,000 steps, while during learning the agent collects only 25 episodes per iteration. Figure 4 shows the results with KL bound $\epsilon = 1$. Not performing any DR or just preprocessing the state representation led to a very poor controller. In the first case, the state dimensionality is too high compared to the few number of samples used for a controller update step. In the second case, the pre-learned state representation is not informative. The preprocessing is performed, in fact, on 500,000 samples collected by the random initial controller. Therefore, such samples do not include many goal-relevant states, since the agent mostly bumps into the wall or into the environment boundaries, thus rarely experiencing positive rewards. As a consequence, the state representation is neither parsimonious nor return-driven. Similarly, CMI-aided RWR achieves poor results, not being able to even collect the coin. The algorithm used 256 features, chosen among the ones with the highest CMI. This dimensionality was chosen as equal to the one learned by preprocessing with PCA. On the contrary, interleaving PCA and RL tremendously improved the learning performance, as both vanilla PCA and rwPCA are able to converge to high quality policies. However, rwPCA outperforms the former both in terms of quality of the final controller and stability of the learning. As already discussed, some pixels, although relevant for the state description, are irrelevant for the agent's goal. Therefore, applying a return-weighted procedure allows the agent to construct features that better support the RL algorithm. Finally, DDPG is not able to solve the task. Instead of collecting the coin and delivering it to the goal, the agent learns to get close to the coin in order to get minimal penalties, but then avoids collecting it in order not to receive the following highly negative rewards (for not being close to the goal). This behavior may be due to DDPG's ineffective 'dithering' exploration in the action space, in contrast to the exploration in the controller space of our algorithm. Another reason may be DDPG's inaccurate value function approximation. For example, even though the agent received episodic returns of about -300, the value function predicted returns of about -60. Similar inaccuracies have been reported by Lillicrap et al. [14], however without severe consequences on the learning.

B. Simulated Robot Tetherball Game

Description. The simulated robot tetherball game [23], shown in Figure 5 (left), is a contextual episodic task characterized by complex system dynamics and high-dimensional actions. An episode is defined by each player turn, which starts when the opponent hits or misses the ball. At the beginning of each episode, the robot plans its trajectory according to its observation of the environment (i.e., the context), consisting in 20 variables: its joints position, the paddle position and orientation, the ball position and estimated velocity and the pivot point of the ball. However, not all variables are relevant, e.g., the paddle position and orientation can be determined by the robot joints position. For details of the reward function we refer the reader to the original paper.

Results. The robot is controlled by dynamic motor primitives [24] for a total of 24 parameters. These parameters are drawn by a Gaussian distribution π, initialized by imitation learning on 20 demonstrations in random context. Note that since DDPG performs exploration in the action space, it is not applicable to this task. Similarly to the original work, the KL divergence bound is 0.9 and at each iteration the agent collects 50 new trajectories and keeps track of the last 450. As shown in Figure 5 (right), rwPCA-aided REPS once again achieves the best results, reducing the context space to eight features on average. Vanilla PCA-aided version reduces the space to the same dimension, but is not able to attain the same results. Preprocessing the space by PCA on 10,000 samples led to a 18 dimensional space, which did not help the player. CMI, which exploits 13 features, fails as well.



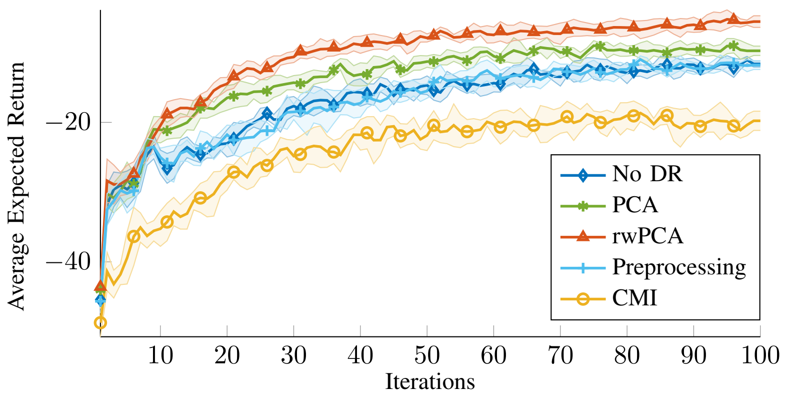


Fig. 5: In the tetherball game (left) one robot has to hit a ball hanging from a pole without giving the opponent the chance to unwind it. On the right, the results averaged over ten trials. rwPCA outperforms all other algorithms, training the agent to return the ball 85% of the time. With the second best policy, learned by PCA, the robot achieves only 75% of accuracy.

V. Conclusion

Learning a parsimonious and informative state representation is crucial for RL. Methods to date usually rely either on filter-type preprocessing procedures, without considering a more structured integration of DR into RL, or on computationally and samples demanding deep networks to handle high-dimensional spaces. With this paper, we studied the integration of DR into RL and presented a method for helping an agent to learn both a low-dimensional state representation altogether with an optimal controller. We provided a method able to perform *return-based* feature construction *without relying on the value function to select the optimal action* and without requiring to fix the dimensionality of the constructed space representation. Even though the explained variance has to be fixed, its choice is usually much easier than choosing the number of features a priori. Evaluated on two problems, our method outperformed both classical and state-of-the-art deep learning approaches both in terms of sample efficiency and quality of the final controller, and it was successful in the presence of both high-dimensional state and action spaces.

Our method opens several avenues of research and real world applications, as it can be easily applied to systems relying on high sensory input (e.g., real robotic or vision problems). Real robot tetherball would be a suitable application, as vision data can be easily added to the context, as well as features describing the opponent state. In follow-up work, we will also study the integration of more complex controller representations, e.g., deep networks [14], and more sophisticated exploration strategies, e.g., entropy-bounded [25].

References

[1] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics." *Foundations and Trends in Robotics*, 2013.
[2] S. Levine and V. Koltun, "Guided policy search," in *ICML*, 2013.
[3] S. Bitzer, M. Howard, and S. Vijayakumar, "Using dimensionality reduction to exploit constraints in reinforcement learning," in *IROS*, 2010.
[4] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *IJCNN*, 2010.
[5] I. Menache, S. Mannor, and N. Shimkin, "Basis function adaptation in temporal difference reinforcement learning," *Annals of Operations Research*, 2005.
[6] P. W. Keller, S. Mannor, and D. Precup, "Automatic basis function construction for approximate dynamic programming and reinforcement learning," in *ICML*, 2006.
[7] J. Z. Kolter and A. Y. Ng, "Regularization and feature selection in least-squares temporal difference learning," in *ICML*, 2009.
[8] A. Farahmand, M. Ghavamzadeh, C. Szepesvári, and S. Mannor, "Regularized fitted Q-iteration for planning in continuous-space markovian decision problems," in *AAC*, 2009.
[9] M. Ghavamzadeh, A. Lazaric, O. Maillard, and R. Munos, "LSTD with random projections," in *NIPS*, 2010.
[10] Y. Sun, M. Ring, J. Schmidhuber, and F. J. Gomez, "Incremental basis construction from temporal difference error," in *ICML*, 2011.
[11] C. Painter-Wakefield and R. Parr, "Greedy algorithms for sparse reinforcement learning," in *ICML*, 2012.
[12] M. M. Fard, Y. Grinberg, A. Massoud Farahmand, J. Pineau, and D. Precup, "Bellman error based feature generation using random projections on sparse spaces," in *NIPS*, 2013.
[13] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *ICML*, D. Blei and F. Bach, Eds., 2015.
[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR*, 2015.
[15] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-Learning with Model-based acceleration," in *ICML*, 2016.
[16] D. P. Bertsekas, "Dynamic programming and suboptimal control: A survey from ADP to MPC*," *European Journal of Control*, 2005.
[17] M. Toussaint and A. Storkey, "Probabilistic inference for solving discrete and continuous state markov decision processes," in *ICML*, 2006.
[18] J. Peters, K. Muelling, and Y. Altun, "Relative entropy policy search," in *AAAI*, 2010.
[19] H. H. Yue and M. Tomoyasu, "Weighted principal component analysis and its applications to improve fdc performance," in *CDC*, 2004.
[20] F. Agahian, S. A. Amirshahi, and S. H. Amirshahi, "Reconstruction of reflectance spectra using weighted principal component analysis," *Color Research & Application*, 2008.
[21] Q. Jiang and X. Yan, "Weighted kernel principal component analysis based on probability density estimation and moving window and its application in nonlinear chemical process monitoring," *Chemometrics and Intelligent Laboratory Systems*, 2013.
[22] H. Hachiya and M. Sugiyama, "Feature selection for reinforcement learning: Evaluating implicit state-reward dependency via conditional mutual information," in *ECML/PKDD*. Springer, 2010.
[23] S. Parisi, H. Abdulsamad, A. Paraschos, C. Daniel, and J. Peters, "Reinforcement learning vs human programming in tetherball robot games," in *IROS*, 2015.
[24] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *NIPS*, 2002.
[25] R. Akrour, A. Abdolmaleki, H. Abdulsamad, and G. Neumann, "Model-free trajectory optimization for reinforcement learning," in *ICML*, 2016.