Policy Reuse for Transfer Learning Across Tasks with Different State and Action Spaces

Fernando Fernández FFERNAND@INF.UC3M.ES
Universidad Carlos III de Madrid, Avda. de la Universidad 30, 28911 Leganés, Madrid, Spain

Manuela Veloso VELOSO@CS.CMU.EDU
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA

Abstract

Policy Reuse is a reinforcement learning method in which learned policies are saved and reused in similar tasks. The policy reuse learner extends its exploration to probabilistically include the exploitation of past policies, with the outcome of significantly improving its learning efficiency. In this paper we demonstrate that Policy Reuse can be applied for transfer learning among tasks in different domains by defining and using a mapping between their state and action spaces. We use the Keepaway domain where we show that Policy Reuse can outperform the results of basic learning.

1. Introduction

Reinforcement Learning (RL) (Kaelbling et al., 1996) is a powerful control learning technique based on a trial and error process guided by reward signals received from the environment. Classical RL algorithms require an intense exploration of the action and state spaces. To reduce the exploration several methods rely on the appealing idea of reusing the knowledge acquired in one learning process to solve other problems, including the transfer of value functions (Taylor & Stone, 2005) or the reuse of options (Sutton et al., 1999).

Policy Reuse is a technique where the learner is guided by past policies balancing among three choices: the exploitation of the ongoing learned policy, the exploration of new random actions, and the exploitation of past policies. Policy Reuse contributes an exploration strategy able to probabilistically bias the exploration of the domain with a predefined past policy; and a similarity metric that allows the estimation of the similarity of past policies with respect to a new one. Policy Reuse has been demonstrated in a serie of discrete grid-based learning tasks where the efficiency of the learner significantly improves when reusing past policies described in the same action and state space (Fernández & Veloso, 2006).

In this paper we extend Policy Reuse to transfer knowledge between continuous tasks with different state and/or action spaces. We apply the transfer learning using Policy Reuse to the Keepaway domain (Stone et al., 2005). Policy Reuse favorably compares with other transfer learning methods that have been recently applied (Taylor & Stone, 2005) to this same domain.

Appearing in *Proceedings of the ICML'06 Workshop on Structural Knowledge Transfer for ML*, Pittsburgh, PA, 2006.

2. Policy Reuse

Policy Reuse focuses on Reinforcement Learning domains where different *tasks* can be solved. It introduces a task as a specific reward function, $\mathcal{R}$, while the state space, $\mathcal{S}$, the action space, $\mathcal{A}$, and the transition function, $\mathcal{T}$, stay constant for all the tasks. Thus, we extend the concept of an MDP by introducing two new concepts: domain and task. Policy Reuse characterizes a domain, $\mathcal{D}$, as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T} >$. It defines a task, Ω, as a tuple $< \mathcal{D}, \mathcal{R}_\Omega >$, where $\mathcal{D}$ is a domain as defined before, and $\mathcal{R}_\Omega$ is the stochastic and unknown reward function.

The learning objective is to maximize the expected average reinforcement per episode, say W, defined as $W = \frac{1}{K} \sum_{k=0}^{K} \sum_{h=0}^{H} \gamma^h r_{k,h}$, where K is the total number of episodes, H is the maximumn number of steps per episode, $r_{k,h}$ is the immediate reward obtained in the step h of the episode k, and γ ($0 \leq \gamma \leq 1$) discounts future rewards. An action policy, $\Pi : \mathcal{S} \rightarrow \mathcal{A}$, defines for each state, the action to execute. The ac-

tion policy Π^* is optimal if it maximizes the gain W in the task Ω, say W^*_Ω.

Policy Reuse has the objective of solving a task Ω, i.e., to learn Π^*_Ω based on two main elements: (i) a set of previously solved tasks $\{\Omega_1, \ldots, \Omega_n\}$ resulting in a set of policies, $\{\Pi^*_1, \ldots, \Pi^*_n\}$; (ii) the use of the policies, Π^*_i to learn the new policy Π^*_Ω.

Policy Reuse defines a new exploration stratategy called π-reuse. It biases the learning of a new policy with one past policy. The goal of the π-reuse strategy is to balance random exploration, exploitation of the past policy, and exploitation of the new policy being learned. The π-reuse strategy follows the past policy, Π_{past} and the new policy with a probability ψ and $1-\psi$, respectively. As random exploration is always required, when exploiting the new policy, π-reuse follows an ϵ-greedy strategy.

Interestingly, the π-reuse strategy also contributes a similarity metric between policies, based on the gain obtained when reusing each policy. Let W_i be the gain obtained while executing the π-reuse exploration strategy, reusing the past policy Π_i. W_i is used as an estimation of how similar the policy Π_i is to the one we are currently learning. The set of W_i values, for $i = 1, \ldots, n$, is unknown a priori, but it can be estimated on-line while the new policy is computed in the different episodes. This idea is formalized in the PRQ-Learning algorithm, described in Table 1.

PRQL(Ω, L, K, H)
Given:
  A new task Ω we want to solve
  A Policy Library $L = \{\Pi_1, \ldots, \Pi_n\}$
  A maximum number of episodes to execute, K
  A maximum number of steps per episode, H
Initialize:
  $Q_\Omega(s,a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
  $W_\Omega = W_i = 0$, for $i = 1, \ldots, n$
For $k = 1$ to K do
  Choose an action policy, Π_k, assigning to each policy the probability of being selected computed by:
    $P(\Pi_j) = \frac{e^{\tau W_j}}{\sum_{p=0}^{n} e^{\tau W_p}}$ where W_0 is set to W_Ω
  Execute a Q-Learning episode k:
    If $\Pi_k = \Pi_\Omega$, exploit greedily the policy Π_k
    Otherwise, reuse Π_k through the π-reuse strategy
    In any case, receive the reward obtained in that episode, say R, and the updated Q function, $Q_\Omega(s,a)$
  Recompute W_k using R
Return the policy derived from $Q_\Omega(s,a)$

Table 1. PRQ-Learning.

3. Policy Reuse Across Tasks with Different State and Action Spaces

As defined in previous section, Policy Reuse requires that the action and state spaces of the different policies, and therefore, the transition function, are homogeneous. The core contribution of this work consists of applying Policy Reuse in a transfer learning problem with different state and action spaces among the different tasks, by creating a mapping between them.

We assume a past policy, Π_{past} that solves the task $\Omega_{past} =< \mathcal{D}_{past}, \mathcal{R}_{past} >$, where $\mathcal{D}_{past} =< \mathcal{S}_{past}, \mathcal{A}_{past}, \mathcal{T}_{past} >$. We want to learn a new policy Π_{new} to solve a new task $\Omega_{new} =< \mathcal{D}_{new}, \mathcal{R}_{new} >$, where $\mathcal{D}_{new} =< \mathcal{S}_{new}, \mathcal{A}_{new} >$. As a transfer learning goal, we want to reuse the past policy, Π_{past} to learn the new problem Π_{new}. Interestingly, given that our policy reuse method relies on policies, we only need to map states and actions, and we do not need to make any mapping between tasks, rewards nor transition functions. Thus, we need to find a mapping ρ that, given the policy Π_{past} in the domain $\mathcal{D}_{past}$, outputs a new policy $\hat{\Pi}_{past}$ that is executable in the domain $\mathcal{D}_{new}$. Equation 1 defines the mapping $\rho =< \rho_\mathcal{S}, \rho_\mathcal{A} >$, where $\rho_\mathcal{A} : \mathcal{A}_{past} \rightarrow \mathcal{A}_{new}$ is a function that maps the actions in the space $\mathcal{A}_{past}$ to the actions in the space $\mathcal{A}_{new}$, and $\rho_\mathcal{S} : \mathcal{S}_{new} \rightarrow \mathcal{S}_{past}$ maps states in the space $\mathcal{S}_{new}$ to the space $\mathcal{S}_{past}$. Next section describes the application of Policy Reuse for transfer learning in the Keepaway task.

$$\hat{\Pi}_{past}(s) = \rho_\mathcal{A}(\Pi_{past}(\rho_\mathcal{S}(s))) \quad (1)$$

4. Keepaway as a Reinforcement Learning Domain

The Keepaway domain is a subdomain of the robot soccer simulation domain (Stone et al., 2005). It consists of two teams, the keepers and the takers. The behavior of the takers is fixed. There are some low level skills predefined for the keepers, as "Go to Ball" or "Hold the Ball." A task consists of learning a high level control function of those skills for each of the keepers. The goal is to maximize the time that the keepers maintain the possesion of the ball. The end condition for each episode is that the keepers lose the ball or that the ball goes out of a fixed sub-area of the field.

The state space is defined in Table 2, that shows some features of the state space for different keepaway configurations (3 vs. 2, 4 vs. 3, and 5 vs. 4 keepaway). The action space is limited to the execution of two different behaviors, $HoldBall()$ and $PassBall(k_i)$. PassBall receives a parameter, which is the player who will receive the pass. Thus, when increasing the number of keepers, the number of actions also increases.
Different methods for state space generalization has been applied in this domain, as CMAC or neural networks. In our case, we have used the VQQL algorithm (Vector Quantization for Q-Learning) (Fernández & Borrajo, 2000). It is based on the unsupervised discretization of the state space with the k-means algorithm. This method generates a set of prototypes,

3vs2-keepaway	4vs3-keepaway	5vs4-keepaway
$dist(k_1, C)$	$dist(k_1, C)$	$dist(k_1, C)$
$dist(k_2, C)$	$dist(k_2, C)$	$dist(k_2, C)$
$dist(k_3, C)$	$dist(k_3, C)$	$dist(k_3, C)$
	$dist(k_4, C)$	$dist(k_4, C)$
		$dist(k_5, C)$
...	...	...
$\min(dist(k_2, t_1), dist(k_2, t_2))$	$\min(dist(k_2, t_1), dist(k_2, t_2), dist(k_2, t_3))$	$\min(dist(k_2, t_1), dist(k_2, t_2), dist(k_2, t_3), dist(k_2, t_4))$
$\min(dist(k_3, t_1), dist(k_3, t_2))$	$\min(dist(k_3, t_1), dist(k_3, t_2), dist(k_3, t_3))$	$\min(dist(k_3, t_1), dist(k_3, t_2), dist(k_3, t_3), dist(k_3, t_4))$
	$\min(dist(k_4, t_1), dist(k_4, t_2), dist(k_4, t_3))$	$\min(dist(k_4, t_1), dist(k_4, t_2), dist(k_4, t_3), dist(k_4, t_3))$
		$\min(dist(k_5, t_1), dist(k_5, t_2), dist(k_5, t_3), dist(k_5, t_3))$
13 features	19 features	25 features

Table 2. State spaces of the different keepaway tasks.

which together with the nearest neighbor rule, defines Voronoi regions. Each of these regions clusters a set of states of the original state space representation.

5. Evaluation

This section describes the experiments performed in Keepaway. Three different keeapway configurations, 3vs2-keepaway, 4vs3-keepaway and 5vs4-keepaway, are sequentially learned as defined in Table 3.

	3vs2-keepaway	4vs3-keepaway	5vs4-keepaway
Keeper 1	Learn $\Pi^{k_1}_{3vs2}$ from scratch	Learn $\Pi^{k_1}_{4vs3}$ by reusing $L^{k_1} = \{\Pi^{k_1}_{3vs2}\}$	Learn $\Pi^{k_1}_{5vs4}$ by reusing $L^{k_1} = \{\Pi^{k_1}_{3vs2}, \Pi^{k_1}_{4vs3}\}$
Keeper 2	Learn $\Pi^{k_2}_{3vs2}$ from scratch	Learn $\Pi^{k_2}_{4vs3}$ by reusing $L^{k_2} = \{\Pi^{k_2}_{3vs2}\}$	Learn $\Pi^{k_2}_{5vs4}$ by reusing $L^{k_2} = \{\Pi^{k_2}_{3vs2}, \Pi^{k_2}_{4vs3}\}$
Keeper 3	Learn $\Pi^{k_3}_{3vs2}$ from scratch	Learn $\Pi^{k_3}_{4vs3}$ by reusing $L^{k_3} = \{\Pi^{k_3}_{3vs2}\}$	Learn $\Pi^{k_3}_{5vs4}$ by reusing $L^{k_3} = \{\Pi^{k_3}_{3vs2}, \Pi^{k_3}_{4vs3}\}$
Keeper 4	Not Playing	Learn $\Pi^{k_4}_{4vs3}$ from scratch	Learn $\Pi^{k_4}_{5vs4}$ by reusing $L^{k_4} = \{\Pi^{k_4}_{4vs3}\}$
Keeper 5	Not Playing	Not Playing	Learn $\Pi^{k_5}_{5vs4}$ from scratch

Table 3. Description of the tasks solved.

5.1. Policy Reuse in the Keepaway

The 3vs2-keepaway is a task, $\Omega^{3vs2} =< \mathcal{D}^{3vs2}, \mathcal{R} >$, defined in the domain $\mathcal{D}^{3vs2}$, with a reward function $\mathcal{R}$. The domain is defined as a tuple, $\mathcal{D}^{3vs2} =< \mathcal{S}^{3vs2}, \mathcal{A}^{3vs2}, \mathcal{T}^{3vs2} >$. $\mathcal{S}^{3vs2}$ and $\mathcal{A}^{3vs2}$ were defined in previous section. Both the transition function $\mathcal{T}^{3vs2}$ and the reward function are unknown for the agent. The goal is to learn an action policy $\Pi_{\Omega^{3vs2}} : \mathcal{S}^{3vs2} \rightarrow \mathcal{A}^{3vs2}$ that outputs actions, given any state of the discretized state space.

In a similar way, the 4vs3-keepaway task is formalized as following: $\mathcal{D}^{4vs3} =< \mathcal{S}^{4vs3}, \mathcal{A}^{4vs3}, \mathcal{T}^{4vs3} >$; $\Omega^{4vs3} =< \mathcal{D}^{4vs3}, \mathcal{R} >$; and $\Pi_{\Omega^{4vs3}} : \mathcal{S}^{4vs3} \rightarrow \mathcal{A}^{4vs3}$.

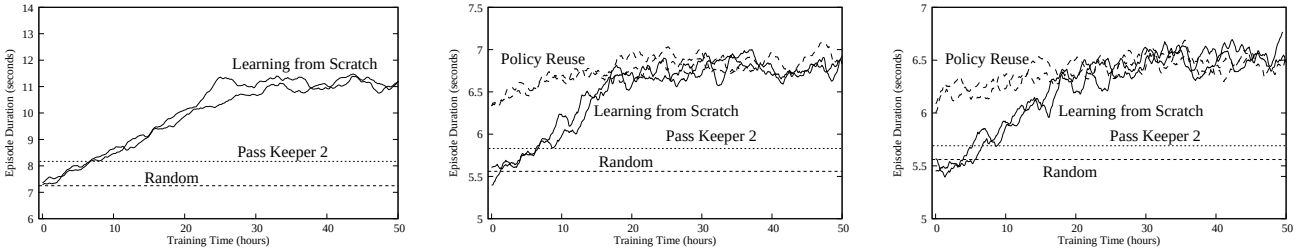
Following the notation introduced in Section 3, the mapping from a policy in the 3vs2-keepaway to the 4vs3-keepaway is performed as $\hat{\Pi}_{\Omega^{4vs3}}(s) = \rho_{\mathcal{A}}(\Pi_{\Omega^{3vs2}}(\rho_{\mathcal{S}}(s)))$, where $\rho_{\mathcal{S}}$ is a function $\rho_{\mathcal{S}} : \mathcal{S}^{4vs3} \rightarrow \mathcal{S}^{3vs2}$ that, given a state in the 4vs3-keepaway state space, $\mathcal{S}^{4vs3}$, projects it on the 3vs2-keepaway state space, $\mathcal{S}^{3vs2}$; and $\rho_{\mathcal{A}}$ is a function $\rho_{\mathcal{A}} : \mathcal{A}^{3vs2} \rightarrow \mathcal{A}^{4vs3}$ that maps an action in the 3vs2-keepaway action space on the 4vs3-keepaway action space.

In the keepaway task, these projections are derived from the semantic of the features and actions (Taylor & Stone, 2005). In the case of the action spaces, $\rho_{\mathcal{A}}(a) = a$, i.e. is the identity function, given that the action space in 3vs2-Keepaway is a subspace of the 4vs3-keepaway one. In the case of the state space, $\rho_{\mathcal{S}}$, the projection is derived from Table 2. For instance, each feature in 4vs3-keepaway maps to the feature of 3vs2-keepaway in the same row; the features in 4vs3-keepaway that does not have equivalent in the 3vs2-keepaway are eliminated. The same process can be generalized to different number of keepers and takers.

5.2. Results

In the experiments, we have used the Keepaway layer Framework 0.6 (Stone et al., 2005). Although the results can not be compared because we use a different simulator version, we use the same parameter settings that in a previous transfer learning paper (Taylor & Stone, 2005), namely the field size is 25×25, vision capabilities are set to full, and the synchronous mode is set on to speed up the simulator. In all the cases we introduce the performance of a random policy, and a policy where the agents always passes to the second keeper.

The parameter setting is the following: the size of the discretized state space is 512 states; in the Q-Learning equation, $\alpha = 0.125$, and $\gamma = 1$, as defined in other previous works (Taylor & Stone, 2005); in the π-reuse exploration strategy, $\psi = 1$, $\upsilon = 0.95$ and $\epsilon = 1 - \psi_h$; and in the PRQ-Learning algorithm, τ is initialized to 0, and incremented by 0.05 in each episode. When learning from scratch, an ϵ-greedy strategy is followed, increasing the value of *epsilon* from 0 (random behaviour) to 1 (greedy behaviour) by 0.0001 in each episode. All the experiments have been repeated twice, showing a small variance on the results.



(a) 3vs2-keepaway (b) 4vs3-keepaway (b) 5v4-keepaway

Figure 1. Results of learning in different configurations of Keepaway

Firstly, we have learned the 3vs2-keepaway. The results are summarized in Figure 1(a). The x axis of the figure describes the training time, while the y axis shows the episode duration, which is the value to maximize. In this task, the random behavior obtains a performance of 7.25 seconds, while the policy "Pass K2" obtains an average value of 8.17. When learning, the average values raises from 7.25 up to around 11 seconds.

Then, the agents are given the 4vs3-keepaway task. The results are summarized in Figure 1(b). We can differentiate two phases. The first one ranges from training time 0 to 20. In that phase, when learning from scratch, the performance raises from the same result than random behavior (around 5.5) up to more than 6.5. However, when the agents reuse the policy learned in the 3vs2-keepaway through Policy Reuse, the initial value is around 6.4, raising up to more than 6.5 in around 11 hours. This demonstrates that reusing the past policy improves the behavior of the learning agent since the early steps of the learning. The second phase, after 20 hours of learning, does not show significant differences in the learning curves.

Then, the 5vs4 keepaway is learned, obtaining the results shown in Figure 1(c). Qualitatively, these results are similar to the ones ontained for 4vs3-keepaway.

6. Conclusions

In this paper we demonstrate two main issues. Firstly, that Policy Reuse, together with a function approximation method, is an accurate transfer learning method, applicable also in domains with a large state space. And second, that policies that solve tasks in different state and action spaces can be succesfully reused to learn policies in a different state/action space.

Acknowledgments

This research was conducted while the first author was visiting Carnegie Mellon from the Universidad Carlos III de Madrid, supported by a generous grant from the Spanish Ministry of Education and Fullbright. He was partially sponsored also by the Ministry of Education and Science project number TIN2005-08945-C06-05 and by CAM-UC3M project number UC3M-INF-05-016. The second author was partially sponsored by Rockwell Scientific Co., LLC under subcontract no. B4U528968 and prime contract no. W911W6-04-C-0058 with the US Army, and by BBNT Solutions, LLC under contract no. FA8760-04-C-0002 with the US Air Force.

References

Fernández, F., & Borrajo, D. (2000). VQQL. Applying vector quantization to reinforcement learning. In *Robocup-99: Robot soccer world cup iii*, no. 1856 in Lecture Notes in Artificial Intelligence, 292–303. Springer Verlag.

Fernández, F., & Veloso, M. (2006). Probabilistic policy reuse in a reinforcement learning agent. *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Stone, P., Sutton, R. S., & Kuhlmann, G. (2005). Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, *13*.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Taylor, M. E., & Stone, P. (2005). Behavior transfer for value-function-based reinforcement learning. *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*. To appear.