Reinforcement Learning Transfer using a Sparse Coded Inter-Task Mapping

Haitham Bou Ammar[1], Matthew E. Taylor[2], Karl Tuyls[1], and Gerhard Weiss[1]

[1] Department of Knowledge Engineering, Maastricht University
The Netherlands
[2] Department of Computer Science, Lafayette College
USA

Abstract. Reinforcement learning agents can successfully learn in a variety of difficult tasks. A fundamental problem is that they may learn slowly in complex environments, inspiring the development of speedup methods such as transfer learning. Transfer improves learning by reusing learned behaviors in similar tasks, usually via an inter-task mapping, which defines how a pair of tasks are related. This paper proposes a novel transfer learning technique to autonomously construct an inter-task mapping by using a novel combinations of sparse coding, sparse projection learning, and sparse pseudo-input gaussian processes. Experiments show successful transfer of information between two very different domains: the mountain car and the pole balancing task. This paper empirically shows that the learned inter-task mapping can be used to successfully (1) improve the performance of a learned policy on a fixed number of samples, (2) reduce the learning times needed by the algorithms to converge to a policy on a fixed number of samples, and (3) converge faster to a near-optimal policy given a large amount of samples.

1 Introduction

Reinforcement Learning (RL)is a popular framework that allows agents to solve sequential-action selection tasks with minimal feedback. Unfortunately, RL agents may learn slowly in large or complex environments due to the amount of computational effort and/or experience needed to attain an acceptable performing policy. Transfer Learning [17] (TL) is one technique used to cope with this difficulty by providing a good starting prior for the RL agent attained in a related source task.

The source task can differ from the target task in many ways. If the tasks have different representations of state or action spaces, some type of mapping between the tasks is required. This inter-task mapping matches each state/action pair of the source task to its corresponding state/action pair in the target facilitating transfer. While there have been a number of successes in using such a mapping, the approaches are typically hand-coded and may require substantial human knowledge [17, 19]. Our contributions in this paper are twofold. First, we propose a novel scheme to automatically learn an inter-task mapping between two tasks. Second, we introduce the new *Transfer Least Squares Policy Iteration* (TrLSPI) algorithm for transfer between tasks of continuous state spaces and discrete action spaces.

To the best of our knowledge, this paper shows the first successful attempts to automatically transfer between RL benchmark tasks that are very different. Namely, we conduct experiments to automatically transfer from the Mountain Car to the Pole Balancing problem. Our results show (1) improved performance on a fixed number of samples, (2) a reduction in the convergence times to attain a policy on a fixed number of samples, and (3) a reduction in the time needed to attain a near-optimal policy on a large amount of samples.

The rest of the paper proceeds as follows. Related work is discussed next in Section 2. Background information is presented in Section 3. Section 4 describes how an inter-task mapping can be learned between two tasks by leveraging sparse coding, sparse projection learning and sparse pseudo-input gaussian processes. In Section 5, we introduce our novel TrLSPI algorithm showing how the learned mapping can be used to transfer information between a source task and target task. Experiments of transfer between two very different tasks are presented in Section 6. Section 7 presents a discussion on the scope an applicability of our framework. Section 8 concludes and reflects upon interesting future work directions.

2 Related Work

In the past few years there has been a significant amount of work done in transfer learning for RL tasks. This section outlines the most related work and contrasts it with the work in this paper.

The majority of current transfer learning work in RL assumes that either 1) the two agents are very similar and no mapping is needed, or 2) the inter-task mapping is provided by a human. For instance, [19] transfers advice and [17] transfers Q-values — both methods assume that a mapping between the state and action variables in the two tasks has been provided. Another approach is to frame different tasks as having a shared *agent space* [5], so that no explicit mapping is needed. However, this requires that the agent acting in both tasks share the same actions and a human to map new sensors back into the agent space. The primary contrast between these methods and the current work is that we are interested in *learning* a mapping between states and actions in pairs of tasks, rather than assuming that it is provided or unnecessary.

Our previous work [1] required the presence of hand-coded features shared between two tasks in order to automatically learn the inter-task mapping. This work extends the previous approach to overcome the need for a *predefined common subspace* to determine the inter-task mapping.

There has also been recent work that approaches fully autonomous transfer. For example, semantic knowledge about state features between two tasks may be used [6, 9], background knowledge about the range or type of state variables can be used [14, 18], or transition models for each possible mapping could be generated and tested [15]. Transfer learning has also been successful across different domains, e.g., using a simple discrete, deterministic task to improve learning on a complex, continuous, noisy task [16]. However, there are currently no general methods to learn an inter-task mapping without requiring (1) background knowledge, or (2) an expensive analysis of an exponential number of inter-task mappings. This paper overcomes these problems by automatically discovering high-level features and using them to conduct transfer within reasonable time requirements.

Unlike all other existing methods (to the best of our knowledge) and complementary to our previous work [1, 15, 16], we assume differences among all the variables of

Markov Decision Processes describing the source and target tasks and focus on learning an *inter-state mapping*, rather than a state-variable mapping. Additionally, our framework can use state-dependent action mappings, allowing flexibility that other existing algorithms do not.

3 Background

This section provides the reader with a short overview of *sparse coding*, *reinforcement learning*, *gaussian processes*, *transfer learning* and other learning methods used in this paper.

3.1 Reinforcement Learning (RL)

In an RL problem, an agent must decide how to sequentially select actions to maximize its expected long term reward [3, 13]. Such problems are typically formalized as Markov decision processes (MDPs). An MDP is defined by $\langle S, A, P, R, \gamma\rangle$, where S is the (potentially infinite) set of states, A is the set of all possible actions that the agent may execute, $P : S \times A \to S$ is a state transition probability function defining the transition dynamics, $R : S \times A \to \mathbb{R}$ is the reward function measuring the performance of the agent, and $\gamma \in [0, 1)$ is the discount factor. A policy $\pi : S \to A$ is defined as a probabilistic mapping from a state to an action, where $\pi(a|s)$, represents the probability of choosing an action a in a state s. The goal of an RL agent is to improve its policy, potentially reaching the optimal policy $\pi^\star$ represented by taking greedy actions in the optimal Q-function:

$$Q^\star(s, a) = \max_\pi E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)|s = s_0, \pi] \tag{1}$$

In tasks with continuous state and/or action spaces, the Q functions and policies cannot be represented in a table format, typically requiring sampling and function approximation techniques. This paper uses one such technique, *Least Squares Policy Iteration* (LSPI), which will be explained in Section 5.2.

3.2 Transfer Learning in RL Tasks

Typically, when using Transfer Learning (TL) in RL tasks, there is a source and a target task [17]. When the source and the target tasks are related, transferring a learned source behavior should improve learning in the target task by providing an informative prior. The prior will restrict the exploration in the target task by biasing the agent so that it chooses actions that are better than random exploration, reducing the target task learning times and improving the overall performance. In our formulation, each of these tasks is defined as an MDP which is a tuple of $\langle S^{(i)}, A^{(i)}, P^{(i)}(s, a), R^{(i)}, \gamma^{(i)}\rangle$ for $i \in \{1, 2\}$ where $S^{(i)}$, $A^{(i)}$, $P^{(i)}(s, a)$, $R^{(i)}$ and $\gamma^{(i)}$ represent the state spaces, action spaces, state transition probabilities, reward functions and discount factors for each of the source ($i = 1$) and target ($i = 2$) tasks.

The source and the target task may differ in their state spaces and/or action spaces (as well as other components of the MDP). If transfer is to be useful when such differences exist, an inter-task mapping relating these state-action spaces differences [17] can be used. Traditionally, such a mapping was thought to be a one-to-one mapping between the state/action variables representing the tasks [17]. This paper instead considers a mapping that relates state-action successor state triplets from the source with the target task. Mathematically, $\chi : S_s \times A_s \times S_s \to S_t \times A_t \times S_t$, where S and A represent the state space and the action space of the source and the target task, respectively. This paper's inter-task mapping is more than just a one-to-one mapping between the state and/or action spaces of the MDPs. It also includes other terms that are automatically discovered by our global approximators, which ultimately enhance the transfer approach.

The main focus in this paper is the automatic discovery of an inter-task mapping to enable transfer. The upcoming sections will further clarify the need for such a mapping as well as describe our novel framework.

3.3 Sparse Coding

Sparse coding (SC) [8] is an unsupervised learning technique used to find a high-level representation for a given set of unlabeled input data. It does this by discovering a succinct over-complete basis for the provided data set. Given m k-dimensional input vectors, $\boldsymbol{\zeta}$, SC finds a set of n basis vectors, $\mathbf{b}$, and activations, a, with $n > k$ such that $\boldsymbol{\zeta} \approx \sum_{j=1}^{n} a_j^{(i)} \mathbf{b}_j$, where i and j represent the number of input data patterns and number of bases, respectively. SC begins by assuming a Gaussian and a sparse prior on the reconstruction error ($\boldsymbol{\zeta}^{(i)} - \sum_{j=1}^{n} a_j^{(i)} \mathbf{b}_j$) and on the activations, leading to the following an optimization problem:

$$\begin{aligned} \min_{\{\mathbf{b}_j\},\{a_j^{(i)}\}} & \sum_{i=1}^{m} \frac{1}{2\sigma^2} ||\boldsymbol{\zeta}^{(i)} - \sum_{j=1}^{n} \mathbf{b}_j a_j^{(i)}||_2^2 \\ & + \beta \sum_{i=1}^{m} \sum_{j=1}^{n} ||a_j^{(i)}||_1 \\ s.t. \ & ||\mathbf{b}_j||_2^2 \leq c, \forall j = \{1, 2, \ldots, n\} \end{aligned} \tag{2}$$

The problem presented in Equation 2 is considered to be a "hard" optimization problem as it is not jointly convex (i.e, in the activations and bases). However, fast and efficient optimization algorithms exist [8] and were used in our work.

3.4 Gaussian Processes

Gaussian processes (GPs) constitute a research field by themselves. It is beyond the scope of this paper to fully detail the mathematical framework. This section briefly explains GPs and refers the reader elsewhere [11] for a more in-depth treatment.

GPs are a form of supervised learning used to discover a relation between a given set of input vectors, $\mathbf{x}$, and set of output pairs, $\mathbf{y}$. As opposed to normal regression techniques that perform inference in the weight space, GPs perform inference directly in the functional space, making learning simpler. Following existing notation [11], if a function is sampled according to a GP we write:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}^{'})), \tag{3}$$

where $m(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}^{'})$, represent the mean and covariance function that fully specify a GP.

Learning in a GP setting involves maximizing the marginal likelihood:

$$\log p(\mathbf{y}|\mathbf{x}) = -\frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}log|\mathbf{K}| - \frac{n}{2}log2\pi. \tag{4}$$

Maximizing Equation 4 may be computationally complex as we must invert the covariance matrix $\mathbf{K}$, which is of order of $O(N^3)$, where N is the number of input points. Therefore, we use a fast learning technique, *sparse pseudo-input gaussian processes* (SPGP), as proposed elsewhere [12].

SPGPs aim to reduce the complexity of learning and prediction in GPs by parametrizing the regression model with $M << N$ pseudo-input points, while still preserving the full Bayesian framework. The covariance of the GP model is parametrized by the location of the $M << N$ pseudo-inputs. Existing results [12] show a complexity reduction in the training cost (i.e., the cost of finding the parameters of the covariances) to $O(M^2N)$ and in the prediction cost (i.e., prediction on a new set of inputs) to $O(M^2)$.

4 Learning the Inter-Task Mapping

In order to automatically construct an inter-task mapping, χ, this paper proposes a novel framework using sparse coding, a L_1 projection scheme, and *sparse pseudo-input gaussian processes*. Each of these methods is necessary to solve a problem that is inherent to TL in RL tasks. We approach the problem of learning the inter-task mapping, χ, as a supervised learning problem. As χ is a mapping relating state-action triplets from the source with the target task, related triplets should be provided as training data points. Unfortunately, this is itself a hard problem — it is not trivial for the user to describe what state triplets in the source task correspond to what in the target task. We therefore approach this problem by automatically transforming the problem spaces (i.e., the state-action spaces of the two tasks) into a higher representational space through SC, projecting the target task data onto those attained bases and then using a Euclidean distance measure to gauge similarity (Section 4.1). At this stage, the data set is provided to a regressor to construct the inter-task mapping. Many regression techniques could be applied to the approach but we chose to use a non-parametric approximation scheme because of its generalization advantages.

Our framework can conceptually be split into three essential parts. The first is the dimensional unification of both the source and target task state-action spaces of the MDPs. The second is the automatic discovery of a high dimensional informative space for the source task. This is achieved through SC, as described in Section 4.1, ensuring that transfer is conducted in a high representational space of the source task. In order to use a similarity measure among different patterns, the data should be present in the same space. That is why the target task samples still need to be projected to the attained high representational space of the source. This is done using sparse projection learning, described in Section 4.2. The third and final step is to approximate the inter-task mapping via a non-parametric regression technique, explained in Section 4.3.

4.1 Sparse Coding Transfer for RL

As described in Section 3.3, SC is an efficient way to discover higher level information in an unlabeled data set. We use SC to solve two inherent problems in transfer learning for RL tasks. The first is to unify the dimensions of the state action spaces of the two different MDPs. The second is to discover a higher level representation for the attained bases and activations of the source task state-action spaces. This step guarantees that our scheme works with the "best" available representation/information space of the source task.

Unifying the Source and Target Dimensions Our problem commences by first unifying the dimensions of the state action spaces of the two MDPs, an essential step for discovering the inter-task mapping. After this step has finished, any existing TL in RL technique may be used. However, this paper goes further and proposes a new transfer framework based on the attained bases and activations, described in Section 5.

This "dimensional unification" process is described in Algorithm 1. In short, Algorithm 1 sparse codes random samples from the source task, constrained by learning the same number of bases (d_t) as the target task.

The algorithms proposed elsewhere [8] solve Equation 6 on line 3 of Algorithm 1. After this stage is done, new activations and bases describing the samples are attained.[3] Note that, these newly attained samples—described as a linear combinations of the bases and activations (**Ab**)—do not yet relate anything to the target task ones. The target task samples still need to be projected towards these bases. This is done as described in Section 4.2.

Algorithm 1 Sparse Coding Transfer Reinforcement Learning

```
Require: Source MDP samples $\{\langle s_s, a_s, s_s^{'}\rangle\}_{i=1}^{m}$, target MDP samples $\{\langle s_t, a_t, s_t^{'}\rangle\}_{j=1}^{f}$
1: Calculate $d_s$ and $d_t$ which are the dimensions of each of the state action spaces of the MDPs
2: Sparse code the source by solving:
3:
```

$$\min_{\{\mathbf{b}_j\},\{a_j^{(i)}\}} \sum_{i=1}^{m} \frac{1}{2\sigma^2} ||\langle s_s, a_s, s_s^{'}\rangle^{(i)} - \sum_{j=1}^{d_t} b_j a_j^{(i)}||_2^2 \tag{5}$$

$$+\beta \sum_{i=1}^{m}\sum_{j=1}^{d_t} ||a_j^{(i)}||_1$$

$$s.t.\ \ ||b_j||_2^2 \leq c, \forall j = \{1, 2, \dots, d_t\}$$

```
4: Solve the problem of Equation 6 using the algorithm proposed in [8]
5: Return the activation matrix ($\mathbf{A} \in \mathbb{R}^{m\times d_t}$) and the bases ($\mathbf{b} \in \mathbb{R}^{d_t\times 1}$)
```

[3] Please note that while writing Algorithm 1 it was assumed that the dimensions of the source task d_s are lower than those of the target task d_t. But it is worth noting that it works as well for the other cases with no restrictions.

After Algorithm 1 is finished, new features in the source task state action spaces are discovered. This is reasonable as TL typically transfers between a low dimensional source task to a high dimensional target task. Here, SC is determining new bases that are of a higher number than the original state action dimensions in the source task. If successful, new patterns and representations are discovered in the source task state-action spaces. These new features describe new representations not anticipated by the original dimensions. Therefore, this new information can be used to help and guide the transfer learning scheme.

High Information Representation After dimensional unification, as described in the previous section, SC is again used to discover a succinct higher informational/representational bases of the activations than the unified dimensional spaces. This insures that our transfer approach operates in the "richest" space described through the samples. This is done in a similar framework to that in Section 4.1 and is described in Algorithm 2.

Algorithm 2 Succinct High Information Representation of MDPs

```
Require: Activations acquired through Algorithm 1, number of new high dimensional bases $d_n$
1: Represent the activations in the $d_n$ bases by solving the following problem using the algorithm in [8]:
2:
```

$$\min_{\{\mathbf{z}_j\},\{c_j^{(i)}\}} \sum_{i=1}^{m} \frac{1}{2\sigma^2} ||\langle \mathbf{a}_{1:d_t}\rangle^{(i)} - \sum_{j=1}^{d_n} \mathbf{z}_j c_j^{(i)}||_2^2 + \beta \sum_{i=1}^{m}\sum_{j=1}^{d_n} ||c_j^{(i)}||_1 \quad s.t. \ ||\mathbf{z}_j||_2^2 \leq o, \forall j = \{1, 2, \dots, d_n\} \tag{6}$$

```
3: return activations $\mathbf{C} \in \mathbb{R}^{m \times d_n}$ and bases $\mathbf{z} \in \mathbb{R}^{d_n \times 1}$
```

The idea presented by Algorithm 2 is to sparse code the activations, representing the original samples of the MDPs, to a higher representational space, d_n.[4] This stage should guarantee that we project the samples of the source task MDP into a high informational space where a similarity measure can be used to find a relation between the source and target task triplets. Noting that there are no restrictions on the number of bases to be determined: unneeded bases have an activation of zero once the SC problem has been solved.

At this stage, the source state action spaces are described in a rich informational space determined by the newly discovered bases and activations. The next step is to project the target task samples to that space described by $\mathbf{Z}$ so that triplets can be ordered and the inter-task mapping approximated.

[4] In our experiments we have set d_n to be 100, a relatively high number.

4.2 L_1 Sparse Projection Learning

Once the above stages have finished, the source samples are described via the activations generated in Algorithm 2. However, target task samples still have no relationship to the learned activations. In other words, the bases and activations that have been attained successfully describe high informational patterns and representations in the source task state-action spaces but do not represent the target state-action spaces. Since we are seeking a similarity correspondence between the source and target task triplets, the target task samples should be represented in the same high informational space.

Therefore, the next step is to learn a sparse projection to project the target task samples onto the **Z** basis representing the source task MDP. In other words, the goal now is to learn a sparse projection that is capable of representing the random target task samples as a combination of some activations, automatically learned, and the **Z** bases generated by Algorithm 2. The overall scheme is described in Algorithm 3, where the activations are learned by solving the L_1 regularized least squares optimization problem of Equation 7. This optimization problem guarantees that the attained activations are as sparse as possible and is solved using the interior point methods [4].

At this stage all the samples from both the target and source task are projected to the same space described by the sparse coded vectors **Z**. The next step will be to order the data points from both the source and the target task so to approximate the inter-task mapping.

Algorithm 3 Reflecting Target Task Samples

```
Require: Sparse coded bases Z generated by Algorithm 2, target MDP samples $\{\langle s_t, a_t, s_t^{'}\rangle\}_{i=1}^{f}$
1: for $i = 1 \rightarrow f$ do
2:     Represent the target data patterns in the sparse coded bases, Z, by solving:
3:
4: end for
5: return activations $\boldsymbol{\Phi}$
```

$$\hat{\phi}^{(i)}(\langle s_t, a_t, s_t^{'}\rangle) = \arg\min_{\phi^{(i)}} ||\langle s_t, a_t, s_t^{'}\rangle - \sum_{j=1}^{d_n} \phi_j^{(i)} z_j||_2^2 + \beta||\phi^{(i)}||_1 \tag{7}$$

4.3 Similarity Measure and Inter-Task Mapping Approximation

As mentioned previously, we tackle the problem of learning an inter-task mapping via supervised learning. Since χ maps triplets from the source task to their corresponding triplets in the target task, the problem at this stage is to attain the training patterns to approximate χ.

After reaching the rich space representing the random samples of the 2 MDPs (i.e., **Z**), a Euclidean distance measure is used to compare triplets, providing a data set to the regressor (i.e, SPGPs) to approximate the inter-task mapping χ. This similarity measure is used to determine the correspondence of the source and target tasks triplets. Once applied, the similarity measure will seek the triplets of the source task closest to those of the target task and map them together as being inputs and outputs for the regression algorithm, respectively. This is shown on line 2 of Algorithm 4. Since the similarity measure is used in the sparse coded spaces, the distance is calculated using the attained activation (**C** and $\boldsymbol{\Phi}$) rather than the samples themselves. Therefore, the scheme has to trace the data back to the original dimensions of the state-action pairs of the MDPs.

There are few restrictions on the function approximation techniques that could be used. We use nonparametric regression with *sparse gaussian processes* technique [12]. We prefer *sparse gaussian processes* rather than normal *gaussian processes* regression technique as the latter may have problems dealing with large data sets. To clarify, consider the learning phase of a GP that involves maximizing Equation 4. It is clear that the inversion of the covariance matrix, **K**, is required on each iteration with complexity $O(n^3)$, where n is the number of samples. Additionally, the maximiztion algorithm (Conjugate Gradient Descent [10]) may get stuck in a local maximum of Equation 4, a common problem in function approximation schemes and maximization problems.

Algorithm 4 Similarity Measure & Inter-Task mapping approximation

```
Require: Sparse coded basis Z, sparse coded activations of the source task $\mathbf{C} \in \mathbb{R}^{m \times d_n}$, pro-
    jected target task activations $\phi \in \mathbb{R}^{m \times d_n}$
1: for all $\phi$ do
2:    Calculate the closest activation in C minimizing the Euclidean/similarity distance mea-
      sure.
3: end for
4: Correspond the triplets with the minimum similarity measure as being inputs and outputs to
   create a data set $\mathcal{D}$
5: Approximate the inter-task mapping, $\chi$ using SPGPs
6: return The approximated inter-task mapping $\chi$
```

5 Transfer Scheme

Assuming there exists a "good enough" policy, $\pi_s^\star$ for the source task, we propose a novel transfer algorithm for pairs of tasks with continuous state spaces and discrete action spaces, titled Transfer Least Squares Policy Iteration.

This section describes the novel transfer scheme and reflects on the details and technicalities of the approach. It starts by describing a well-known reinforcement learning algorithm (LSPI), that our novel transfer algorithm builds on. Then clarifies all the technicalities involved in the proposed TrLSPI algorithm.

5.1 Least Squares Policy Iteration

LSPI [7] is an approximate RL algorithm that is considered an actor/critic method. LSPI is composed of two parts. The first is an evaluation step, *Least Squares Temporal Difference Q-learning* (LSTDQ) and the second is a policy improvement step. In LSTDQ the algorithm will update the weights representing the policy so that the new parameters minimize certain error criteria. For example, the LSTDQ could be set to minimize the

Bellman residual error of the projected Bellman equations. Once this step has finished, LSPI uses the attained weights to improve the policy by taking greedy actions in the approximated Q-function.

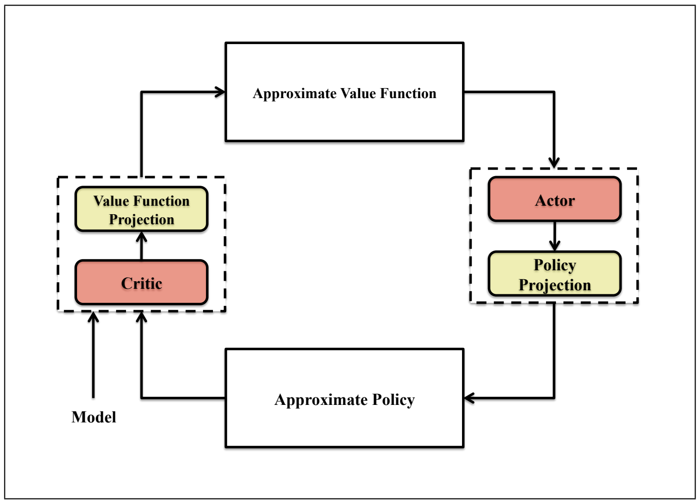

Fig. 1. Least Squares Policy Iteration schematic [7].

Figure 1 highlights the actor and critic organization of LSPI. Since LSPI uses function approximators to represent the Q-functions and/or policies, there also exist two projection phases for both the Q-function and the policy, as can been seen in the schematic. A more thorough treatment may be found elsewhere [7].

5.2 Transfer Least Squares Policy Iteration (TrLSPI)

TrLSPI, Algorithm 5, can be split into two sections. The first determines χ (Section 4), using source task samples[5] from $\pi_s^\star$. The second provides those samples for the evaluation phase of the LSPI algorithm (LSTDQ), to learn a policy for the target task. The intuition here is that if the tasks are similar and if the inter-task mapping is "good enough," then those samples will bias the target task controller towards choosing good actions and restricting its area of exploration and reducing learning times and increasing performance.

Provided that the tasks are related, Algorithm 5 is capable of attaining a good starting behavior for the target task. The performance of this policy depends on the state space region where those samples were provided. In other words, it is not possible to achieve near-optimal performance with a small number of samples that are in regions far from the goal state.[6] Therefore, if the agent has to seek a near-optimal policy, then

[5] If using an approximate RL algorithm in the source task, the policy would instead by near-optimal.

[6] This is a problem that is inherentt to LSPI and is not due to the TrLSPI algorithm.

Algorithm 5 TrLSPI

```
Require: Source MDP samples $\{\langle s_s, a_s, s_s^{'}\rangle\}_{i=1}^{m}$, target MDP samples $\{\langle s_t, a_t, s_t^{'}\}$, number for
    re-samples $n_s$, a (near-)optimal policy for the source system $\pi_s^\star$, state action basis functions
    for the target task $\psi_1, \dots, \psi_k$
 1: Unify the dimensions using Algorithm 1
 2: Discover high informational representation using Algorithm 2
 3: Sparse project the target task samples using Algorithm 3
 4: Use a similarity measure to attain the data set and approximate $\chi$ using Algorithm 4
 5: Randomly sample $n_s$ source task triplets $\langle s_s, a_s, s_s^{'}\rangle_{i=1}^{n_s}$ greedily in the optimal policy $\pi_s^\star$,
    set of state-dependent basis function $\psi_1, \dots, \psi_k : S_t \times A_t \rightarrow \mathbb{R}$
 6: for $i = 1 \rightarrow n_s$ do
 7:     Find the corresponding target task triplets as $\langle s_t^{(i)}, a_t^{(i)}, s_t^{(i)\prime}\rangle = \chi(\langle s_s^{(i)}, a_s^{(i)}, s_s^{(i)\prime}\rangle)$
 8: end for
 9: Find the closest triplet in the initial samples to the ones predicted by $\chi$
10: Use LSTDQ described by [7] to evaluate those samples
11: Learn and improve policy till convergence using LSPI [7]
12: return Learned policy $\pi_{target}^\star$
```

either a new sampling step using the current policy should be added to Algorithm 5, or a large amount of samples should be provided. It is worth noting that it is not necessary for the algorithm to be provided by a model for the system to perform that sampling. A black box generative model taking inputs being states and actions and producing outputs of successor states and rewards is sufficient.

6 Experiments & Results

Two very different tasks were chosen to evaluate the proposed framework, the RL benchmark tasks Mountain Car (MC) and Pole Balancing (see Figure 4).

The control objective of MC, the source task, is to drive the car up the hill (Figure 2). The difficulty is that gravity is stronger than the car's motor—even at maximum throttle the car can not directly reach the top of the hill. The solution is to first move away from the target to the opposite side of the hill and then accumulate enough energy to reach the top of the hill. The dynamics of the car are described via two continuous state variables $(x,\ \dot{x})$ representing the position and velocity of the center of gravity of the car, respectively. There are three actions: maximum throttle forward (+1), zero throttle (0), and maximum throttle reverse (-1). The car is rewarded by $+1$ once it reaches the top of the hill, -1 if it hits the wall, and zero elsewhere. At the end of each episode the start state is randomly initialized at the bottom of the hill.

The target task is the Pole Balancing problem described in Figure 3. The control goal of the Pole Balancing system is balancing the pole in an upright position (i.e., $\theta = \dot{\theta} = 0$). The allowed actions are (+1) for full throttle right and (-1) for full throttle left. The reward function of the system consists of two parts: (1) $cos(\theta)$, which yields its maximum value of $+1$ at the upright position of the pole, and (2) -1 if the cart hits the boundaries of the track. The angle was restricted to be within $|\theta| < \frac{\pi}{9}$ while the position was restricted to $|x| < 3$ and the start state was randomly chosen within that interval.

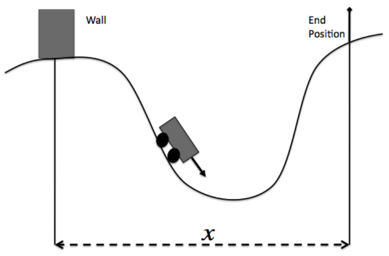

Fig. 2. Mountain Car

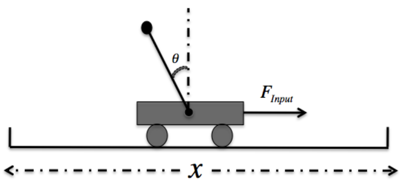

Fig. 3. Pole Balancing

Fig. 4. Mountain Car to Pole Balancing Transfer

As is clear from the description, the two MDPs representing the tasks are significantly different. The source and target task have different state spaces, action spaces, transition probabilities, and reward functions. No previous work can learn to autonomously transfer between such different tasks.

Our framework requires an optimal policy in the MC source task, $\pi^{\star}_{MC}$. SARSA(λ) [13] is used to learn $\pi^{\star}_{MC}$. The learned policy is then used to randomly sample different numbers of states, to be used by χ. We started with $5,000$ and $2,000$ randomly sampled states (using a random policy) for the MC and the Pole Balancing, respectively. These samples were used by the algorithm described in Section 4 to attain the inter-task mapping χ. After χ has been learned, different amounts of samples were sampled from the source task using the optimal policy $\pi^{\star}_{MC}$. Specifically, we have sampled 500, $1,000$,... $20,000$ states as input to the TrLSPI algorithm to measure performance and convergence times.

Our results show 1) an increase in the performance on a fixed number of samples, 2) a decrease in the convergence time when using a predefined number of samples, and 3) a decrease in the time required to learn a near-optimal policy.

6.1 Performance on a Fixed Number of Samples

The first sets of experiments we conducted were to measure the performance in the target task, given a fixed number of source task transferred samples. Namely, we measured the number of successful control steps in the target task given a fixed number of $500, \ldots 20,000$ transferred MC samples. We compared the performance to a normal LSPI (i.e., random sampling) learner in the target task. The graph in Figure 5 summarizes the attained results and clearly shows an increase in the number of control steps in the case of the transferred samples compared to a random sampling scheme. For example it can be seen that at a small number of samples, e.g. $2,000$, our transfer scheme was able to attain an average of 600 control steps with about 400 for the random case. This performance increases with the number of samples to reach 800 steps at $4,000$ transferred samples. The random case needed to be provided by $9,000$ samples to attain an 800 step performance. Finally, the transferred algorithm and the random selection

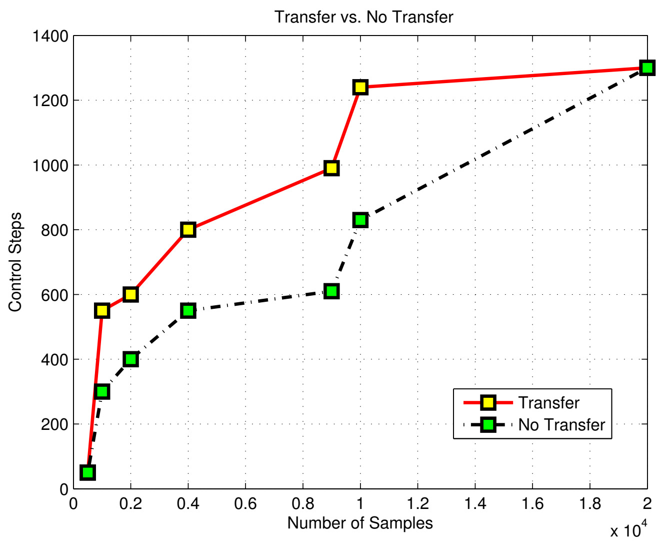

Fig. 5. Transfer Results on the Pole Balancing task.

scheme seem to converge, on a large amount of samples (20,000) to the same number of control steps, about $1,300$. This leads to the following conclusion:

Conclusion 1: TrLSPI has provided a better distribution of samples compared to sampling with a random policy in the target task.

6.2 Convergence Times on Fixed Number of Samples

The second improvement we report is the decrease in the convergence times, represented by the number of iterations in LSPI, provided a fixed number of transferred sampled. This time was measured by comparing the convergence times of TrLSPI and LSPI on a fixed number of transferred and random samples, respectively. To clarify, LSPI was able to converge faster once provided the transferred samples compared to a random sample data set. For example, it took LSPI 7 iterations to converge provided $5,000$ transferred samples with 12 iterations for the random case. Further the algorithm converged within 14 iterations provided $20,000$ transferred samples while it took it about 21 for the random case.

Conclusion 2: TrLSPI converged faster provided a fixed amount of samples.

6.3 Convergence Times to a Near-Optimal Policy

The last performance measure we tested was the amount of time required by TrLSPI to converge to a near-optimal policy, which was compared to normal LSPI learners in the target tasks. LSPI was able to converge to an acceptable policy within 22.5 minutes after being provided a random data set, compared to 17 minutes with the transferred data set[7]. Calculating χ took an addition 3.7 minutes.

[7] Our experiments were performed on a 2.8 Ghz Intel Core i7.

Conclusion 3: TrLSPI converged faster to a near-optimal policy compared to a random selection scheme.

7 Discussion

The proposed TL framework is compatible with other sample-based model-free learning methods and can be used on a variety of RL tasks with continuous state spaces and discrete action spaces. The framework has the advantage of automatically finding the inter-task mapping using SC and any "good" regression technique. But there is one potential weakness, discussed next.

Our framework should work correctly when the two tasks at hand are *semantically* similar, as the rewards of the two systems were not taken into account in the explained scheme. For instance, consider the transfer example between the same robot but with opposite rewards.

Our mapping scheme from Section 4, once applied, will produce a one-to-one mapping from the source to the target task. In other words, since the two tasks have the same state and action spaces, the mapping that will be one-to-one, mapping the same state action successor state triplets of the two tasks together. Therefore, the transition of the robots to the rewardable/ un-rewardable states will map together. Since the optimal policies of the two robots are opposite, it is easy to see that in this case the target task has been provided with a "bad" biased starting policy which will decrease the agents performance rather than enhancing it. Such *negative transfer* is a well-known problem in TL for RL tasks. This paper's approach may be able to avoid this problem once rewards are added to the similarity measure, used to generate the training set to approximate the inter-task mapping χ. However, this enhancement will be left to future work.

8 Conclusions & Future Work

This paper has presented a novel technique for transfer learning in reinforcement learning tasks. Our framework may be applied to pairs of reinforcement learning problems with continuous state spaces and discrete action spaces. The main contributions of this paper are (1) the novel method of automatically attaining the inter-task mapping, χ and (2) the new TrLSPI algorithm for tasks with continuous state spaces and discrete actions. We approached the problem by framing the approximation of the inter-task mapping as a supervised learning problem that was solved using *sparse pseudo input gaussian processes*. *sparse coding*, accompanied with a similarity measure, was used to determine the data set required by the regressor for approximating χ. Our results demonstrate successful transfer between two very different tasks, the mountain car to the Pole Balancing task. Success was measured both in an increase in learning performance as well as a reduction in convergence time.

There are many exciting directions for future work. First, different distance metrics should be compared and their effects on the overall performance of the algorithm measured. Second, the distance measure will be improved by incorporating the rewards in the framework, helping to avoid the problem of negative transfer, as well as determine a criterion for TL in RL. Third, this approach should be applied to other RL method, such as our preliminary investigation described elsewhere [2].

References

1. H. B. Ammar and M. E. Taylor. Common subspace transfer for reinforcement learning tasks. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS-11)*, May 2011.
2. H. B. Ammar, K. Tuyls, M. E. Taylor, K. Driessens, and G. Weiss. Reinforcement learning transfer via sparse coding. In *Proceedings of the Eleventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, June 2012.
3. L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Boca Raton, Florida, 2010.
4. S. jean Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine Learning Research*, 2007, 2007.
5. G. Konidaris. A framework for transfer in reinforcement learning. In *In Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.
6. G. Kuhlmann and P. Stone. Graph-based domain mapping for transfer learning in general games. In *Proceedings of The Eighteenth European Conference on Machine Learning*, September 2007.
7. M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149, December 2003.
8. H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *In NIPS*, pages 801–808. NIPS, 2007.
9. Y. Liu and P. Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 415–20, July 2006.
10. J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, Aug. 1999.
11. C. E. Rasmussen. In *Gaussian processes for machine learning*. MIT Press, 2006.
12. E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances In Neural Information Processing Systems*, pages 1257–1264. MIT press, 2006.
13. R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction, 1998.
14. E. Talvitie and S. Singh. An experts algorithm for transfer learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.
15. M. E. Taylor, G. Kuhlmann, and P. Stone. Autonomous transfer for reinforcement learning. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 283–290, May 2008.
16. M. E. Taylor and P. Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML)*, June 2007.
17. M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, December 2009.
18. M. E. Taylor, S. Whiteson, and P. Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 156–163, May 2007.
19. L. Torrey, T. Walker, J. Shavlik, and R. Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *In Proceedings of the Sixteenth European Conference on Machine Learning*, pages 412–424, 2005.