Regularizing Reinforcement Learning with State Abstraction

Riad Akrour[1], Filipe Veiga[1], Jan Peters[1,2] and Gerhard Neumann[1,3]

***Abstract*— State abstraction in a discrete reinforcement learning setting clusters states sharing a similar optimal action to yield an easier to solve decision process. In this paper, we generalize the concept of state abstraction to continuous action reinforcement learning by defining an abstract state as a state cluster over which a near-optimal policy of simple shape exists. We propose a hierarchical reinforcement learning algorithm that is able to simultaneously find the state space clustering and the optimal sub-policies in each cluster. The main advantage of the proposed framework is to provide a straightforward way of regularizing reinforcement learning by controlling the behavioral complexity of the learned policy. We apply our algorithm on several benchmark tasks and a robot tactile manipulation task and show that we can match state-of-the-art deep reinforcement learning performance by combining a small number of linear policies.**

I. Introduction

Abstraction in reinforcement learning (RL) has long been seen as a promising direction to combat the curse of dimensionality by exploiting the structure of the problem at hand [1], [2], [3]. Its application to RL takes two forms: temporal abstraction [4], [5], [6], [7], [8] and state abstraction [9], [10], [11]. Temporal abstraction augments the *ground* action space with temporally extended actions (actions lasting more than a single time-step) allowing the agent to reason on a less granular time-scale and reducing the number of decisions to optimize for. State abstraction exploits the structure of a Markov Decision Process (MDP) to derive an *abstract* MDP having a compressed state space but a similar optimal policy to that of the ground MDP.

As computational power significantly increased in the last decade, exploiting structure to increase sample efficiency has been outweighed by the appeal of hand-free algorithms that are able to generalize to several problems without relying on expertly crafted representations, sub-policies or to a lesser degree hand tuned hyper-parameters [12], [13], [14]. Adding structure to the portfolio of deep RL algorithms through a hierarchical reinforcement learning (HRL) decomposition should not be at the expense of generality and sub-policies should emerge from data rather than be handcrafted for every problem. Unfortunately, when the sub-policies are not hand-crafted but are learned concurrently to the upper level policy, it was shown that HRL is not always more efficient than flat RL in theory [15], [16], [17] and is rarely so in practice [18]. In this paper, we will not use HRL to improve sample efficiency of the learning process but as a mean of controlling the complexity of the learned policy.

[1]CLAS/IAS, TU Darmstadt, 64289 Darmstadt, Germany
[2]Max Planck Institute for Intelligent Systems, 72070 Tbingen, Germany
[3]L-CAS, University of Lincoln, Lincoln, United Kingdom.
Correspondence to Riad Akrour, riad@robot-learning.de

Our work can be seen as an extension of state abstraction. Structure in state abstraction emerges by grouping together several states from the ground MDP into an abstract state. Several criteria were studied to group states together in the exact case where the optimal policies of the abstract and ground MDPs coincide [9] and the approximate case [10] where the optimal policy of the abstract MDP is only near-optimal in the ground MDP. A policy defined on the abstract MDP selects a unique action for all the ground states sharing the same abstract representation. In this paper we extend state abstraction by assuming that the abstract policy does not assign a unique action to all the ground states but more generally defines a policy of simple shape for each of the abstract states.

The contribution of this paper is to propose a framework that solves both the abstraction problem (clustering the state space) and the learning of appropriate sub-policies for each cluster. The used definition of a state cluster is a set of states over which a policy of simple shape can be near-optimal. As neither the sub-policies nor the optimal Q-Value is known in advance, it becomes clear that the two problems need to be solved concurrently. As such we introduce a hierarchical decomposition of the policy and the state space clustering will be implicitly provided by the upper level policy. Section II provides an overview of HRL and relates our HRL decomposition to those already introduced in the literature.

Section V provides an empirical evaluation of the proposed algorithm on continuous control reinforcement learning benchmarks and a simulated dexterous manipulation task. The proposed method is well suited for robotics applications since the resulting policy will be a mixture of simple primitives that can be more easily analyzed than a black-box non-linear function; in the prospect of providing safety guarantees when the policy is executed on a physical system. Additionally, the hierarchical structure provides an easy way of controlling the behavioral complexity of the returned policy by modifying the functional class of the primitives (e.g. polynomial in the states) or their number. The main take away message from our experimental section is that very competent policies, performing on par with deep RL policies, can be learned on high dimensional control tasks by switching between a small number—typically less than ten—linear policies.

II. Related Work

There are several hierarchical decompositions of RL problems in the litterature. Perhaps the most ubiquitous decomposition is the option framework [1], [16], [7], [8] which is

based on the call-and-return paradigm where an upper level policy selects an option and waits until its termination to select the next option. The sub-policy associated with an option can be hand-defined [19], learned from trajectory data [7] or learned by interaction with the environment [8]. In the latter case, while theoretical convergence results are long known [1], the benefits of temporally extended actions on sample efficiency are not guaranteed.

For instance, [16] showed that convergence speed can be accelerated by using temporally extended actions only if some conditions are satisfied such as the expected duration of executed options being sufficiently long. As such, we do not focus in this paper on sample complexity but rather on behavioral complexity and provide an automated way of answering the question: what is the best performance one can achieve with the combination of a given number of linear policies. [20] proposed the linear option framework, but sub-policies are linear in features of the state and not the state directly. In this paper, we show that linear in state policies are indeed sufficient for many control problems if an appropriate partitioning of the state space is learned. Moreover, unlike [20] we do not consider termination functions for each option. In our setting a sub-policy is terminated as soon as the next state exits the current state cluster. This HRL decomposition is thus closer to state abstraction.

State abstraction [15], [9], [10] usually consists in the application of two distinct steps: i) building the abstract MDP and ii) finding the optimal policy in the abstract MDP. The abstract MDP is simply a partitioning (i.e. clustering) of the state space. In [9], [10], four criteria to create the state space clustering are presented. However, all of these criteria are hard to use in practice as they often require full knowledge of the transition probabilities or the optimal Q-Function. A preliminary work in the literature [21] has investigated the discovery of abstract states in the absence of transition model, but the behaviour of such partitioning within a full reinforcement learning framework remains unclear. We propose instead in this paper to perform the partitioning and learning of optimal policy simultaneously in a hierarchical reinforcement learning (HRL) framework, where the state partitioning is provided by the gating policy (upper level policy).

Perhaps the most related work to ours is the Adapted Skills Adapted Partitions algorithm [11]. In [11], a hierarchical policy chooses an action according to a mixture of parametric policies. The weighting of the mixture depends on a set of parameterized hyper-planes splitting the state space. The main differentiating factor is that their hyper-plane is linear in some hand-crafted features whereas we use a neural network to learn the state space partitioning. Note that, albeit we do not consider time extended actions, our algorithm can be straightforwardly extended to this setting if the sub-policies output parameters of primitives—such as the goal position of a Dynamical Movement Primitive [22]—instead of a single action, similar to the hierarchical settings in [4], [6], [23].

III. Generalized state abstraction

The main algorithmic contribution of the paper is to propose a practical way of performing state abstraction. To do so, we phrase the state space partitioning and the subsequent reinforcement learning in the abstracted MDP as a single learning problem. Specifically, we solve a reinforcement learning problem on an extended action space where the policy has to provide for a given state both its cluster identifier and a primitive action. Within this formulation RL is performing exploration in order to find both the optimal clustering and their associated (sub-)policies. In the following we provide the notations used throughout the paper in Section III-A, formalize our reinforcement learning reduction of state abstraction in Section III-B and provide implementation details for solving the reinforcement learning problem in Section IV.

A. Notations

We consider tasks framed as discounted Markov Decision Processes (MDP) defined by the quintuple $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$ where $\mathcal{S} \subset \mathbb{R}^{\mathrm{d}_s}$ is a state space, $\mathcal{A} \subset \mathbb{R}^{\mathrm{d}_a}$ the action space, $P(s_{t+1}|s_t, a_t)$ the transition probability to state s_{t+1} upon the execution of action a_t in s_t and $R(s_t, a_t)$ the associated reward. A stochastic policy π gives a probability $\pi_t(a|s)$ of executing $a \in \mathcal{A}$ in $s \in \mathcal{S}$. Our goal is to find the policy maximizing the policy return

$$J(\pi) = \mathbb{E}_{s_0, a_0, \dots}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right],$$

where the expectation is taken w.r.t. all random variables s_t and a_t. We will additionally rely on the usual quantities; the Q-function

$$Q_\pi(s, a) = \mathbb{E}_{s_0, a_0, \dots}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \;\middle|\; s_0, a_0 = s, a\right]$$

denoting the expected cumulative discounted reward of performing action a in state s under policy π and $A_\pi(s, a) = Q_\pi(s, a) - \mathbb{E}_{a' \sim \pi} Q_\pi(s, a')$ the advantage function giving the difference between the Q-function and the value function.

To solve the generalized state abstraction problem we introduce an augmented MDP given by the tuple $(\mathcal{S}, \mathcal{A}', R', P', \gamma)$. The main difference compared to the initial MDP lay in the augmented action space $\mathcal{A}' = \mathcal{A} \times \mathcal{C}$, where $\mathcal{C} = \{1, \dots, K\}$ is a set of K cluster identifiers; while both R' and P' can be rewritten in term of their counterpart R and P of the original MDP by simply ignoring the cluster identifier. For example, the reward in the augmented MDP $R'(s, \{a, c\})$ is equal to $R(s, a)$, the reward of the state-action pair (s, a), ignoring the cluster identifier $c \in \mathcal{C}$.

B. RL reduction of state abstraction

State abstraction consists in clustering the state space such that a policy defined on the state space partition (i.e. a policy selecting the same action for any state in a given cluster) remains optimal [9] or near-optimal [10]. We propose to learn the state clustering concurrently to learning the

action to perform by reinforcement learning in the previously introduced augmented MDP. The policy decomposition in the augmented MDP will be given by

$$\pi(\{a, c\}|s) = \pi_p(c|s)\pi_c(a),$$

where π_p is the partitioning policy and π_c is the sub-policy in cluster $c \in \mathcal{C}$. Note that both π_p and the sub-policies π_c are stochastic policies. As the initial partitioning policy $\pi_p(c|s)$ (which is a discrete action policy implemented using a softmax distribution and decaying uniform exploration as will be discussed in Sec. IV-A) starts with a high amount of exploration, π_p only provides a soft partitioning of the state space. However, as exploration decreases and π converges to a deterministic policy, π_p will converge to a hard partitioning of the state space where the action executed in each cluster $c \in \mathcal{C}$ is given by π_c.

The main limitation of this clustering procedure compared to standard state abstraction clusterings [9], [10] is that the number of clusters is fixed a priori. While in approximate state abstraction [10] the smallest number of clusters achieving ϵ-optimal performance is sought, our practical state abstraction implementation provides the best achievable performance with K clusters.

Note that this reinforcement learning formulation of state abstraction is different from action discretization where a continuous space is discretized into a finite set of actions a priori. Here, the RL algorithm continually explores cluster-action associations by virtue of the stochastic policies until convergence.

C. Generalized state abstraction with linear sub-policies

Performing the same action for all the cluster's states might be limiting and could result in non-smooth behaviors due to discontinuities when transitioning between state clusters. In order to increase the expressiveness of the policy while keeping the ability of adjusting the behavioral complexity, we generalize the notion of state abstraction to finding sub-policies of simple shape $\pi_c(a|s)$—encompassing the constant policy in commonly known state abstraction.

The policy on the augmented action space becomes $\pi(\{a, c\}|s) = \pi_p(c|s)\pi_c(a|s)$ where we choose the sub-policy $\pi_c(a|s)$ to be a linear in state Gaussian distribution, $\pi_c(a|s) = \mathcal{N}(a|Ls + b, \Sigma_{\text{diag}})$, with L a $\mathrm{d}_a \times \mathrm{d}_s$ matrix, b a d_a dimensional vector and $\Sigma_{\text{diag}} = \text{diag}(\sigma_1^2, \ldots, \sigma_{\mathrm{d}_a}^2)$ a diagonal variance matrix. This decomposition lays on the complexity axis between the original state abstraction formulation $\pi_c(a|s) = \pi_c(a)$ that gives the same action to all the states in a cluster on one extreme, and on the other extreme to 'flat' reinforcement learning with non-linear policies (e.g. neural networks) that can have large fluctuations in the actions of neighboring states.

IV. Solving the RL reduction

Because of the limited complexity of the sub-policies, solving the joint problem of clustering the state space and learning the linear sub-policies is usually harder than learning a single flat policy with the same number of parameters. We found that the main reason of convergence to sub-optimal solutions was a too fast reduction of exploration. In the following we detail the exploration strategy for both the lower level policies π_c and the partitioning policy π_p used in order to limit the risk of premature convergence. Subsequently, the initialization procedure of the sub-policies is presented before concluding the section by discussing the RL algorithm used for both our approach and the baseline 'flat' RL in the experimental section (Section V).

A. Exploration

Exploration is still a largely open research topic in reinforcement learning. A badly tuned initial exploration parameter or an uncontrolled reduction of exploration can be the cause of premature convergence to sub-optimal solutions. The importance of exploration is twofold in our setting. Not only exploration at the action level needs to be preserved; but a premature convergence to a badly partitioned state space presents an insurmountable challenge for the simple linear sub-policies. As such, we adopt a cautious approach for handling exploration noise in order to reduce variation across runs. Exploration control of Gaussian policies—such as our sub-policies π_c—in the literature is ensured by the KL update constraint [13] or by adding a term favoring higher entropy policies [24], [25]. The latter approach however requires to manually set a trade-off parameter between reward and entropy maximization while the former couples the change of Σ_{diag} to that of the mean (i.e. a smaller KL has the desired effect of limiting exploration loss but additionally limits the change of the policy mean). To decouple exploration control and update rate of the sub-policies we impose a lower bound $\sigma_{\min}$ to each diagonal entry and let the reinforcement learning algorithm (Section IV-C) optimize for Σ_{diag} with the additional constraint that $\sigma_i > \sigma_{\min}$ for all $i \in \{1, \ldots, \mathrm{d}_a\}$. After each iteration $\sigma_{\min}$ is decreased by a constant factor.

For the partitioning policy π_p, having a finite action space $\mathcal{C}$, we use the softmax distribution with logits outputted by a neural network to perform exploration. To reproduce a similar exploration scheme to that of Gaussian policies, one would lower bound the temperature (instead of $\sigma_{\min}$)

Algorithm 1 Generalized State Abstraction

1: **Input:** Cluster count K and neural network structure for the state space partitioning
2: Initialize sub-policies (Section IV-B)
3: **repeat**
4: Generate trajectories $(s_0, \{a_0, c_0\}, r_0, s_1 \ldots, s_T, \{a_T, c_T\}, r_T)$
5: Learn the V-Function (Section IV-C)
6: Update the probabilistic partitioning π_p and linear sub-policies π_c (Section IV-C)
7: Decrease exploration lower-bounds (Section IV-A)
8: **until** Iteration limit reached
9: **return** State space partitioning π_p and linear sub-policies π_c

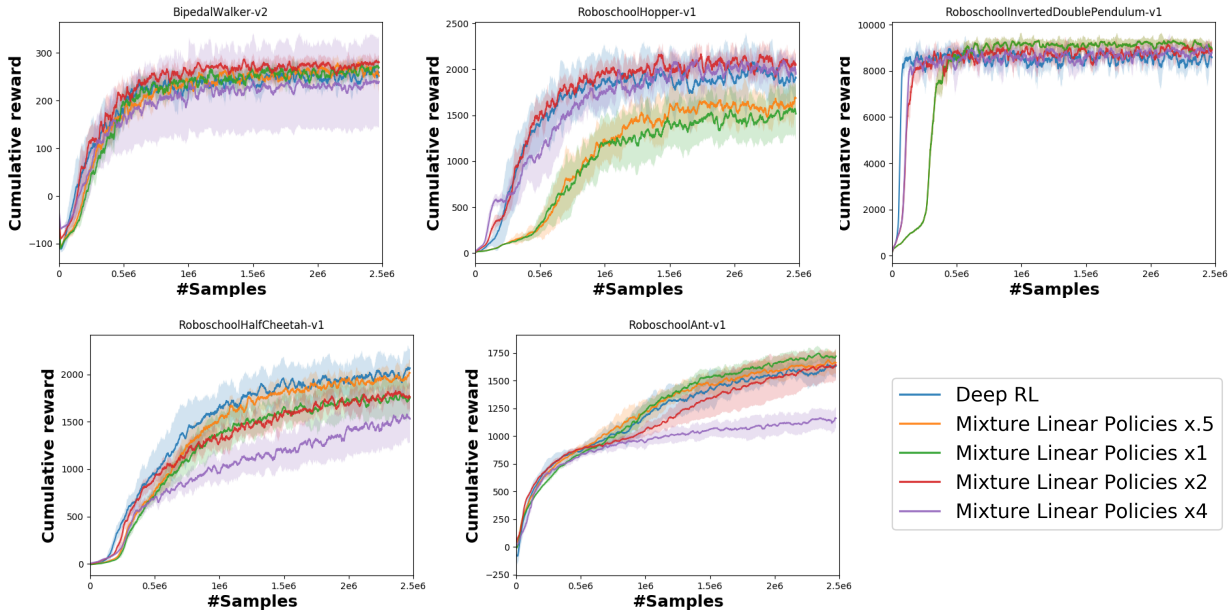

Fig. 1: Comparison of Deep RL to mixture of linear policies (with varying number of linear policies) on five environments from OpenAI's gym [26]. The number of linear policies is a multiple of each task's action space dimensionality. In most of the tasks we can learn policies on par with deep reinforcement learning using a small number of sub-policies (half of the action space dimensionality) except on the Hopper where at least six linear policies are required to solve the task. Plots are averaged over seven independent runs.

such that the entropy remains over a threshold, and have the threshold decrease every iteration. Unfortunately, no closed form expression giving such target temperature exists. To unsure sufficient exploration we revert to standard ϵ-exploration by adding a probability ϵ of selecting an action uniformly at random and have ϵ decrease at each iteration.

B. Sub-policy initialization

Having diversity in the initial sub-policies accelerates the clustering of the state space and the learning of specialized sub-policies. We increase diversity by spreading the location of each sub-policies' linear-Gaussian bias b within the action space which is typically of the form $[-1, 1]^{\mathrm{d}_a}$. Additionally, the initial exploration noise Σ_{diag} is scaled inversely proportional to the number of cluster K. The rational behind such a decision is that the higher the cluster count K is, the more densely the set of b will populate $[-1, 1]^{\mathrm{d}_a}$ and the less variance the sub-policies need to cover the action space, while avoiding overlap to favor specialization. Matrices L of the linear-Gaussian sub-policies are initialized to zero.

C. Base RL algorithm

Our reduction of state abstraction is not specific to a reinforcement learning algorithm. The action space of our augmented MDP mixes both discrete decisions for selecting a cluster and continuous actions given by the associated sub-policy. Because of the mixed nature of the augmented action space we used as our base RL learner the Proximal Policy Optimization (PPO) algorithm as it demonstrated state-of-the-art empirical performance on both high dimensional discrete and continuous action problems [27]. The state-of-the-art performance of PPO makes it a representative baseline of 'flat' deep RL. PPO is an approximate policy iteration algorithm [28], alternating between policy evaluation and policy update. Letting q denote the current policy, the policy evaluation step consists in evaluating an advantage function $A_q(s, a)$. We compute the advantage function from the value function as described in [27], and use [29] to compute the value function. The policy update step returns a new policy π by maximizing an objective function $L^{\text{PPO}}(\pi; q)$ given by

$$L^{\text{PPO}}(\pi; q) = \mathbb{E}_{s,a\sim q}\Big[\min\big(I(a,s)A_q(s,a), \\ \mathrm{c}\left(I(a,s),\epsilon\right)A_q(s,a)\big)\Big],$$

where $\mathrm{c}\left(I(a,s),\epsilon\right) = \max(\min(I(a,s)-1,\epsilon),-\epsilon)$ clips $I(a,s)$ to the interval $[1-\epsilon, 1+\epsilon]$ and $I(s,a) = \frac{\pi(a|s)}{q(a|s)}$ is short for the importance sampling ratio between π and q.

In addition to its competitive empirical performance, the advantage of using PPO is that its policy update only depends on the action probability. Hence, the policy update of our state abstraction reduction is fully defined from the decomposition of the policy given in Section III-C, and the bounding of the exploration noise as discussed in Section IV-A is done for the sub-policies by fixing the standard deviation to $\max(\sigma_i, \sigma_{\min})$ for all $i \in \{1, \ldots, \mathrm{d}_a\}$ during the maximization of L^{PPO}.

Algorithm 1 synthesizes our approach to generalized state abstraction (Section III). First we initialize the sub-policies by spreading them over the action space, accelerating their specialization. We then generate trajectories by sampling

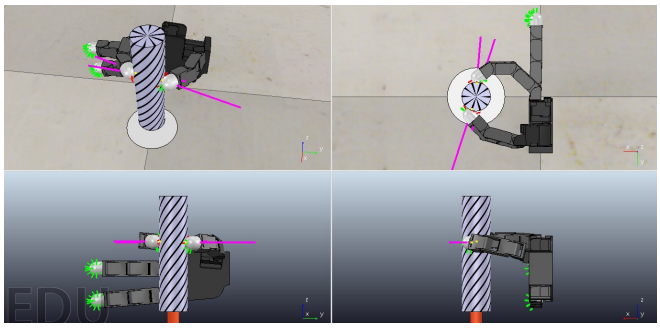

Fig. 2: Multiple views of the simulated Allegro Hand in the V-REP simulator. The task consists of having the cylinder rotate counter clockwise as much as possible during the episode. Each episode starts with the index and thumb in contact with the cylinder and only these two fingers are enabled during the task execution.

cluster identifiers (soft-max distribution given by the neural network) and actions (from the associated linear-Gaussian sub-policy). Then we proceed by evaluating the value function from the generated data and updating the policy while ensuring that exploration is not reduced too fast. After sufficiently many iterations, the algorithm returns a partitioning of the state space and a linear-Gaussian policy associated to each of the state partition, maximizing the expected return.

V. Experiments

We evaluate the performance of our generalized state abstraction algorithm on five environments from OpenAI's gym [26] and a simulated dexterous manipulation task. The research questions we attempt answering in this section are: i) can our framework learn mixture of linear policies competitive with state-of-the-art deep RL algorithms, and ii) what is the order of magnitude for the number of required linear policies to mimic the behavior of a neural network on each of these continuous action tasks.

In all of the experiments we adjust the number of sub-policies to the dimensionality of the action space. The underlying assumption is that the complexity of the task is correlated with the dimensionality of the action space and harder tasks will require a higher number of linear sub-policies. We refer to our framework as "mixture of linear policies" and the multipliers in e.g. Figure 3 are define by $\frac{K}{\mathrm{d}_a}$. For the dexterous manipulation task in Section V-B, since $\mathrm{d}_a = 8$ then a multiplier of .5 indicates that there are 4 sub-policies.

A. RL benchmarks

We evaluate our algorithm on five RL benchmarks with action spaces ranging from $\mathrm{d}_a = 1$ for the inverted double pendulum environment to $\mathrm{d}_a = 8$ for the Ant environment. Figure 1 compares flat deep RL, represented by the PPO algorithm learning a two hidden layers of size 64 neural network policy, to our mixture of linear policies. The results show that most of the tasks can be learned with a number of sub-policies that is half of the action space dimensionality. This is however not true for the Hopper environment where at least six linear policies are required to solve the task. Note that when the number of sub-policies is four times the action spaces, it usually results in slowed down performance. This might be due to each sub-policy having less samples to perform the update or the policy might be limited by the neural network structure as we do not alter the structure of the partitioning policy. As a result, on the Ant environment where the mixture of linear policies with four times the action spaces has its worse relative performance, the neural network output is 32 dimensional while the hidden layers are only 64 dimensional.

It was already shown in [31] that some of these tasks can be learned using a linear policy while other tasks required RBF features. We extend these results by showing that with our hierarchical structure we can match deep RL performance only by switching between a small number of linear policies.

B. Dexterous manipulation

To compare the two approaches on a complex robotic task, we use the V-REP simulator (Coppelia Robotics) to simulate a dexterous in-hand manipulation task. The task consists of having an Allegro hand maximizing the rotation of a cylinder in the counter clockwise direction for the duration of the episode. The simulated robot can be seen in Figure 2.

The Allegro Hand (Wonik Robotics), is a four fingered hand with four joints per finger, for a total of 16 actuated degrees of freedom. The hand is simulated as being controlled by a PD controller on the joint positions, running at a frequency of 100 Hz. Tactile sensor arrays, each with 18 sensing elements, are projected onto each of the four fingers for a total of 72 tactile sensing elements. Each element

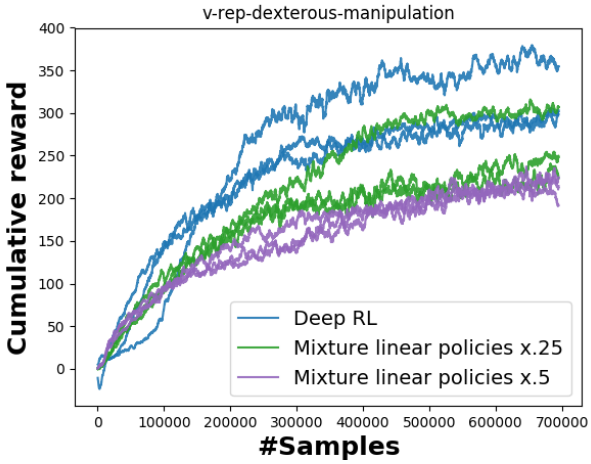

Fig. 3: Comparison of Deep RL to mixture of linear policies (with 2 and 4 linear policies) on the simulated dexterous manipulation task. The mixture of linear policies is able to learn good behaviors with only two linear policies. However, the performance gap to neural network policies is more pronounced than on the Gym environments.

measures the force applied locally.

During the simulation, the task state is given by the eight joint positions of the index and thumb and by the average force captured by the sensing elements of each finger for a total of ten state variables. The actions consist of small perturbations to the current joint positions of the two fingers. The reward is the angular velocity of the cylinder at each time step, being positive or negative respectively for counter clockwise and clockwise velocities. There is also a small penalty on the actions taken by the robot.

Figure 3 gives the cumulative reward achieved by both PPO learning a neural network (same setting as in Section V-A) and the mixture of linear policies. The figure shows that good policies can be learned using only two linear policies although the difference to a neural network policy is more pronounced in this task than on the Gym environments.

VI. Conclusion

We have proposed in this paper an algorithm for partitioning the state space and learning linear sub-policies in each cluster. We have shown that on several continuous action control tasks, the mixture of linear policies can learn similar behaviors than those of a neural network policy with a small number of linear policies. The number of linear policies emulating a given neural network provides insight into the complexity of the learned behavior. However, additional insights should be gained on the regularity of the neural network state space clustering in order to achieve a complete understanding of the learned behavior and provide a human understandable description of it.

Acknowledgments

The research leading to these results has received funding from the DFG Project LearnRobotS under the SPP 1527 Autonomous Learning, from the Intel Corporation, and from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 640554 (SKILLS4ROBOTS). Computing time for the experiments was granted from Lichtenberg cluster.

References

[1] D. Precup, R. S. Sutton, and S. P. Singh, "Theoretical results on reinforcement learning with temporally abstract options," in *European Conference on Machine Learning (ECML)*, 1998, pp. 382–393.

[2] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Neural Information Processing Systems (NIPS)*, 1999.

[3] T. G. Dietterich, "State abstraction in maxq hierarchical reinforcement learning," in *Neural Information Processing Systems (NIPS)*, 1999, pp. 994–1000.

[4] G. Konidaris and A. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," in *Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 1015–1023.

[5] B. da Silva, G. Konidaris, and A. Barto, "Learning Parameterized Skills," in *International Conference on Machine Learning (ICML)*, 2012.

[6] M. J. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," *CoRR*, vol. abs/1511.04143, 2015.

[7] C. Daniel, H. van Hoof, J. Peters, and G. Neumann, "Probabilistic inference for determining options in reinforcement learning," *Machine Learning*, vol. 104, no. 2-3, pp. 337–357, 2016.

[8] P. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Conference on Artificial Intelligence (AAAI)*, 2017, pp. 1726–1734.

[9] L. Li, T. J. Walsh, and M. L. Littman, "Towards a unified theory of state abstraction for mdps," in *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2006.

[10] D. Abel, D. E. Hershkowitz, and M. L. Littman, "Near optimal behavior via approximate state abstraction," in *International Conference on Machine Learning (ICML)*, 2016, pp. 2915–2923.

[11] D. J. Mankowitz, T. A. Mann, and S. Mannor, "Adaptive skills adaptive partitions (ASAP)," in *Neural Information Processing Systems (NIPS)*, 2016, pp. 1588–1596.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015.

[13] J. Schulman, S. Levine, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *International Conference on Machine Learning (ICML)*, p. 16, 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," *CoRR*, 2015.

[15] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. L. Dean, and C. Boutilier, "Hierarchical solution of markov decision processes using macro-actions," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1998, pp. 220–229.

[16] T. Mann and S. Mannor, "Scaling up approximate value iteration with options: Better policies with fewer iterations," in *International Conference on Machine Learning (ICML)*, 2014, pp. 127–135.

[17] R. Fruit and A. Lazaric, "Exploration-exploitation in mdps with options," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[18] N. K. Jong, T. Hester, and P. Stone, "The utility of temporal abstraction in reinforcement learning," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008, pp. 299–306.

[19] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Neural Information Processing Systems (NIPS)*, 2016, pp. 3675–3683.

[20] J. Sorg and S. P. Singh, "Linear options," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010, pp. 31–38.

[21] R. Krishnamurthy, A. S. Lakshminarayanan, P. Kumar, and B. Ravindran, "Hierarchical reinforcement learning using spatio-temporal abstractions and deep neural networks," *CoRR*, vol. abs/1605.05359, 2016.

[22] A. Ijspeert and S. Schaal, "Learning Attractor Landscapes for Learning Motor Primitives," in *Advances in Neural Information Processing Systems (NIPS)*, ser. (NIPS). Cambridge, MA: MIT Press, 2003.

[23] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *Conference on Artificial Intelligence (AAAI)*, 2016, pp. 1934–1940.

[24] R. J. Williams and J. Peng, "Function optimization using connectionist reinforcement learning algorithms," *Connection Science*, 1991.

[25] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2016.

[26] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[28] C. Szepesvari, *Algorithms for Reinforcement Learning*. Morgan & Claypool, 2010.

[29] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Neural Information Processing Systems (NIPS)*, 2016, pp. 1046–1054.

[30] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *International Conference on Machine Learning (ICML)*, 2017.

[31] A. Rajeswaran, K. Lowrey, E. Todorov, and S. M. Kakade, "Towards generalization and simplicity in continuous control," in *Conference on Neural Information Processing Systems (NIPS)*, 2017.