TRANSFER LEARNING FOR CONTINUOUS STATE AND ACTION SPACES

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

Computer Science, Instituto Nacional de Astrofísica
Optica y Electrónica
Tonantzintla, Puebla 72840, Mexico
*ecomargr@inaoep.mx
†jemc@inaoep.mx
‡emorales@inaoep.mx

Received 20 February 2014
Accepted 18 July 2014
Published 22 September 2014

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 (RL) 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 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 RL, which show a clear improvement in the overall performance when compared to 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. RL algorithms achieve such an objective by exploring the world and collecting information about it in order to learn the optimal sequence of actions (see Ref. 27). However, RL applied to real-world tasks reveal two major drawbacks: (i) A large number of samples or interactions with the environment are needed to learn an optimal solution, and (ii) after an agent has

*Corresponding author.
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) (see Ref. 29 for a deeper review on TL). This is an emerging area of study where several methods are used to learn a task faster by taking advantage of the knowledge acquired by solving a related task. This work’s objective is twofold: (i) to make a target task be learnt faster, and (ii) to start the learning process with a better than random initial policy (initial jumpstart). Furthermore, we want both objectives to be achieved without sacrificing the quality of the solution.

The type of information that we aim for transferring knowledge is qualitative, gathered from the transition function.\textsuperscript{a} We take advantage of Gaussian processes (GPs) to represent families of transition functions, and the qualitative properties of the transition function (like smoothness, noise, etc.) are transferred to the target task. In doing so, the new task starts with more information about the transition function. In order to jumpstart the learning process, we also investigate how an initial transfer of policy fares: This with the idea of gaining information in the neighborhood of the possible solution early in the learning process. We will show that by providing a family of functions as prior information about the underlying state transition function, significant reductions to the convergence time can be obtained. The main contribution of this paper is an effective approach for TL in continuous state and action spaces that are based on qualitative 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 to start exploring states close to the solution obtained from the source task. We performed experiments on RL and control benchmarks, like cart-pole, mountain car and a more realistic scenario of transferring knowledge from a quadcopter to a helicopter, and show a significant improvement in both of our objectives when compared with state-of-the-art algorithms.

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

2. Related Work

TL for RL already provides several approaches for many scenarios where previously acquired knowledge is used to learn a new task (Ref. 29). Nevertheless, not of the methods can be applied to any given task. A simple case, for instance, is when only the reward function changes between the source and target task. In Ref. 3, an

\textsuperscript{a}Which is the function that describes the way the states of the world change according to the actions taken.
Transfer Learning for Continuous State and Action Spaces

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 subtasks that might also be found in the target task (Ref. 9). Some other methods focus in domains where the source and target tasks have different variables in their state vector, but can still be related through an inter-task mapping (Refs. 2 and 28). In Ref. 30, 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. In Ref. 26, 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 of actions can be identified, but in tasks where the actions are continuous, each action will probably only occur once during the whole learning process, so sequences of actions would hardly be recognized. References 15, 17 and 28, propose 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.

Unfortunately all 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 describe that the state and actions are continuous and we assume the problem cannot be discretized. In this scenario, all other TL 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., Refs. 7, 13, 14, 16, 18 and 21), they do not TL between tasks. In fact, when working directly with continuous spaces, most of the published work is related to the use of function approximators. In particular, GPs have been used to represent value functions (Refs. 4, 10, 11 and 23) and more recently, to represent transition function models with very promising results (Refs. 5–7, 20 and 22). In Ref. 12, we introduced an approach to transfer qualitative information from a source task to a target task, focusing on transition function represented with a GP. This paper extends our previous work in the following ways: (i) We introduce a new Bayesian rule for updating the hyper-parameters that takes into account the uncertainty in the new task data and compare it with our previous approach. (ii) We performed experiments on two new relevant tasks in RL, namely mountain car and a more realistic and challenging problem, helicopter control. (iii) We performed more tests and incorporate more evaluation metrics. (iv) We also show a case in which the proposed approach reaches its limits and propose to use a measure called task compliance to deal with such cases.
3. Background

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

3.1. Markov decision processes

RL problems are typically formalized as MDPs, defined by $(S, A, P, R)$, where $S$ is the set of states, $A$ is the set of possible actions that the agent may execute, $P: S \times A \times S \rightarrow [0, 1]$ is the state transition probability function, describing the task dynamics, $R: S \times A \rightarrow \mathbb{R}$ is the reward function that defines the goal and measures the desirability of each state. A policy $\pi: S \rightarrow A$ 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 $\pi$. 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 $x_i$ arranged as a matrix $X = [x_1, \ldots, x_n]$ and a vector of samples or training observations $y = [y_1, \ldots, y_n]^T$, GP methods for regression problems assume that the observations are generated as $y_i = h(x_i) + \epsilon$, where $\epsilon$ is additive noise that follows an independent and identically distributed Gaussian distribution with zero mean and variance $\sigma^2_\epsilon$ ($\epsilon \sim \mathcal{N}(0, \sigma^2_\epsilon)$).

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

$$E_h[h] = k(x_i, X)(K + \sigma^2_\epsilon I)^{-1}y$$

$$\text{var}_h[h] = k(x_i, x_i) - k(x_i, X)(K + \sigma^2_\epsilon I)^{-1}k(X, x_i),$$

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

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

$$k(x, x') = \alpha^2 \exp\left(-\frac{1}{2}(x - x')^T \Lambda^{-1}(x - x')\right) + \delta_{xx'} \sigma^2_\epsilon$$

with $\Lambda = \text{diag}(\ell_1^2, \ldots, \ell_n^2)$ and $\ell_k$ for $k = 1, \ldots, n$, being the characteristic length-scales, $\sigma^2_\epsilon$ the noise term and $\delta_{xx'}$ denotes the Kronecker delta. The parameter $\alpha^2$ describes the variability of the latent function $h$. The parameters of the covariance function or hyper-parameters of the GP ($\alpha^2$, $\ell$, $\sigma^2_\epsilon$) are collected within the vector $\theta$. 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. 24 for more details on GPs and evidence maximization.

3.3. RL for continuous state and action spaces

The unknown transition function $P$ can be described as $x_t = f(x_{t-1}, a_{t-1})$, $f \sim \mathcal{GP}(m, k)$, where $x_t \in S$ is the state of the agent at time $t$, and is estimated by function $f$ with the previous state and action $(x_{t-1}, a_{t-1})$ as arguments. The transition model $f$ is distributed as a GP with mean function $m$ and covariance function $k$. The sample tuples of the form $(x_{t-1}, a_{t-1}) \in \mathbb{R}^{D+F}$ are taken as inputs and the corresponding $\Delta_t = x_t - x_{t-1} + \epsilon \in \mathbb{R}^D$, $\epsilon \sim \mathcal{N}(0, \Sigma_t)$, 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 $\pi: S \mapsto A$ that maximizes the expected accumulative reward given as:

$$V^\pi(x_0) = \sum_{t=0}^{T} \mathbb{I}[r(x_t)], \quad x_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$$

which is the sum of the expected rewards $r(x_t)$ obtained from a trace $(x_0, \ldots, x_T)$, $T$ steps ahead, where $\pi$ is a continuous function approximated by $\tilde{\pi}$, using a set of parameters $\psi$. For most continuous tasks, it is sometimes useful to use a saturating reward function $r(x_t) = \exp(-d^2/\sigma^2_{\pi})$ that rewards when the Euclidean distance $d$ of the current state $x_t$ to the target state $x_{\text{target}}$ is short, where $\sigma^2_{\pi}$ 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:

$$\tilde{\pi}(x_s) = \sum_{s=1}^{N} \beta_s k_{\pi}(x_s, x_s) = \beta^T_{\pi} k_{\pi}(X_{\pi}, x_s),$$

where $x_s$ is a test input, $k_{\pi}$ is the squared exponential kernel and $\beta_{\pi} = (K_{\pi} + \sigma^2_{\pi}I)^{-1}y_{\pi}$ is a weight vector. $K_{\pi}$ is formed as $(K_{\pi})_{ij} = k_{\pi}(x_i, x_j)$, where $y_{\pi} = \tilde{\pi}(X_{\pi}) + \epsilon_{\pi}$, $(\epsilon_{\pi} \sim \mathcal{N}(0, \sigma^2_{\pi}I))$ represents the training targets for the policy, with $\epsilon_{\pi}$ measurement noise. $X_{\pi} = [x_1, \ldots, x_N]$, $x_s \in \mathbb{R}^D$, $s = 1, \ldots, N$, are the training inputs. The support points $X_{\pi}$ and the corresponding training targets $y_{\pi}$ are pseudo-training set for the preliminary policy, that means they are adjusted by the algorithm that learns the policy. In this paper, we use PILCO (Ref. 7) 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. 7, 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 TL

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 \( m(x) = 0 \) 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 \( \tilde{x} = [x^T a^T]^T \), 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 \( \alpha^2 \), \( \ell \) and \( \sigma^2 \). 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 by evidence maximization (see Ref. 24 for more details). 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 approach to this problem, by transferring from the solution and the model learned in the source task. Specifically we take information from the policy learned and the transition function, which models the source task. Transferring one or both of them does not lead to a solution. The policy learned in the source task does not necessarily lead to the desired state as in the source task (unless the source and target tasks are the same), but provides a clear advantage over an initial random policy. On the other hand, assuming that the transition function is the same in both tasks is not true, but assuming that both transition functions have common features and just has to be refined in the target task happens to be an effective approach. In Sec. 5.1, we report experiments of how both parts influence the learning in 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 Sec. 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 \( \psi_s \), consisting \( \beta_s \), and the corresponding free parameters for the policy kernel.

In order to have a better approximation of the transition function distribution, we must use information obtained from the source task. Every time the agent repeats the task and interacts with the environment (which is formally known as episode), the tuples and hyper-parameters values from the target task are updated and eventually will replace the source task information. To search for the optimal policy, we used the approach followed in PILCO (Ref. 6) 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. An overall of the proposed algorithm is shown in Algorithm 1.

4.2. Updating knowledge

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. We propose to adjust the hyper-parameters using two approaches: forgetting factor and Bayesian updating.

Let \( \theta = [\alpha^2, \ell, \sigma^2] \) denote a vector of hyper-parameters. Let \( \theta_s \) denote the hyper-parameters transferred from the source task, \( \theta_i \) the hyper-parameters used in the kernel for the target task at episode \( i \), \( \theta_{pi} \) the hyper-parameters learned by evidence maximization in the target task at episode \( i \). In the forgetting factor approach,
we calculate the values of the hyper-parameters in the target task as follows:

\[
\theta_0 = \theta_s, \\
\theta_i = \gamma \theta_{i-1} + (1 - \gamma) \theta_{p_i}, \quad i > 0,
\]

where \( \gamma \in [0,1] \) is the ratio at which previous episode hyper-parameters are incorporated into the kernel function. The value of \( \gamma \) 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.

We also propose another way to update the hyper-parameters by using a Bayesian approach. Bayesian inference has been widely applied to solve problems in many fields of computer science (see Ref. 19), it allows to combine information from several sources taking uncertainty into account.

The Bayesian inference is used to refine the \textit{a priori} probability with new observations. In our problem, this allows to combine information from both tasks (source and target) on each episode by considering the hyper-parameters as continuous Gaussian distributed variables. For continuous random variables, Bayesian inference can update the \textit{a priori} probability distribution with new knowledge from other source, offering a more accurate \textit{a posteriori} distribution. In this approach, treating each hyper-parameter variable as a normally distributed random variable allows us to model uncertainty on each of the tasks. The value learned on the source task is \textit{a priori} knowledge, and we condition it on the value learned on the target task. Then, the value inferred on episode \( k - 1 \) is used as prior for episode \( k \). In the very first episodes of the learning process on the target task, the agent will have very few samples from the transition function. As a result, the model learned from those
samples will not be accurate and the combination process should weight more the transferred hyper-parameters than those learned from the samples.

At each episode $k$, we combine successive approximations of the value of the hyper-parameters learned by evidence maximization $\theta_{p_k}$. We model this by assuming that the learned value of each of the hyper-parameters in the target task has Gaussian noise:

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

while the posterior distribution is:

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

where

$$\sigma_k^2 = \frac{\sigma_p^2 \sigma_k^{2-1}}{\sigma_p^2 + \sigma_k^{2-1}}$$

$$\mu_k = \sigma_k^2 \left( \frac{\mu_k^{2-1}}{\sigma_k^{2-1}} + \frac{\mu_p}{\sigma_p^2} \right)$$

and $\sigma_k^{2-1}$ represents the prior variance and $\sigma_k^2$ the posterior variance.

Fig. 1. This illustration shows a hyper-parameter distribution in the source task (filled dot red), target task (squared blue) and the inferred distribution (continuous green). In the first episodes (a), hyper-parameters on the target task have high uncertainty, so the inferred values are inclined to the source task values. As more samples are gathered in the target task (b), the hyper-parameters learned have lower uncertainty and the inferred values tend to follow the learned distribution of the target task (color online).
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, thus the inferred hyper-parameters distribution is adjusted according to the available data, as shown in Fig. 1. The initial confidence is set according to the amount of samples collected by the agent, using the following equations:

\[
\sigma^2_{k=0} = \frac{1}{n_{\text{source}}},
\]

\[
\sigma^2_{p_k} = \frac{1}{n_{\text{target}}},
\]

where \(n_{\text{source}}\) and \(n_{\text{target}}\) are the number of samples collected in the source and target tasks, respectively.

5. Experiments

Most TL approaches for RL transfer tuples and only work on discrete action spaces, so it is difficult to make a fair comparison. Moreover, transfer algorithms for RL that might work in batch mode (Refs. 17 and 28) 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, a state-of-the-art technique used for learning in continuous spaces (Ref. 7) that is reported to learn several orders of magnitude faster than other algorithms, and that we use as part of our approach to search the policy. The objective of comparing them is to show how transferring knowledge fares as opposed to learning from scratch.

We show experimental results in three different tasks commonly used as benchmarks to compare RL and control algorithms. In all the 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. 7. The initial training set for the transition function was generated by applying actions drawn uniformly from \([-u_{\text{max}}, u_{\text{max}}]\).

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 nontrivial to solve.

For our experiments, we tested the inverted pendulum problem with continuous action and state spaces, as described in Ref. 7. 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 \( x \) is formed by the position \( x \), its velocity \( \dot{x} \), the angle \( \theta \) of the pendulum, and its angular velocity \( \dot{\theta} \). The reward function is expressed as:

\[
r(x) = \exp\left(-\frac{1}{2} ad^2\right)
\]

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

\[
d(x, x_{\text{target}})^2 = x^2 + 2xl \sin \theta + 2l^2 + 2l^2 \cos \theta.
\]

The reward remains close to zero if the distance of the pendulum tip to the target is greater than \( l = 0.6 \text{ m} \). 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, eight hyper-parameters for each of the kernels \( K_i \), 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 \( \alpha^2 \) and noise variance \( \sigma_\epsilon^2 \). 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 \( \gamma \), 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 Eqs. (8) to (11).

A comparison of the learning curves for target tasks is shown in Fig. 2, where we plot PILCO and QTL with different values of \( \gamma \) 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 \( r(x) \) at every step. This experiment shows that proposed TL 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 performance of the learned policy is measured as the average of the cumulative reward in the latest three episodes of the learning process. Our proposed transfer method had the best performance in three of the four tasks (see Table 1).

Both, QTL–PILCO with \( \gamma = 0.9 \) and the Bayesian approach converge faster than PILCO. The more evident results are when the mass of the pendulum is 2.0 kg, where QTL–PILCO converges 13 episodes before learning without transfer, as can be seen in Table 1 and Fig. 2(b). The time to convergence is calculated as the number of
episodes to reach 95% of the performance value. For QTL–PILCO with $\gamma = 1$ and Policy transfer, the algorithm did not converge to a policy after 30 episodes. The values of the hyper-parameters learned by evidence maximization (PILCO) can change drastically during the first iterations of the learning process due to poor samples. However, with our proposals, that gradually incorporate information from

Fig. 2. Learning curves of knowledge transferred to a 1.5x and 4x the mass of the original pendulum. This is 1.5 and 2.0 kg, respectively for a 0.5 kg original mass pendulum. Error bars represent the standard deviation (color online).
the target task, the new hyper-parameters change more smoothly so the learning process can focus more on learning the policy than trying to guess the hyper-parameters without having enough information.

Accumulated reward is an indicator of how good the agent is solving the task since the beginning until the end of the learning process. As this is the case in most of the tasks, it is desirable to avoid mistakes (obtain high rewards) even during learning process. On the other hand, an agent is required not only to learn with a low number of mistakes, but to have a high performance after the learning process. Bayesian approach obtained a higher accumulated reward and a higher performance in most of the tasks, and is preferable to forgetting factor if there is not an indicator of what rate (\(\gamma\)) to choose.

5.2. **Mountain car**

Mountain car (Ref. 27) consists of a car in a valley between two hills and 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.

The problem is normally described using discrete variables for actions (left, zero, right) like in Ref. 27. 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. 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 \(\mathbf{x}_{\text{target}} = (1, 0)\). The reward function is expressed as Eq. (14) 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.

<table>
<thead>
<tr>
<th>Approach</th>
<th>0.8 kg</th>
<th>1.0 kg</th>
<th>1.5 kg</th>
<th>2.0 kg</th>
</tr>
</thead>
<tbody>
<tr>
<td>PILCO ((\gamma = 0))</td>
<td>36.95/228.47/9</td>
<td>35.53/206.09/10</td>
<td><strong>33.32</strong>/218.63/22</td>
<td>29.71/204.41/25</td>
</tr>
<tr>
<td>Hyper-parameters transfer</td>
<td>37.08/236.50/9</td>
<td>33.04/185.99/11</td>
<td>32.98/260.33/22</td>
<td>29.66/240.98/25</td>
</tr>
<tr>
<td>QTL–PILCO (\gamma = 0.9)</td>
<td>37.55/287.44/7</td>
<td>34.97/236.20/9</td>
<td>32.84/589.57/9</td>
<td>29.79/527.65/12</td>
</tr>
<tr>
<td>QTL–PILCO (\gamma = 0.5)</td>
<td>37.20/240.86/9</td>
<td>34.50/217.40/10</td>
<td>32.78/415.95/20</td>
<td>29.90/366.68/21</td>
</tr>
<tr>
<td>QTL–PILCO (\gamma = 1)</td>
<td>37.06/246.41/9</td>
<td>34.59/212.92/10</td>
<td>NC/146.44/Unknown</td>
<td>NC/117.63/Unknown</td>
</tr>
<tr>
<td>QTL–PILCO (\gamma = 0)</td>
<td><strong>37.57</strong>/272.91/8</td>
<td><strong>35.66</strong>/236.43/9</td>
<td>32.68/602.55/10</td>
<td><strong>29.97</strong>/539.79/12</td>
</tr>
<tr>
<td>Bayesian</td>
<td>35.77/214.35/10</td>
<td>32.81/156.76/11</td>
<td>NC/173.90/Unknown</td>
<td>NC/150.64/Unknown</td>
</tr>
</tbody>
</table>

Transfer Learning for Continuous State and Action Spaces

Table 1. Final performance, accumulated reward and time to convergence triplets for each task. Time to convergence is measured by number of episodes to reach 95% of performance. Final performance measured averaging the latest 3 episodes (NC means No convergence thus policy was not learned during 30 episodes). Bayesian approach performs better than other methods in most of the tasks.
For this experiment, we consider as source task the one specified in Ref. 27. 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 Fig. 3 show an improvement in the first two tasks, however, for the third task, the car has enough power to climb the hill without swinging. In this case, we experienced negative transfer (transferring hyper-parameters leads to worst results than without transfer). This is because the transition function is not similar between the source and the target task. Negative transfer is a common phenomenon in TL. In Ref. 17, the authors propose a task compliance metric that measures the probability that the source task $S$ is the model from where the tuples $\hat{T}_{(s,a)}$ might have been generated, this is described by Eq. (15) (for more details see Ref. 17).

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

(15)

where $\hat{U}$ contains all the state–action pairs in the tuples from the target task $\hat{T}$. Applying this measure to the three tasks (see Table 2) it can be observed that the last task is the less similar, so by setting some threshold value $\Lambda_{\text{compl}}$ this negative effect can be avoided.

![Figure 3. Mountain car. Learning curves for target tasks of 50% (a), 150% (b) and 300% (c) car engine power, learned from a car with 100% power available. Error bars represent the standard deviation. In (a) and (b), the learning curves are improved by the transferred knowledge, (c) Illustrates a task where transfer does not improve learning, in such case a task difference measurement would predict that it is better to learn from scratch than transferring (color online).](1460007-14)
Table 2. Task compliance measurement on the three target tasks for the mountain car problem.

<table>
<thead>
<tr>
<th>Engine Power</th>
<th>Task Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>0.64</td>
</tr>
<tr>
<td>150%</td>
<td>0.76</td>
</tr>
<tr>
<td>300%</td>
<td>0.34</td>
</tr>
</tbody>
</table>
5.3. **Quadcopter to helicopter**

This task is the most complex and interesting experiment and requires knowledge to be transferred from a quadcopter to a helicopter. The task consists of finding a policy to get from an initial (on land) position to a desired position, specified by \((x_{\text{target}}, y_{\text{target}}, z_{\text{target}})\) coordinates. So the agent must learn to take off, deal momentarily with the *ground effect,*\(^b\) reach a specific three-dimensional position, and finally keep the vehicle stabilized at that spot. This task is learned in a quadcopter and then transferred 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 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 done on apprenticeship learning where the agent is able to do aerobatic flight (Ref. 1), but it requires 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 aircrafts have the same state variables and same action variables, they behave differently 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 altitude 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 the 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 \(\phi\), pitch \(\theta\), yaw \(\omega\)), 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. (14), with \(a\) set to 0.25 and \(d\) evaluated as \(d(x, x_{\text{target}})^2 = x^2 + y^2 + z^2\). As all of the state and action variables are continuous, this sort of tasks have an infinity number of states.

For simulation purposes, we use V-REP (Ref. 25), which is a robotics simulator, where the quadcopter and helicopter dynamics models run. Figure 4 shows the learning curves for the helicopter task. We plot the learning curves for PILCO and QTL–PILCO, as well the reward obtained by the control algorithm provided by the simulator. 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

\(^b\)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.
V-REP’s autopilot which tends to overshoot. The comparison of PILCO and QTL approaches in terms of convergence, performance and accumulated rewards is shown in Table 3, where QTL–PILCO Bayesian accumulates the highest reward. Our approach reaches a correct policy in 19 rounds while PILCO uses 24 rounds to learn the task, a 20.8% reduction in time. The final performance is very similar in all cases.

Transfer in this challenging scenario shows empirically that the proposed approach is useful in real problems with high dimensionality and where the transition function is very sensitive to inaccuracies (the behavior of the autonomous helicopter is very sensitive to the variations in the actions).

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.

Table 3. Time to convergence is measured in number of episodes to reach 95% of performance. Performance is total reward averaged from the last three episodes.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Time to Convergence</th>
<th>Performance</th>
<th>Accumulated Rewards</th>
</tr>
</thead>
<tbody>
<tr>
<td>PILCO ($\gamma = 0$)</td>
<td>24</td>
<td>132.5</td>
<td>1890.4</td>
</tr>
<tr>
<td>QTL–PILCO ($\gamma = 0.1$)</td>
<td>21</td>
<td>130.61</td>
<td>2096.8</td>
</tr>
<tr>
<td>QTL–PILCO ($\gamma = 0.5$)</td>
<td>22</td>
<td>130.95</td>
<td>2123.9</td>
</tr>
<tr>
<td>QTL–PILCO ($\gamma = 0.9$)</td>
<td>21</td>
<td>131.53</td>
<td>2153.5</td>
</tr>
<tr>
<td>QTL–PILCO Bayesian</td>
<td>19</td>
<td>131.99</td>
<td>2225.4</td>
</tr>
</tbody>
</table>

Fig. 4. 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 (color online).
This approach is suitable for several tasks, including those with many variables in state and action spaces.

In all the tests, the Bayesian approach leads to slightly better results than using a forgetting factor, but its main advantage resides in not depending on parameters, like $\gamma$.

6. Conclusions

In this paper, we have presented a TL approach for RL with continuous state and action spaces. It works by refining the transition function model in the target task (modeled as a GP) using qualitative knowledge from the source task, like smoothness, 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 hyper-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 TL 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. A more sophisticated Bayesian filter, like the one presented in Ref. 8, could be used, modified to represent the transition function as GP in order to make Bayesian inference more robust. We would also like to develop methods to synthesize tuples in the target task to also adjust the posterior distribution of the transition function.

References


---

**Omar García** received his B.Sc. and M.Sc. degrees in Computer Sciences from “Benemerita Universidad Autonoma de Puebla” and “Instituto Nacional de Astrofisica, Optica y Electronica” (INAOE) respectively, in Puebla, Mexico. He is currently a Ph.D. student at INAOE in Mexico. His recent research focuses on reinforcement learning and transfer learning.

**Enrique Munoz de Cote** is an Assistant Professor of Computer Science at INAOE (Mexico). He has published papers on machine learning, computational game theory and multiagent systems. He received a Ph.D. in Computer Science from Politecnico di Milano (Italy) and a postdoc from Southampton University (UK). He is currently working on the computational aspects of economic theory.

**E. F. Morales** received his Ph.D. in Computer Science from the Turing Institute, University of Strathclyde, in Scotland. He has been responsible for 20 research projects sponsored by different funding agencies and has published more than 100 articles in journals, book’s chapters and conference proceedings.

He is currently the head of Computer Science at the National Institute of Astrophysics, Optics and Electronics (INAOE) in Mexico where he conducts research in Machine Learning and Robotics.