TRANSFER LEARNING FOR CONTINUOUS STATE AND ACTION SPACES

ESTEBAN O. GARCIA* and ENRIQUE MUNOZ DE COTE† and EDUARDO F. MORALES‡

Computer Science, Instituto Nacional de Astrofisica, Optica y Electronica, Luis Enrique Erro Num. 1
Tonantzintla, Puebla 72840, Mexico
**eomargr@inaoep.mx*
†*jemc@inaoep.mx*
‡*emorales@inaoep.mx*

Transfer learning focuses on developing methods to reuse information gathered from a source task in order to improve the learning performance in a related task. In this work, we present a novel approach to transfer knowledge between tasks in a reinforcement learning framework with continuous states and actions, where the transition and policy functions are approximated by Gaussian processes. The novelty in the proposed approach lies in the idea of transferring qualitative knowledge between tasks. Our approach transfers information about the hyper-parameters of the state transition function from the source task, which represents qualitative knowledge about the type of transition function that the target task might have, constraining the search space and accelerating the learning process. We performed experiments on relevant tasks for reinforcement learning, which show a clear improvement in the overall performance of the system when compared to a state of the art reinforcement learning and transfer learning algorithms for continuous state and action spaces.

Keywords: Transfer learning; Reinforcement learning; Gaussian processes; Continuous states; Continuous actions

1. Introduction

The objective in reinforcement learning (RL) is to find a sequence of actions that maximizes a long-term cumulative reward. A RL algorithm achieves such an objective by exploring the world and collecting information about it in order to find the optimal sequence of actions.[1]

However, when a RL algorithm is applied to real world tasks, like the one just described, two major problems arise: (i) a large number of samples or interactions with the environment are needed to learn an optimal solution, and (ii) after an agent has learned to solve a task, if it is required to solve a different (although similar) task, the learning process needs to start from scratch.

A common approach to lessen the problem of learning a new, although similar task is to use transfer learning (TL).[2] This is an emerging area of study where several methods (see Section 2) are used to learn a task faster by taking advantage of the knowledge acquired by solving a related task. Lets say an agent has learned to control a multicopter, which like a helicopter, is also a vertical-take-off vehicle

but with several propellers instead of a rotor. The policy learned to control the multicopter cannot simply be used to control a helicopter because the dynamics are different and the actions would lead to undesirable results in the helicopter. However, as both vehicles have the same state variables and actions, and their dynamics are somehow related, both share a general behavior. For instance, in both cases an increase in throttle will lift the aircraft, which is the sort of information that can be transferred, so the new task might be learned faster.

The type of information that we harness in this work is qualitative, gathered from the transition function, which is the function that describes the way the states of the world change according to the actions taken. We take advantage of the Gaussian processes to represent distributions of transition functions, and the qualitative properties of the transition function (like smoothness, noise, etc.) are transferred to the target task, so the new task starts with more information about the transition function. In particular, we propose a transfer for *batch learning* methods which transfers qualitative information from the state transition function in the form of a prior about the type of function behind the transition of the target task. In order to jumpstart the learning process, we also investigate how an initial transfer of policy fares: This with the idea of gaining tuples around the solution trace in the first episode. We will show that by providing a family of functions as prior information about the underlying state transition function, significant reductions can be obtained in the convergence of the algorithm. Our proposal gradually incorporates the information from the target task producing a more stable process and faster convergence times. The main contribution of this paper is a very effective approach for transfer learning in continuous state and action spaces that is based on some intuitive ideas: (i) Within similar domains, you can expect similar properties on the type of state transition functions. (ii) Without any prior knowledge, it is probably better if one start exploring states using the previously learned policy from the source task. We performed experiments on relevant tasks in the context of RL and control, including cart-pole, mountain car and quadcopter to helicopter transfer, and show a significant improvement in the learning process.

The structure of this article is as follows. In Section 2 we give an overview of the work in transfer learning. Section 3 briefly introduces RL, GPs and how GPs can be used to represent state transition functions. In Section 4, the proposed transfer method is described in detail. Section 5 presents experimental results in several relevant problems for RL, under different conditions and comparisons. Section 6 summarizes the article and proposes future research directions.

2. Related work

Transfer learning for reinforcement learning already provides several approaches for many scenarios where previously acquired knowledge is used to learn a new task.[2] Nevertheless, not all of the methods can be applied to any given task, because it depends on the conditions between the tasks. A simple case, for instance, is when

only the reward function changes between the source and target task. In Ref. 3 an approach based on regression can infer the new reward function and improve learning in the target task. There are other methods that are only useful when the source task can be decomposed in a sort of subtasks that might be also found in the target task.[4] Some other methods focus in tasks where the source and target have different variables in their state vector, but can still be related through an *inter-task mapping*.[5,6] Unfortunately the mentioned approaches have been developed to face discrete problems or at most to deal with continuous states, but not actions. In this work, we focus in a challenging scenario related to real world tasks, where the variables that describe the state and actions are continuous and we assume the problem can not be discretized. In this scenario all other transfer learning approaches are unfeasible, at least without significant adaptations.

Even when several approaches in RL have been proposed to learn tasks in continuous domains, (e.g., Ref. 7, 8, 9, 10, 11, 12), they do not accomplish any transfer. On the other hand, most of the published methods in TL highly depend on the discrete algorithms, thus are not suitable to transfer information among continuous tasks. In Ref. 13 the source task action's Q-values are used to generate recommended actions for the target task. In the target task, the actions are constrained, with preference for different actions in different states so there is a bias for some actions according to the state of the agent. This schema is useful for tasks where the actions are discrete, however in tasks like the considered in this work, this sort of methods are unfeasible, because discretizing the actions would lead to unmanageable amounts of information. In Ref. 14 relationships between state-action pairs of both, the source and target tasks are found, a set of subroutines or skills that the agent performs (called *options*) are discovered, by grouping sequential transitions, so they can be used as preferred traces within specific states on the target task. That sort of subroutines are discovered only within a specific type of tasks, where sequences actions can be identified, but in the tasks where the actions are continuous, each action probably only occur once during the whole learning process, so sequences of actions would hardly be recognized. In Ref. 15, 16 and 5, it is proposed a transfer of samples or instances composed by ⟨state, action, reward, next state⟩ from the source task to the target task following similarity measures based on distance. In their proposed approaches, the actions are discrete and are used as indexes to cluster the tuples, which is the main reason those approaches could not be used directly with continuous actions.

When working directly with continuous spaces, most of the published work is related to the use of function approximators. In particular, Gaussian processes (GPs) have been used to represent value functions, [17,18,19,20] and more recently, to represent transition function models with very promising results. [21,22,23,24,11]

In Ref. 25 we introduced an approach to transfer qualitative information from a source task to a target task, focusing on transition function represented with a GP. In the present work, we extend the previous idea by using new Bayesian rule for combination that takes into account the uncertainty in the new task data. We

compare the performance of the two proposed methods. Two new relevant tasks in RL (*mountain car* and *helicopter control*) are implemented as benchmarks and the result are analyzed using more metrics than in previous work. In the present work, we also show a case in which the proposed approach reaches its limits. As a way to detect the cases in which the source and target tasks are different enough that the transfer is not useful, in this work a distance measurement called *task compliance* is calculated and analyzed.

3. Background

This section briefly covers Markov decision processes (MDPs) and Gaussian processes and how they can be used to represent the transition function.

3.1. *Markov decision processes (MDPs)*

RL problems are typically formalized as MDPs, defined by $\langle S, A, P, R\rangle$, where S is the set of states, A is the set of possible actions that the agent may execute, $P : S \times A \times S \to [0, 1]$ is the state transition probability function, describing the task dynamics, $R : S \times A \to \mathbb{R}$ is the reward function that defines the goal and measures the desirability of each state. A policy $\pi : S \to A$ maps states to actions. In the case of continuous domains $S = \mathbb{R}^D$ and $A = \mathbb{R}^F$, where D and F are the dimensions of the state and action vector respectively. Functions approximators can be used to represent the state transition function P and the policy function π. In this work, we use GPs to represent these functions, as described in the following sections.

3.2. *Gaussian processes*

A GP denoted by $\mathcal{GP}(m, k)$, is specified by a mean function $m(\cdot)$ and a covariance function $k(\cdot, \cdot)$, also called a *kernel*. Given a set of input vectors $\mathbf{x_i}$ arranged as a matrix $\mathbf{X} = [\mathbf{x_1}, \ldots, \mathbf{x_n}]$ and a vector of samples or training observations $\mathbf{y} = [y_1, \ldots, y_n]^\top$, Gaussian process methods for regression problems assume that the observations are generated as $y_i = h(\mathbf{x_i}) + \epsilon$, where ϵ is additive noise that follows an independent and indentically distributed Gaussian distribution with zero mean and variance σ_ϵ^2 ($\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$).

Given a Gaussian process model of the *latent function* $h \sim \mathcal{GP}(m, k)$, it is possible to predict function values for an arbitrary input $\mathbf{x}_*$. The predictive distribution of the function value $h_* = h(\mathbf{x}_*)$ for a test input $\mathbf{x}_*$ is Gaussian distributed with mean and variance given by:

$$E_h[h_*] = k(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\epsilon^2\mathbf{I})^{-1}\mathbf{y} \tag{1}$$

$$\mathtt{var}_h[h_*] = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\epsilon^2\mathbf{I})^{-1}k(\mathbf{X}, \mathbf{x}_*) \tag{2}$$

where $\mathbf{K} \in \mathbb{R}^{n\times n}$ is the kernel matrix with $K_{ij} = k(\mathbf{x_i}, \mathbf{x_j})$ and σ_ϵ^2 is a noise term.

The covariance function k commonly used is the squared exponential kernel:

$$k(\mathbf{x}, \mathbf{x}') = \alpha^2 \exp(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \mathbf{\Lambda}^{-1}(\mathbf{x} - \mathbf{x}')) + \delta_{xx'}\sigma_\epsilon^2 \tag{3}$$

with $\Lambda = \text{diag}([\ell_1^2, \ldots, \ell_n^2])$ and ℓ_k for $k = 1, \ldots, n$, being the characteristic length-scales, σ_ϵ^2 the noise term and $\delta_{xx'}$ denotes the Kronecker delta. The parameter α^2 describes the variability of the latent function h. The parameters of the covariance function or hyper-parameters of the GP (α^2, ℓ, σ_ϵ^2) are collected within the vector θ. The hyper-parameters define the shape of the functions in the prior distribution.

The kernel hyper-parameters are often optimized to adjust prior Gaussian distribution to data, using evidence maximization. See Ref. 26 for more details on Gaussian processes and evidence maximization.

3.3. *RL for continuous state and action spaces*

The unknown transition function P can be described as $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}), f \sim \mathcal{GP}(m, k)$, where $\mathbf{x}_t \in S$ is the state of the agent at time t, and is estimated by function f with the previous state and action (x_{t-1}, u_{t-1}) as arguments. The transition model f is distributed as a Gaussian process with mean function m and covariance function k. The sample tuples of the form $(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \in \mathbb{R}^{D+F}$ are taken as inputs and the corresponding $\Delta_t = \mathbf{x}_t - \mathbf{x}_{t-1} + \epsilon \in \mathbb{R}^D$, $\epsilon \sim \mathcal{N}(0, \Sigma_\epsilon)$, as training targets of the latent function. As the differences vary less than the original function, learning the differences is better than learning the function values directly.

The objective in RL is to find a policy π: $S \mapsto A$ that maximizes the expected accumulative reward given as:

$$V^\pi(\mathbf{x}_0) = \sum_{t=0}^{T} \mathbb{E}[r(\mathbf{x}_t)], \mathbf{x}_0 \sim \mathcal{N}(\mu_0, \Sigma_0) \tag{4}$$

which is the sum of the expected rewards $r(\mathbf{x}_t)$ obtained from a trace $(\mathbf{x}_0, \ldots, \mathbf{x}_T)$, T steps ahead, where π is a continuous function approximated by $\tilde{\pi}$, using a set of parameters ψ. For most continuous tasks, it is sometimes useful to use a saturating reward function $r(\mathbf{x}_t) = \exp(-d^2/\sigma_r^2)$ that rewards when the Euclidean distance d of the current state $\mathbf{x}_t$ to the target state $\mathbf{x}_{target}$ is short, where σ_r^2 controls the width of r.

The preliminary policy $\tilde{\pi}$ can be approximated by a radial basis function network with Gaussian basis functions, given by:

$$\tilde{\pi}(\mathbf{x}_*) = \sum_{s=1}^{N} \beta_s k_\pi(\mathbf{x}_s, \mathbf{x}_*) = \beta_\pi^\top k_\pi(\mathbf{X}_\pi, \mathbf{x}_*) \tag{5}$$

where $\mathbf{x}_*$ is a test input, k_π is the squared exponential kernel and $\beta_\pi = (\mathbf{K}_\pi + \sigma_\pi^2\mathbf{I})^{-1}\mathbf{y}_\pi$ is a weight vector. $\mathbf{K}_\pi$ is formed as $(K_\pi)_{ij} = k_\pi(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{y}_\pi = \tilde{\pi}(\mathbf{X}_\pi) + \epsilon_\pi$, $(\epsilon_\pi \sim \mathcal{N}(\mathbf{0}, \sigma_\pi^2\mathbf{I}))$ represents the training targets for the policy, with ϵ_π measurement noise. $\mathbf{X}_\pi = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$, $\mathbf{x}_s \in \mathbb{R}^D$, $s = 1, \ldots, N$, are the training inputs. The support points $\mathbf{X}_\pi$ and the corresponding training targets $\mathbf{y}_\pi$ are

pseudo-training set for the preliminary policy, that means that are adjusted by the algorithm that learns the policy. In this paper we use PILCO[11] to learn the policy.

The state transition function is learned as a GP, using available data, going from a prior distribution of transition functions to a posterior one. The learned transition model is then used to simulate the system and speculate about the long-term behavior without the need of interaction (batch learning). The policy can be represented with any approximator, but in this work we decided to use the approach proposed in Ref. 11, so it is learned using estimates of the gradient of the value function according to the simulations and after optimizing the policy, it is used to get more tuples (state, action, successor state) interacting with the environment again. This iteration cycle can be repeated as long as the desired behavior is not reached.

4. Qualitative transfer learning

The problem that we study is one where the source and target tasks have the same state and action spaces. For instance, the source task could be to learn how to drive a car while the target task could be to learn how to drive a small truck. In this sort of problems the general properties of the transition functions in both task are similar.

We transfer information from the hyper-parameters of the transition function of the source task to the target task, to qualitatively describe the expected shape of the transition function in the target task. In our case, we use a GP with a mean function defined as $m(\mathbf{x}) = \mathbf{0}$ and a squared exponential kernel k with covariance function as defined in Eq. 3. The inputs to the kernel function k are of the form $\tilde{\mathbf{x}} = [\mathbf{x}^\top \mathbf{u}^\top]^\top$, where all the variables from the state vector and the action are coupled into a single vector. The hyper-parameters that describe the shape of the transition function (e.g., smoothness, periodicity, variability, noise tolerance) in the prior distribution are α^2, ℓ, and σ_ϵ^2. However, when no expert knowledge is available about the function properties, the kernel hyper-parameters are often adjusted by an optimization process taking data into account and optimizing the log *marginal likelihood* (Equation **??**) by evidence maximization (see Ref. 26 for more detail). In our RL setting, a policy is used to gather new data and with this data a new transition function is estimated, thus the hyper-parameters are adjusted on every cycle.

4.1. *Initialization*

In this work, we use the information acquired in the source task to model a transition function prior distribution for the target task. We start the target task with the same distribution of transition function as the source task, by transferring the hyper-parameters and gradually updating them with information from the target task. We propose two updating approaches: (i) a forgetting geometric updating, which allows to use expert knowledge to control the ratio in which the new information

from the target task is incorporated into the model and (ii) a bayesian updating approach, which takes into account the confidence in the new information gathered in the target task. We will later show in Section 5 that these approaches create more stable values for the hyper-parameters at the beginning of the learning process and significantly reduce the convergence times of the algorithm.

Our algorithm starts with an initial policy to explore the environment. Once the algorithm gathers some data with its current policy it updates its transition function and recomputes a policy that considers the newly acquired information. The sooner the system obtains meaningful tuples of the task at hand, the faster it will learn an adequate policy. Furthermore, we also use the final policy of the source task as initial policy of the target task, which empirically provides, in our experiments, better initial traces than an initial random policy. We transfer the policy parameters denoted as ψ_s, consisting in β_π, and the corresponding free parameters for the policy kernel.

Transferring the hyper-parameters and the initial policy is not enough to solve the target task because the policy (action learned for each of the states) does not lead to the desired state as in the source task, but provides a clear advantage over an initial random policy. One advantage is to have some tuples (hopefully) in the neighborhood of the solution trace in the target task, which is better than exploring without any prior knowledge. In order to have a better approximation of the transition function distribution, we must use information obtained from the target task. So, after the initial episode, tuples and hyper-parameters values from the target task are updated and eventually will replace the source task information. To search the policy, we used the approach followed in PILCO[24] because it is reported as one of the fastest in the literature, but any other method for batch learning might be easily adapted as long as it could handle continuous states and actions.

Once a policy has been learned, it is tested in the environment and new tuples are collected. From the new tuples, new hyper-parameters are learned and combined to the transferred ones, following the approaches as will be explained in the following sections.

4.2. *Forgetting factor updating*

Qualitative properties from the source task are used as a departure point for the target task, but gradually, as information becomes more reliable in the target task, the information from the source task is dismissed. Now, these could be learnt from scratch directly in the target task but having transferred the source task's hyper-parameters gives the learning process useful information about the type of transition function to expect. Furthermore, as we show in the experimental section, it is not good enough to just transfer the hyper-parameters and let the process re-learn the correct hyper-parameters of the target task. The reason for this is that if left free, the process will override the transferred information almost immediately. Instead,

Algorithm 1 Qualitative Transfer learning

Require: θ_s, ψ_s
$\tilde{\pi} \leftarrow \pi(\psi_s)$
$\theta \leftarrow \theta_s$
Interact with environment, apply $\tilde{\pi}$ to obtain tuples.
repeat
  Infer transition function distribution f from tuples and hyper-parameters θ.
  repeat
    Evaluate policy $\tilde{\pi}$ over f. Get $V^{\tilde{\pi}}$
    Improve $\tilde{\pi}$ ▷ Updating parameters ψ
  until convergence
  $\tilde{\pi} \leftarrow \pi(\psi)$
  Interact with environment, apply $\tilde{\pi}$ to obtain more tuples.
  Learn θ_{p_i} from all tuples.
  Apply updating rule (Forgetting/Bayesian)
until task learned

we propose to adjust the hyper-parameters is using a forgetting factor.

Let $\theta = [\alpha^2, \ell, \sigma_\epsilon^2]^\top$ denote a vector of hyper-parameters. Let θ_s denote the hyper-parameters transferred from the source task, θ_i the hyper-parameters used in the kernel for the target task at episode i, θ_{p_i} the hyper-parameters learned by evidence maximization in the target task at episode i. We calculate the values of the hyper-parameters in the target task as follows:

$$\theta_0 = \theta_s \tag{6}$$

$$\theta_i = \gamma\theta_{i-1} + (1-\gamma)\theta_{p_i}, i > 0 \tag{7}$$

where $\gamma \in [0, 1]$ is the ratio at which previous episode hyper-parameters are incorporated into the kernel function. The value of γ specifies how much information about the general properties of the state transition function from the source task to use during the learning process of the target task.

4.3. *Bayesian updating*

Another way to update the hyper-parameters is using a Bayesian approach. At each episode k, we combine successive approximations of the value of the hyper-parameters learned by evidence maximization θ_{p_k}. We treat the learned value as a noisy approximation of the true value of the hyper-parameters. During the first episodes there is a small number of tuples of the target task, so the parameters learned by evidence maximization are less reliable than those learned from a large set of samples.

We model this by assuming that the learned value of each of the hyper-parameters in the target task is noisy with a distribution

$$p(\theta_{p_k}) \sim \mathcal{N}(\mu_p, \sigma_p^2) \tag{8}$$

while the posterior distribution is:

$$p(\theta|\theta_{p_k}) \sim \mathcal{N}(\mu_k, \sigma_k^2) \tag{9}$$

where

$$\sigma_k^2 = \frac{\sigma_p^2 \sigma_{k-1}^2}{\sigma_p^2 + \sigma_{k-1}^2} \tag{10}$$

$$\mu_k = \sigma_k^2 \left(\frac{\mu_{k-1}}{\sigma_{k-1}^2} + \frac{\mu_p}{\sigma_p^2} \right) \tag{11}$$

where σ_{k-1}^2 is the prior variance and σ_k^2 is the posterior variance.

The Bayesian approach allows us to take into account the initial uncertainty on the source task's hyper-parameters and adjust the confidence on the new learned hyper-parameters value, according to the data. The uncertainty is considered as inversely proportional to the amount of data:

$$\sigma_{k-1}^2 \propto \frac{1}{n} \tag{12}$$

where n is the number of tuples collected in the target task.

5. Experiments

Most TL approaches to RL transfer tuples and depend on discrete action spaces, so it is difficult to make a fair comparison. Moreover, algorithms of TL for RL that might work in batch mode (Ref. 16 and 5) were reported to learn in hundreds of episodes while in our work our results are in the order of tens of episodes at most. In our experiments we contrast the performance of our proposed transfer approach, qualitative transfer learning (QTL), with PILCO *tabula rasa*. PILCO is a state of the art technique used for learning in continuous spaces[11] that is reported to learn several orders of magnitude faster than other algorithms, we use it as part of our approach to search the policy.

In this section we show experimental results in three different tasks commonly used as benchmarks to compare reinforcement learning and control algorithms. The inverted pendulum task is a well known problem used in TL and control as benchmark. In this experiment we tested the behavior of the proposed algorithm under different conditions and compare it to other approaches. The mountain car scenario, is to show a a case where the conditions may lead to negative transfer. The transfer from the quadcopter to the helicopter task is a problem, where many variables are involved. In this task we test our approach in a challenging problem with many state and action variables with a very sensitive transition function, in the sense that small changes in actions lead big changes in state. In the mountain car and in the helicopter tasks, we compare only our Bayesian and PILCO approaches.

In all of the above tests we repeated the procedure five times, randomly selecting the initial state, the learning curves were averaged and plotted with their corresponding standard deviation.

For PILCO, the Kernel hyper-parameters in the source task were initialized with heuristic values, as proposed in Ref. 11. The initial training set for the transition function was generated by applying actions drawn uniformly from $[-\mathbf{u}_{\max}, \mathbf{u}_{\max}]$.

5.1. *Inverted pendulum on a cart*

In this task, an inverted pendulum has to be balanced by swinging it. The pendulum is attached to a cart that moves along one axis when an external force is applied. This problem involves applying actions that temporarily move the pendulum away from the target state, and the agent has to apply two different control criteria, one to swing the pendulum up and the other to balance it, thus it is non trivial to solve.

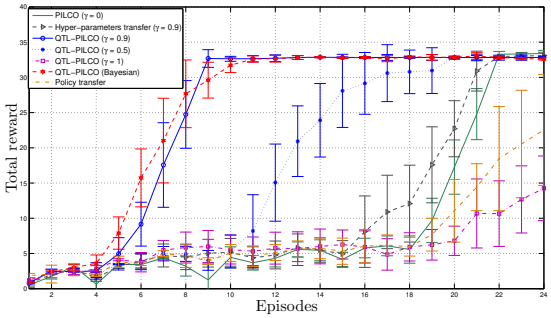
For our experiments, we tested the inverted pendulum problem with continuous action and state spaces, as described in Ref. 11. In this formulation not only the pendulum has to be balanced, but it has to be maintained at a specific point as well, which is more challenging. In the continuous scenario, a state $\mathbf{x}$ is formed by the position x of the cart, its velocity $\dot{x}$, the angle θ of the pendulum, and its angular velocity $\dot{\theta}$. The reward function is expressed as:

$$r(\mathbf{x}) = \exp(-\frac{1}{2}ad^2) \tag{13}$$

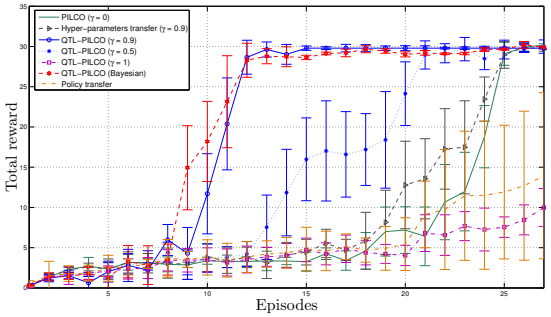
where a is a scale constant of the reward function (set to 0.25 in the experiments) and d is the Euclidean distance between the current and desired states, expressed as $d\left(\mathbf{x}, \mathbf{x}_{target}\right)^2 = x^2 + 2xl\sin\theta + 2l^2 + 2l^2\cos\theta$. The reward remains close to zero if the distance of the pendulum tip to the target is greater than $l = 0.6m$. The source task consists of swinging a pendulum of mass 0.5 Kg. while in the target tasks the pendulums weights are changed to 0.8, 1.0, 1.5, and 2.0 Kg., respectively.

In our proposed methodology, 8 hyper-parameters for each of the kernels $\mathbf{K}_i$, are taken from the source task. The hyper-parameters are length scales $\ell_1^2, \ldots, \ell_D^2$ for each of the D state variables (where angles are represented by their sin and cos values in order to deal with the pendulum spinning beyond one turn). The last two hyper-parameters are signal variance α^2 and noise variance σ_ϵ^2. Therefore 32 free variables are considered (considering one kernel for each of the four state variables for this domain). We performed experiments with different values for γ, from $\gamma = 0$, which is equivalent to learning with PILCO, to $\gamma = 1.0$ which uses the hyper-parameters found in the source task. For the Bayesian updating, after the first episode, the hyper-parameters are fused with the hyper-parameters learned by evidence maximization from the samples, as stated in Equations 8 to 11.

A comparison of the learning curves for target tasks is shown in Figure 1, where we plot PILCO and QTL with different values of γ and with the Bayesian updating. The learning curves for only policy transfer and for only hyper-parameters transfer without updating are plotted too. The horizontal axis shows the number of episodes (interactions with the environment) while the vertical axis shows the total reward, which is computed as the cumulative count of $r(\mathbf{x})$ at every step.



(a) Mass 1.5



(b) Mass 2.0

Fig. 1. Learning curves for target tasks of 1.5Kg and 2.0Kg learned from a 0.5Kg source task. Error bars represent the standard deviation.

The proposed transfer learning approach can significantly reduce the learning process. When the target task is quite similar to the source task (in this case, with a similar mass), QTL-PILCO shows a clear improvement over learning without transfer. When the target task is less similar (larger mass) the improvement is much more noticeable, the transfer learning has more area under the curve (see Table 2), which means our approach obtains more overall reward during the learning process. The performance of the policy learned is measured as the cumulative reward averaged in the latest three episodes of the learning process. Our proposed transfer method had the best performance in three of the four tasks, the 1.5Kg. task was the only where the PILCO approach learned a slightly better policy at the end of the test as shown in Table 1.

Both, QTL-PILCO with $\gamma = 0.9$ and the Bayesian approaches converge faster to a good policy than PILCO. The more drastic transfer is when the mass of the pendulum is 2.0Kg. In that case, the transfer of QTL-PILCO converges 13 episodes before the learning without transfer, as can be seen in Table 3. The time to

Table 1. Final performance measured in total accumulated reward, averaged from the latest 3 episodes (NC means No convergence).

Approach	0.8Kg.	1.0Kg.	1.5Kg.	2.0Kg.
PILCO ($\gamma = 0$)	36.95	35.53	**33.32**	29.71
Hyper-parameters transfer	37.08	33.04	32.98	29.66
QTL-PILCO $\gamma = 0.9$	37.55	34.97	32.84	29.79
QTL-PILCO $\gamma = 0.5$	37.20	34.50	32.78	29.90
QTL-PILCO $\gamma = 1$	37.06	34.59	NC	NC
QTL-PILCO Bayesian	**37.57**	**35.66**	32.68	**29.97**
Policy Transfer	35.77	32.81	NC	NC

Table 2. Area under the learning curve.

Approach	0.8Kg.	1.0Kg.	1.5Kg.	2.0Kg.
PILCO ($\gamma = 0$)	228.47	206.09	218.63	204.41
Hyper-parameters transfer	236.59	185.99	260.33	240.98
QTL-PILCO $\gamma = 0.9$	**287.44**	236.20	589.57	527.65
QTL-PILCO $\gamma = 0.5$	240.86	217.40	415.95	366.68
QTL-PILCO $\gamma = 1$	246.41	212.92	146.44	117.63
QTL-PILCO Bayesian	272.91	**236.43**	**602.55**	**539.79**
Policy Transfer	214.35	156.76	173.90	150.64

Table 3. Time to convergence, measured in number of episodes to reach 95% of performance.

Approach	0.8Kg.	1.0Kg.	1.5Kg.	2.0Kg.
PILCO ($\gamma = 0$)	9	10	22	25
Hyper-parameters transfer	9	11	22	25
QTL-PILCO $\gamma = 0.9$	**7**	**9**	**9**	**12**
QTL-PILCO $\gamma = 0.5$	9	10	20	21
QTL-PILCO $\gamma = 1$	9	10	Unknown	Unknown
QTL-PILCO Bayesian	8	**9**	10	**12**
Policy Transfer	10	11	Unknown	Unknown

convergence is calculated as the number of episodes to reach 95% of the performance value.

One advantage of our method is the stabilization of the hyper-parameters, so the learning process focuses more on learning the policy than trying to guess the hyper-parameters without having enough information. The values of the hyper-

parameters learned by evidence maximization can change drastically during the first iterations of the learning process due to poor samples.

5.2. *Mountain car*

This problem is a common test bed in a RL context, it consists of a car in a valley between two hills, the agent must learn a strategy to take the car up to the right side hill. The car cannot drive up at full throttle, so a strategy to gain inertial energy must be learned. The strategy consist of driving up the opposite hill to gain speed, which implies going away from the objective before reaching it.

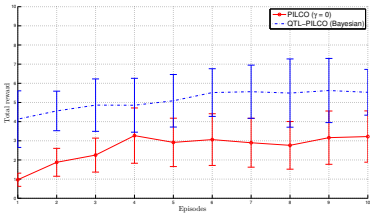
In RL literature, the problem is described using discrete variables for actions (left, zero, right).[1] In our tests, we address the problem using continuous spaces for states and actions. An agent's state is a vector $\mathbf{x} = (x, \dot{x})$, where x is the position on the horizontal axis and $\dot{x}$ is the velocity of the car on the horizontal plane too. The action a is a single variable corresponding to the force applied to the car. The initial state is $\mathbf{x_0} = (-5, 0)$, and the goal state is whenever $x > 0.5$, however, here we consider a bit more challenging task of stopping the car as soon as it reaches the right hill, so the goal state is $\mathbf{x_{target}} = (1, 0)$. The reward function is expressed as Eq. 13 where a is a scale constant of the reward function (set to 0.25 in the experiments) and $d = x$. The agent receives a zero reward at every time step when the goal is not reached.

For this experiment, we consider as source task the one specified in Ref. 1. For target tasks, we tested the same problem with a modified engine power of 50%, 150% and 300% the power of the source task.

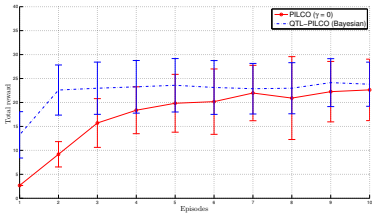
The results in Figure 2 show an improvement in the first two tasks, which are similar to the source task. In this two tasks the agent has to learn to swing the car, because there is not enough power to climb the hill. The third task however is a special case where the car has enough power to climb the hill without swinging. For this task there is negative transfer (transferring hyper-parameters leads to worst results than *tabula rasa*). This is because the transition function is not similar between the source and the target task. In order to try to avoid this scenario, some model distance measurement might be used to decide whether or not to transfer if enough data is available in both source and target tasks. We show in Table 4 the *task compliance* measure on the three target tasks, we used the metric described by Equation 14 (for more detail see Ref. 16). It is noticeable that task compliance is low for the 300% engine power task. A naive approach could be to set a fixed threshold value for Λ_{compl}. The compliance metric, measures the probability that source task S is the model from where the tuples $\hat{T}_{\langle s,a\rangle}$ might have been generated, briefly described by the equation

$$\Lambda_{compl} = \frac{1}{|\hat{U}|} \sum_{\langle s,a\rangle \in \hat{U}} P(S|\hat{T}_{\langle s,a\rangle}) \tag{14}$$

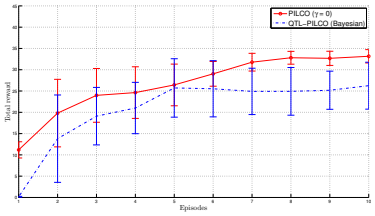
where $\hat{U}$ contains all the state-action pairs in the tuples from the target task $\hat{T}$



(a) 50% Engine Power



(b) 150% Engine Power



(c) 300% Engine Power

Fig. 2. Mountain car. Learning curves for target tasks of 50%, 150% and 300% car engine power, learned from a car with 100% power available. Error bars represent the standard deviation.

Table 4. *Task compliance* measurement on the three target tasks for the mountain car problem.

Engine power	Task compliance
50%	0.64
150%	0.76
300%	0.34

5.3. *Quadcopter to helicopter*

This task is the most complex and interesting experiment for the proposed approach. The task consists of finding a policy to get from an initial (on land) position to a desired position, specified by $(x_{target}, y_{target}, z_{target})$ coordinates. So the agent must learn to take off, deal momentarily with the *ground effect*[a], reach an specific three-dimensional position, and finally keep the vehicle stabilized at that spot. This task is learned in a quadcopter and then transfer to a helicopter which is a related vehicle, but with different dynamics.

This is the most challenging task for our proposed framework because an aircraft autonomous control is a precision task, and from the point of view of RL, the

[a]Ground effect is an aerodynamic effect derived from air hitting the ground when an aircraft is close to a surface that makes harder to lift and control the aircraft.

Table 5. Area under the learning curve for the helicopter task.

Approach	Area under curve
PILCO ($\gamma = 0$)	2225.4
QTL-PILCO Bayesian	3803.7

difficulty comes also from the number of variables of state and control, and the number of episodes required to learn. We must note that there has been work on apprenticeship learning where the agent is able to do aerobatic flight[27], but it requires of an expert to have previously repeated the manoeuvre several times. On the contrary, we focus on the transfer process where no expert or human knowledge is required.

Although both aircraft have the same state variables and same action variables, they behave different due to different aerodynamics. The quadcopter has four propellers which generate lift, the change in the speed of the propellers induces a change in the attitude of the quadcopter and a change in position. In the quadcopter, the difference between the torque generated by the motors is used to change the yaw angle. On the other hand, the helicopter has a main rotor which generates lift and changes position by changing the blades' angle as they rotate around the main axis. The helicopter also has a tail rotor to compensate the torque generated by main rotor. So, in order to control the yaw angle the helicopter changes the pitch in the tail rotor's blades.

Both, quadcopter and helicopter, have a state vector with 12 variables, comprising its position (x, y, z), orientation (roll ϕ, pitch θ, yaw ω), velocity $(\dot{x}, \dot{y}, \dot{z})$ and angular velocity $(\dot{\phi}, \dot{\theta}, \dot{\omega})$. We define the goal position as $[x, y, z] = [-1, -1, 1.5]$, starting from $[x, y, z] = [0, 0, 0]$. The reward function obeys Eq. 13, with a set to 0.25 and d evaluated as $d\left(\mathbf{x}, \mathbf{x}_{target}\right)^2 = x^2 + y^2 + z^2$.

For simulation purposes we use V-REP[28], which is a robotics simulator, where the quadcopter and helicopter dynamics models run. Figure 3 shows the learning curves for the helicopter task. We plot the PILCO (*tabula rasa*) learning curve, the QTL-PILCO approach and the total reward obtained by using the autopilot implemented in the simulator. The reward obtained by the control algorithm provided by the simulator is shown as a reference in the figure. Both, PILCO and our approach learn a better policy than the autopilot, as in both cases the agent learns to compensate the inertia of the helicopter by tilting pitch and roll angles before the helicopter reaches the target position in contrast to V-REP's autopilot which tends to overshoot. The transfer reaches a correct policy about seven episodes before PILCO.

The transfer in this challenging scenario shows empirically that the proposed approach is useful in real problems with high dimensionality where the transition function represents sensitive information (the behavior of the autonomous heli-

Table 6. Time to convergence is measured in number of episodes to reach 95% of performance. Performance is total reward averaged from the last three episodes (showed in parenthesis).

Approach	Time to convergence and performance
PILCO ($\gamma = 0$)	24 (132.5)
QTL-PILCO Bayesian	19 (131.99)

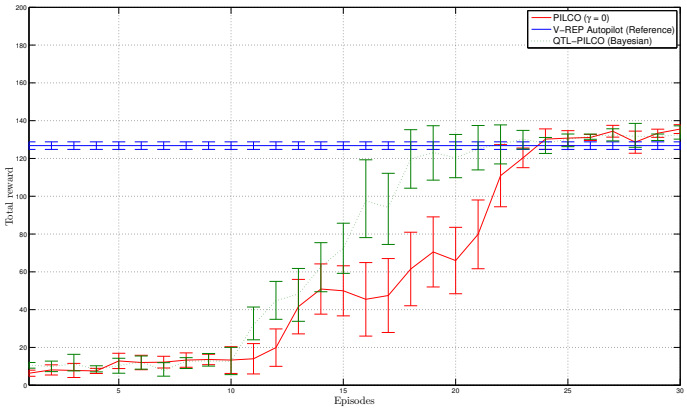


Fig. 3. Learning curves for the helicopter task, learned PILCO and with transfer from the quadcopter task. Reward acquired with V-REP autopilot is shown as reference.

copter is very sensitive to the variations in the actions). These results lead to think this approach could be useful to accelerate learning in real applications by transferring qualitative knowledge from simulated environments.

In a succinct way, the results drawn from these experiments are:

- Transferring information of the general properties of the state transition function (in the form of hyper-parameters) can significantly reduce the convergence times of the algorithm.
- A gradual incorporation of the hyper-parameters found for the target task provides more stable values for the hyper-parameters.
- This approach is suitable for several tasks, including those with many variables in state and action spaces.

6. Conclusions

In this paper we have presented a transfer learning approach for reinforcement learning with continuous state and action spaces. The proposed approach is simple, yet very effective for transferring knowledge between related tasks. It works by

refining the transition function model in the target task (modeled as a Gaussian process) using qualitative knowledge from the source task, like smoothnes, variance and noise properties of the transition function distribution. Two variants are proposed to transfer qualitative knowledge to the target task. The first one gradually incorporates the value of the hiper-parameters learned in the source task. The second one uses uncertainty in the value of the learned hyper-parameters in the target task to fuse with the transferred knowledge accordingly.

We performed experiments in three relevant tasks for reinforcement learning under different conditions and showed that our transfer learning approach significantly improves over a state-of-the-art algorithm that learns without transfer or by transferring directly the policy or hyper-parameters.

Because of its relationship to real tasks, continuous states and actions problems are a relevant part of reinforcement learning research, however, to the best of our knowledge, no other known approach transfers information for tasks with continuous states and actions. The idea of transferring qualitative behaviors between tasks is a novel idea that can be extended along several directions. So far only prior transition function distribution is being refined. We would like to develop methods to synthesize tuples in the target task in order to also adjust posterior distribution of the transition function.

References

1. R. Sutton and A. Barto, *Introduction to Reinforcement Learning*. MIT Press, 1998.
2. M. E. Taylor and P. Stone, "Transfer Learning for Reinforcement Learning Domains : A Survey," *Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.
3. C. Atkeson and J. Santamaria, "A comparison of direct and model-based reinforcement learning," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 4, pp. 3557–3564, IEEE, 1997.
4. C. Drummond, "Accelerating Reinforcement Learning by Composing Solutions of Automatically Identified Subtasks.," *J. Artif. Intell. Res. (JAIR)*, vol. 16, pp. 59–104, 2002.
5. M. E. Taylor, N. K. Jong, and P. Stone, "Transferring Instances for Model-Based Reinforcement Learning," *Machine Learning*, 2008.
6. H. Ammar, K. Tuyls, and M. Taylor, "Reinforcement learning transfer via sparse coding," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems- Volume 1*, no. Aamas, pp. 383–390, 2012.
7. A. Lazaric, M. Restelli, and A. Bonarini, "Reinforcement learning in continuous action spaces through sequential monte carlo methods," in *Advances in neural information processing systems*, Citeseer, 2007.
8. H. V. Hasselt, "Reinforcement Learning in Continuous State and Action Spaces," in *Reinforcement Learning: State of the Art*, Springer, 2011.
9. J. A. Martín H, J. de Lope, and D. Maravall, "Robust high performance reinforcement learning through weighted k-nearest neighbors," *Neurocomputing*, vol. 74, pp. 1251–1259, Mar. 2011.
10. H. P. van Hasselt, *Insights in Reinforcement Learning: formal analysis and empirical evaluation of temporal-difference learning algorithms*. PhD thesis, Universiteit Utrecht, January 2011.

11. M. P. Deisenroth and C. E. Rasmussen, "PILCO: A Model-Based and Data-Efficient Approach to Policy Search," in *ICML* (L. Getoor and T. Scheffer, eds.), ICML '11, pp. 465–472, 2011.
12. A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry, "Autonomous Helicopter Flight via Reinforcement Learning," in *Advances in Neural Information Processing Systems 16* (S. Thrun, S. Lawrence, and B. S., eds.), (Cambridge, MA), MIT Press, 2004.
13. L. Torrey, T. Walker, J. Shavlik, and R. Maclin, "Using advice to transfer knowledge acquired in one reinforcement learning task to another," in *Machine Learning: ECML 2005*, pp. 412–424, 2005.
14. V. Soni and S. Singh, "Using homomorphisms to transfer options across continuous reinforcement learning domains," in *AAAI*, pp. 494–499, 2006.
15. A. Lazaric, *Knowledge transfer in reinforcement learning*. Phd thesis, Politecnico di Milano, 2008.
16. A. Lazaric, M. Restelli, and A. Bonarini, "Transfer of samples in batch reinforcement learning," *Proceedings of the 25th international conference on Machine learning - ICML '08*, pp. 544–551, 2008.
17. Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," *Proceedings of the 22nd international conference on Machine learning - ICML '05*, pp. 201–208, 2005.
18. Y. Engel, S. Mannor, and R. Meir, "Bayes meets bellman: The gaussian process approach to temporal difference learning.," in *ICML* (T. Fawcett and N. Mishra, eds.), pp. 154–161, AAAI Press, 2003.
19. C. Rasmussen and M. Kuss, "Gaussian Processes in Reinforcement Learning," *Advances in Neural Information Processing Systems 16*, vol. 16, 2004.
20. M. Deisenroth, J. Peters, and C. E. Rasmussen, "Approximate dynamic programming with Gaussian processes," in *American Control Conference*, no. June, pp. 4480–4485, 2008.
21. R. Murray-Smith and D. Sbarbaro, "Nonlinear adaptive control using non-parametric Gaussian process prior models," in *IN 15TH IFAC WORLD CONGRESS ON AUTOMATIC CONTROL*, no. July, pp. 21–26, 2002.
22. C. Rasmussen and M. Deisenroth, "Probabilistic inference for fast learning in control," *Recent Advances in Reinforcement Learning*, vol. 5323, no. November, pp. 229–242, 2008.
23. M. Deisenroth, C. Rasmussen, and J. Peters, "Model-based reinforcement learning with continuous states and actions," in *16th European Symposium on Artificial Neural Networks*, no. April, pp. 19–24, 2008.
24. M. Deisenroth, C. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," in *Proceedings of Robotics: Science and Systems*, (Los Angeles, CA, USA), 2011.
25. E. O. Garcia, E. Munoz de Cote, and E. F. Morales, "Qualitative Transfer for Reinforcement Learning with Continuous State and Action Spaces," in *18th. Iberoamerican Congress on Pattern Recognition* (J. Ruiz-Schucloper and G. Sanniti di Baja, eds.), no. ii, pp. 198–205, Springer-Verlag, 2013.
26. C. E. Rasmussen and C. Williams, "Gaussian Processes for Machine Learning," *International Journal of Neural Systems*, vol. 14, no. 2, pp. 69–106, 2006.
27. P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous Helicopter Aerobatics through Apprenticeship Learning," *The International Journal of Robotics Research*, June 2010.
28. C. Robotics, *V-REP Pro Edu, Version 3.0.1*, 2013.