In *AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development*, Arlington, Virginia, November 2007.

Representation Transfer for Reinforcement Learning

Matthew E. Taylor and Peter Stone
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188
{mtaylor, pstone}@cs.utexas.edu

Abstract

Transfer learning problems are typically framed as leveraging knowledge learned on a source task to improve learning on a related, but different, target task. Current transfer learning methods are able to successfully transfer knowledge from a source reinforcement learning task into a target task, reducing learning time. However, the complimentary task of transferring knowledge between agents with different internal representations has not been well explored The goal in both types of transfer problems is the same: reduce the time needed to learn the target with transfer, relative to learning the target without transfer. This work defines *representation transfer*, contrasts it with task transfer, and introduces two novel algorithms. Additionally, we show representation transfer algorithms can also be successfully used for task transfer, providing an empirical connection between the two problems. These algorithms are fully implemented in a complex multiagent domain and experiments demonstrate that transferring the learned knowledge between different representations is both possible and beneficial.

Introduction

Transfer learning is typically framed as leveraging knowledge learned on a *source task* to improve learning on a related, but different, *target task*. Past research has demonstrated the possibility of achieving successful transfer between *reinforcement learning* (RL) (Sutton & Barto 1998) tasks. In this work we refer to such transfer learning problems as *task transfer*.

A key component of any reinforcement learning algorithm is the underlying *representation* used by the agent for learning (e.g. its function approximator or learning algorithm), and transfer learning approaches generally assume that the agent will use a similar (or even the same) representation to learn the target task as it used to learn the source. However, this assumption may not be necessary or desirable. This paper considers an orthogonal question: is it possible, and desirable, for agents to use different representations in the target and source? This paper defines and provides algorithms for this new problem of *representation transfer* (RT) and contrasts it with the more typical task transfer.

The motivation for transferring knowledge between tasks is clear: it may enable quicker and/or better learning on the target task after having learned on the source. Our two motivations for representation transfer are similar, though perhaps a bit more subtle.

One motivation for equipping an agent with the flexibility to learn with different representations is procedural. Suppose an agent has already been training on a source task with a certain learning method and function approximator (FA) but the performance is poor. A different representation could allow the agent to achieve higher performance. If experience is expensive (e.g. wear on the robot, data collection time, or cost of poor decisions) it is preferable to leverage the agent's existing knowledge to improve learning with the new representation and minimize sample complexity.

A second motivating factor is learning speed: changing representations partway through learning may allow agents to achieve better performance in less time. SOAR (Laird, Newell, & Rosenbloom 1987) can use multiple descriptions of planning problems and search problems, generated by a human user, for just this reason. We will show in this paper that it is advantageous to change internal representation while learning in some RL tasks, relative to using a fixed representation, so that higher performance is achieved more quickly.

Additionally, this study is inspired in part by human psychological experiments. Agents' representations are typically fixed when prototyped, but studies show (Simon 1975) that humans may change their representation of a problem as they gain more experience in a particular domain. While our system does not allow for automatic generation of a learned representation, this work addresses the necessary first step of being able to transfer knowledge between two representations.

This paper's main contributions are to introduce representation transfer, to provide two algorithms for RT, and to empirically demonstrate the efficacy of these algorithms in a complex multiagent RL domain. In order to test RT, we train on the same tasks with different learning algorithms, functions approximators, and parameterizations of these function approximators, and then demonstrate that transferring the learned knowledge among the representations is both possible and beneficial. We introduce two representation transfer algorithms and implement them in the RL benchmark domain of robot soccer Keepaway (Stone *et al.* 2006). Lastly, we show that the algorithms can be used for successful task transfer, underscoring the relatedness of representation and task transfer.

RT Algorithms

In this work, we consider transfer in reinforcement learning domains. Following standard notation (Sutton & Barto 1998), we say that an agent exists in an environment and at any given time is in some state $s \in S$, beginning at $s_{initial}$. An agent's knowledge of the current state of its environment, $s \in S$ is a vector of k *state variables*, so that $s = x_1, x_2, \ldots, x_k$. The agent selects an action from available actions, $a \in A$. The agent then moves to a new state based on the transition function $T : S \times A \mapsto S$ and is given a real-valued reward for reaching the new state $R : S \mapsto \mathbb{R}$. Over time the agent learns a policy, $\pi : S \mapsto A$, to maximize the expected total reward. Common ways of learning the policy are temporal difference (TD) (Sutton & Barto 1998) methods and direct policy search.

In this section we present two algorithms for addressing RT problems, where the source and target representations differ. We define an agent's *representation* as the learning method used, the FA used, and the FA's parameterization. As an example, suppose an agent in the source uses Q-Learning with a neural network FA that has 20 hidden nodes. The first algorithm, Complexification, is used to:

1. Transfer between different parameterizations (e.g. change to 30 hidden nodes)

The second, Offline RT, may be used for:

2. Transfer between different FAs (e.g. change to a radial basis function FA)
3. Transfer between different learning methods (e.g. change to policy search)
4. Transfer between tasks with different actions and state variables

We refer to scenarios 1 and 2 as *intra-policy-class transfer* because the policy representation remains constant. Scenario 3 is a type of *inter-policy-class transfer*, and Scenario 4 is task transfer.

Complexification

Complexification is a type of representation transfer where the function approximator is changed over time to allow for more representational power. Consider, for instance, the decision of whether to represent state variables conjunctively or independently. A linear interpolation of different state variables may be faster to learn, but a conjunctive representation has more descriptive power. Using Complexification, the agent can learn with a simple representation initially and then switch to a more complex representation later. Thus the agent can reap the benefits of fast initial training without suffering decreased asymptotic performance.

Algorithm 1 describes the process for transferring between value function representations with different parameterizations of state variables, e.g. FAs with different dimensionalities. The weights (parameters) of a learned FA are used as needed when the agent learns a target value function representation. If the target representation must calculate $Q(s, a)$ using a weight which is set to the default value rather than a learned one, the agent uses the source representation to set the weight. Using this process, a single weight from the source representation can be used to set multiple weights in the target representation.

Algorithm 1 Complexification

1: Train with a source representation and save the learned FA_{source}
2: **while** target agent trains on a task with FA_{target} **do**
3: **if** $Q(s, a)$ needs to use at least one uninitialized weight in FA_{target} **then**
4: Find the set of weights W that would be used to calculate $Q(s, a)$ with FA_{source}
5: Set any remaining uninitialized weight(s) in FA_{target} needed to calculate $Q(s, a)$ to the average of W

Note that this algorithm makes the most sense when used for FAs that exhibit *locality*: step 5 would execute once and initialize all weights when using a fully connected neural network. Thus we employ Algorithm 1 when using a FA which has many weights but only a subset are used to calculate each $Q(s, a)$ (e.g. a CMAC, as discussed later in this paper).

We will utilize this algorithm on a task which requires a conjunctive representation for optimal performance. This provides an existence proof that Complexification can be effective at reducing both the target representation training time and the total training time.

Offline RT

The key insight for *Offline RT* (ORT) is that an agent using a source representation can record some information about its experience using the learned policy. The agent may record s, the perceived state; a, the action taken; r, the immediate reward; and/or $Q(s, a)$, the long-term expected return. Then the agent can learn to mimic this behavior in the target representation without the use of on-line training (i.e. without more interactions with the environment). The agent is then able able to learn better performance faster than if it had learned the target representation without transfer. We consider three distinct scenarios where ORT algorithms could be utilized:

1. Intra-policy-class RT (Algorithm 2a): The representation differs by function approximator.
2. Inter-policy-class RT (Algorithms 2b & 2c): The representation changes from a value function learner to a policy search learner, or vice versa.
3. Task transfer (Algorithm 2d): The representation remains constant but the tasks differ.

Note that this is not an exhaustive list; it contains only the variants which we have implemented. (For instance, intra-policy-class RT for policy learners is similar to Algorithm 2a, and task transfer combined with inter-policy-class transfer is likewise a straightforward extension of the ORT method.) The ORT algorithms presented are necessarily dependant on the details of the representation used. Thus they may be appropriately thought of as meta-algorithms and we will show in later sections how they may be instantiated for specific learning methods and specific FAs.

Algorithm 2a describes intra-policy-class transfer for value function methods with different FAs. The agent saves n (state, action, Q-value) tuples and then trains offline with the target representation to predict those saved Q-values, given the corresponding state. Here offline training still utilizes a TD update, but the target Q-values are set by the recorded experience.

Algorithm 2a ORT: Value Functions

1: Train with a source representation
2: Record n $(s, a, q(s_i, a_i))$ tuples while the agent acts
3: **for** all n tuples **do**
4:     Train offline with target representation, learning to predict $Q_{target}(s_i, a_i) = q(s_i, a_i)$ for all $a \in A$
5: Train on-line using the target representation

When considering inter-policy-class transfer between a value function and a policy search method, the primary challenge to overcome is that the learned FAs represent different concepts: a value function by definition contains more information because it represents not only the best action, but also its expected value. However, the method described above for intra-policy-class transfer also generalizes to inter-policy-class transfer.

Inter-policy-class transfer between a value function and a policy search learner (Algorithm 2b) first records n (s, a) tuples and then trains a direct policy search learner offline so that π_{target} can behave similarly to the source learner. Here offline training simply means using the base learning algorithm to learn a policy that will take the same action from a given state as was taken in the saved experience.

Algorithm 2b ORT: Value Functions to Policies

1: Train with a source representation
2: Record n (s, a) tuples while the agent acts
3: **for** all n tuples **do**
4:     Train offline with target representation, learning $\pi_{target}(s_i) = a_i$
5: Train on-line using the target representation

Inter-policy-class transfer from a policy to a value function (Algorithm 2c) works by recording n (s, a, r) and then training a TD learner offline by (in effect) replaying the learned agent's experience, similar to Algorithm 2a. Step 4 uses the history to calculate q_i. In the undiscounted episodic case, the optimal predicted return from time t_0, q_i, is $\sum_{t_0 < t \leq t_{EpisodeEnd}} r_t$, and can thus be found by summing recorded rewards until the end of the episode is reached. Similarly, the discounted non-episodic case would sum rewards, multiplied by a discount factor. Steps 6 & 7 are used to generate some initial Q-values for actions not taken. If an action was not taken, we know that its Q-value was lower, but cannot know its exact value since the source policy learner does not estimate Q-values.

Lastly, we present an algorithm for inter-task transfer using a value function approximator (Algorithm 2d). Specifically, we assume that we have a pair of tasks that have different action and state variable spaces, but are related by two inter-task mappings. One mapping defines the relationship

Algorithm 2c ORT: Policies to Value Functions

1: Train with a source representation
2: Record n (s, a, r) tuples while the agent acts
3: **for** all n tuples **do**
4:     Use history to calculate the return, q_i, from s_i
5:     Train offline with target representation, learning to predict $Q_{target}(s_i, a_i) = q_i$
6:     **for** all $a_j \in A$ s.t. $a_j \neq a_i$ **do**
7:         Train to predict $Q_{target}(a_j) < q_i$
8: Train on-line using the target representation

between state variables: $\rho_X(x_{i,target}) = x_{j,source}$, and a second defines the relationship between actions in the two tasks: $\rho_A(a_{i,target}) = a_{j,source}$. This assumption is the same as used in past task transfer work (Maclin *et al.* 2005; Soni & Singh 2006; Taylor, Stone, & Liu 2005) for transfer between tasks with different state and action spaces.

Algorithm 2d ORT: Task Transfer for Value Functions

1: Train on a source task
2: Record n $(s, a, q(s_i, a_i))$ tuples while the agent acts
3: **for** all n tuples **do**
4:     Construct s' in the target task so that every state variable, s'_j, is set by the corresponding source task state variable k in s_i: $\rho_X(j) = k$
5:     Train offline in target task, learning to predict $Q_{target}(s', a') = q(s', \rho_A(a'_i))$ for all $a' \in A_{target}$
6: Train on-line using the target task

Keepaway

To test the efficacy of RT we consider the RoboCup simulated soccer Keepaway domain, a multiagent domain with. We use setups similar to past research (Stone, Sutton, & Kuhlmann 2005) and agents based on version 0.6 of the benchmark players distributed by UT-Austin (Stone *et al.* 2006). This section details the Keepaway domain and the RL methods used in our experiments.

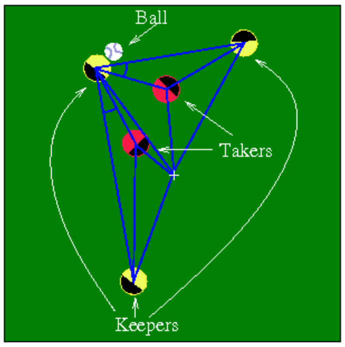

Figure 1: 3 Keepers play against 2 Takers. A Keeper's state is composed of 11 distances to players and the center of the field as well as 2 angles along passing lanes.

This multiagent domain has noisy sensors and actuators, and enforces a hidden state so that agents can perceive only a partial world view at any time. In *Keepaway*, one team of *keepers* attempt to possess a ball on a field, while another team of *takers* attempt to steal the ball or force it out of bounds. Keepers that make better decisions about their actions are able to maintain possession of the ball longer. In 3 vs. 2 Keepaway, three keepers are initially placed in three corners of a 20m $\times$ 20m field with a ball near one of the keepers. Two takers are located in the fourth corner. The keeper's world state is defined by 13 variables, as shown in Figure 1. The keepers receive a +1 reward for every time step the ball remains in play.

Keepers chose from the high level macro actions: Hold Ball, Get Open, Receive, and Pass. A keeper in 3 vs. 2 Keepaway with the ball may either hold the ball or pass it to a teammate: A = $\{$*hold*, *passToTeammate1*, *passToTeammate2*$\}$. Otherwise, keepers execute Receive to chase loose balls and get open for passes. Takers follow a hand-coded policy. Full details for Keepaway can be found elsewhere (Stone, Sutton, & Kuhlmann 2005).

XOR Keepaway

This section describes a modification to the 3 vs. 2 Keepaway task so that the agent's representation must be capable of learning an "exclusive or" to achieve top performance. This is one instance of a task where a linear representation can learn quickly but is eventually outperformed by a more complex representation and is thus a prime candidate for Complexification[1].

In XOR Keepaway, the 3 vs. 2 Keepaway task is modified to change the effect of agents' actions. *Good pass*, executes the pass action and additionally disables the takers for 2 seconds. *Bad pass* causes the keeper's pass to travel directly to the closest taker. These effects are triggered based on the agent's chosen pass action and 4 state variables: the distance to the closest taker, $d(K_1, T_1)$, the distance from the closest teammate to a taker, $d(K_2, T)$, the passing angle to the closest teammate, $ang(K_2)$, and the distance to the closest teammate, $d(K_1, K_2)$. Thus agents which lack the representational power to express an XOR can learn but are unable to achieve optimal performance. This modification to the task changes the effects of the agents' decisions but leaves the rest of the task unchanged. Details appear in Figure 2.

if Keeper attempts pass to closest teammate **then**
  if $(4m < d(K_1, T_1) < 6m)$ XOR $(9m < d(K_2, T) < 12m)$ **then**
    Execute *good pass*
  else
    Execute *bad pass*
else if Keeper attempts pass to furthest teammate **then**
  if $(9m < d(K_1, K_2) < 12m)$ OR $(45^0 < ang(K_2) < 90^0)$ **then**
    Execute *good pass*
  else
    Execute *bad pass*
else
  if Keeper would have executed *good pass* if it had decided to pass **then**
    Execute *bad pass*
  else
    Execute *hold ball*

Figure 2: XOR Keepaway changes the effects of agent's actions but leave the rest of the task unchanged from 3 vs. 2.

[1] In informal experiments, Complexification did not improve 3 vs. 2 Keepaway performance, likely because it can be learned well when all state variables are considered independently (Stone, Sutton, & Kuhlmann 2005).

4 vs. 3 Keepaway

4 vs. 3 Keepaway has been used in the past for task transfer (Soni & Singh 2006; Taylor, Stone, & Liu 2005) because adding more players changes the state representation and available actions. In 4 vs. 3 Keepaway, A = $\{$hold, passToTeammate1, passToTeammate2, passToTeammate3$\}$, and the state is composed of 19 state variables due to added players.

4 vs. 3 is more difficult for keepers to learn due to the extra players and asymptotic performance is lowered. The addition of an extra taker and keeper to the 3 vs. 2 task also results in a qualitative change because of the taker behavior. In 3 vs. 2, both takers must charge the ball, but in 4 vs. 3 one taker is free to roam the field and attempts to intercept passes.

Learning with Sarsa

To learn Keepaway, we use setups similar to past research in this domain (Stone, Sutton, & Kuhlmann 2005), and in particular use Sarsa (Rummery & Niranjan 1994; Singh & Sutton 1996), a well understood TD method. Because Keepaway is continuous, some sort of function approximation is necessary. Cerebellar model articulation controllers (CMACs), Radial basis functions (RBFs), and neural networks have been used successfully for TD function approximation in the Keepaway domain (Stone *et al.* 2006).

CMACs (Albus 1981) use multiple linear tilings to approximate a continuous value function. There can either be a separate CMAC for each state feature so that each is independent, or the CMACs can tile multiple state features together conjunctively, as will be done in XOR Keepaway. RBFs generalize tile coding so that instead of linearly summing tile weights, each "tile" is represented by a continuous bias function which weights the tile's contribution by the distance of a state from the center of the tile. The neural network FA allows a learner to calculate a value from a set of continuous, real-valued state variables. Each input to the neural network is set to the value of a state variable and each output corresponds to an action. Activations of the output nodes correspond to Q values. We define our representation as a feedforward network with a single hidden layer of 20 units, again consistent with past research in this domain. CMAC and RBF FAs have their weights initially set to zero and neural networks are initialized so that weights are near zero. In all cases, a Sarsa update is used to change weights over time to approximate an action-value function.

Learning with NEAT

Policy search methods have had significant empirical success learning policies to solve RL tasks. NeuroEvolution of Augmenting Topologies (NEAT) (Stanley & Miikkulainen 2002) is one such method that evolves populations of neural networks. NEAT is an appropriate choice for this paper due to past empirical successes on difficult RL tasks such as double pole balancing (Stanley & Miikkulainen 2002) and 3 vs. 2 Keepaway (Taylor, Whiteson, & Stone 2006).

NEAT evolves network topology by combining the search for network weights with evolution of network structure.

NEAT starts with a population of networks without any hidden nodes: inputs are connected directly to outputs. Two mutation operators introduce new structure incrementally by adding hidden nodes or adding links to a network. Structural mutations that improve performance tend to survive evolution, and NEAT generally searches through lower weight dimensions before exploring more complex topologies. NEAT is a general purpose optimization technique, but when applied to RL problems, it typically evolves action selectors. Inputs describe the agent's current state and there is one output for each action. The agent executes whichever action has the highest activation.

Results

In this section we present empirical results showing that Complexification and the four variations of ORT can successfully transfer knowledge. Specifically, we test:

1. Complexification in XOR Keepaway with a Sarsa learner utilizing a CMAC FA
2. ORT for Value Functions in 3 vs. 2 Keepaway between RBF and neural network Sarsa learners
3. ORT for Value Functions in 3 vs. 2 Keepaway between neural network and RBF Sarsa learners
4. ORT for Policies to Value Functions in 3 vs. 2 Keepaway between NEAT and Sarsa learners
5. ORT for Value Functions to Policies in 3 vs. 2 Keepaway between Sarsa and NEAT learners
6. ORT for Task Transfer with Value Functions between 3 vs. 2 and 4 vs. 3 Keepaway

We consider two related goals for both representation and task transfer problems. This section shows that all of the methods presented can reduce the training time in the target. Additionally, experiments 1, 5, and 6 show that the total training time may also be reduced, a significantly more difficult goal. For RT, that means that an agent can improve performance on a single task by switching internal representations partway through learning, rather than using a single representation for an equivalent amount of time.

Learning curves presented in the section each average ten independent trials. The x-axis shows the number of Soccer Server simulator hours, where wall clock time is roughly half of the simulator time. The y-axis shows the average performance of the keepers by showing the average episode length in simulator seconds. Error bars show one standard deviation. (Note that we only show error bars on alternating curves for readability.) All parameters chosen in this section were selected via experimentation with a small set of initial test experiments.

Complexification in XOR Keepaway

To master the XOR Keepaway task we use Sarsa to learn with CMAC FAs, with both independently and conjunctively tiled parameterizations. The independently tiled players use 13 separate CMACs, one for each state feature. The conjunctively tiled players use 10 separate CMACs, 9 of which independently tile state features. The last CMAC is a conjunctive tiling of the remaining 4 state features: $d(K_1, T_1)$, $d(K_2, T)$, $ang(K_2)$, and $d(K_1, K_2)$. We train the independently tiled players for 20 hours and then save the weights in their CMAC FAs. To get a small performance improvement, we set all zero weights to the average weight value, a method previously shown to improve CMAC performance (Taylor, Stone, & Liu 2005). We then train conjunctively tiled CMAC players, using the previously learned weights as needed as per Algorithm 1.

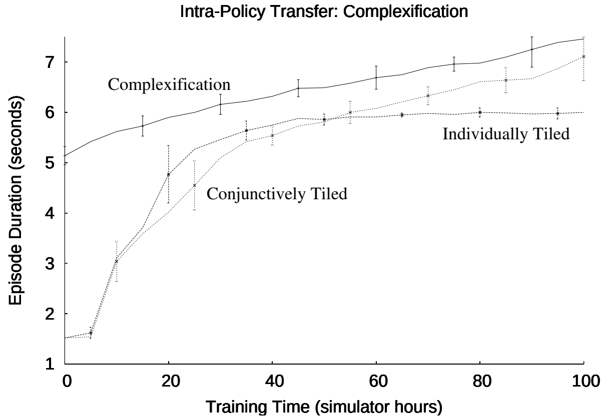

Figure 3: Learning with Complexification outperforms learning with individually tiled CMACs without transfer and partially conjunctive CMACs without transfer.

Agents learn best when the four relevant state features are conjunctively tiled: Figure 3 shows that players learning with conjunctive FAs outperform the players using independently tiled FAs. However, initially, agents using independently tiled state features are able to learn faster. Agents trained with independent CMACs for 20 hours can then transfer to a conjunctive representation via Algorithm 1, significantly outperforming players that only use the independent representation. A Student's t-test confirms that this performance increase is statistically different from learning without transfer with independently tiled CMACs after 40 of total training time.

Additionally, the total training time required is decreased by Complexification relative to learning only the conjunctive tiling. Even when source agent training time is also taken into account, Complexification significantly outperforms learning without transfer with the conjunctive representation until 55 hours of training time has elapsed. Examined differently, learning without transfer with a conjunctive tiling takes 55 hours to reach a 6.0 second hold time, while an agent using both source and target representations take a total of only 45 hours, an 18% reduction in learning time.

Thus, in the XOR Keepaway task, using Complexification to transfer knowledge between two different representations outperforms using either representation alone for the equivalent amount of time.

Offline RT in 3 vs. 2 Keepaway

When learning 3 vs. 2 Keepaway, ORT algorithms may be used to transfer knowledge between different FAs and between different policy representations. The next section presents two experiments to show that if a source representation has been learned and a target representation utilizes a different FA, ORT can successfully reduce the target's train-

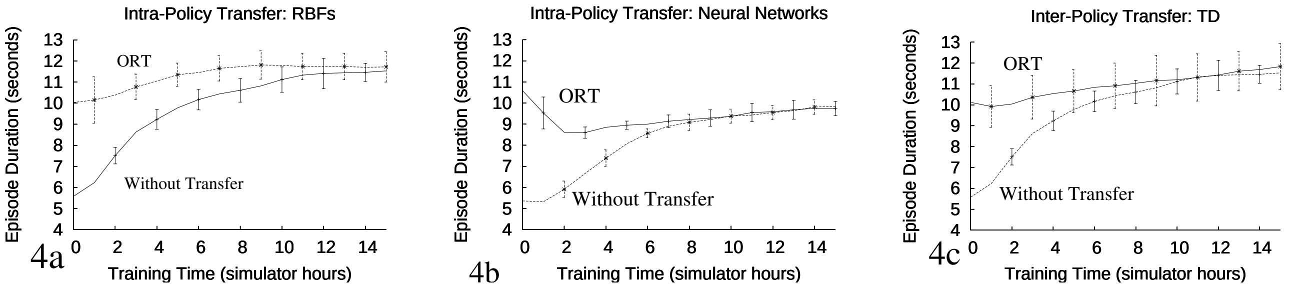

Figure 4: a) RBF players utilize ORT from neural networks to outperform RBF players without transfer. b) Neural network players utilize ORT from RBF players to outperform neural network players without transfer. c) RBF players using ORT from NEAT players outperform RBF players without transfer.

ing time. The subsequent section shows two experiments where the source and target differ both by FA and by type of learning method. In one experiment the target's training time is reduced while the second experiment shows transfer benefit in both learning scenarios: both the target's training time and the total training time are reduced.

Intra-policy ORT To demonstrate intra-policy transfer, we first train Sarsa players using a neural network on 3 vs. 2 Keepaway for 20 simulator hours and then record 20,000 tuples, which took roughly 1.0 simulator hour. TD-RBF players are then trained offline by iterating over all tuples 5 times and updating $Q(s_i, a)$, where $a \in A$, via Sarsa with a learning rate of 0.001 (set after trying 4 different common parameter values). Thus the agent is able to learn in the new representation by replaying data gathered when training with the old representation. Using Algorithm 2a, this process takes roughly 8 minutes of wall clock time. Figure 4a shows that RT from neural network players outperforms RBF players learning without transfer. Differences graphed are statistically significant for times less than 11 simulator hours.

The reverse experiment trains RBF players for 20 simulator hours and then saves 20,000 tuples. We train the neural network players offline by iterating over all tuples five times. We found that updating $Q(s, a_i)$ for $a_i \neq a$ was not as efficient as updating only the Q-value for the action selected in a state. This is likely because of the non-locality effect of neural networks where changing a single weight may affect all output values. Figure 4b shows how RT helps improve the performance of the neural network players. The differences are statistically significant for times less than 8 simulator hours and the offline RT training took less than 1 minute of wall clock time.

Inter-policy ORT To demonstrate value function to direct policy search transfer, we first train NEAT keepers for 500 simulator hours in the 3 vs. 2 Keepaway task and then use RT to initialize RBF players via offline Sarsa training. We found that the value-function learners needed to learn a more complex representation and thus used 50,000 tuples (which takes roughly 2.6 simulator hours to record). If $Q(s_i, a') > Q(s_i, a_i)$, where a' was an action not chosen by the source agent, we set a target value[2] of $Q(s_i, a') = 0.9 \times Q(s_i, a_i)$. The offline training, as described previously, takes roughly 4 minutes of wall clock time.

Figure 4c shows that the RBF players using RT from learned NEAT representations initially have a much higher performance. Training causes an initial drop in performance as the Q-values, and therefore the current policy, are changed to more accurately describe the task. However, performance of the players using RT is statistically better than those learning without transfer until 7 simulator hours of training has occurred. After 7 simulator hours, the performance difference between using RT and learning without transfer is not significant. This shows that if one has trained policies, it is advantageous to use them to initialize TD agents, particularly if the training time is short or if the on-line reward is critical.

The reverse experiment trains 3 vs. 2 Keepaway using the value function RBF players for 20 simulator hours. After learning, one of the keepers saves 1,000 tuples, and we use inter-policy RT to initialize a population of 100 policies offline for 100 generations[3]. After the target keepers have finished learning, we evaluate the champion from each generation for 1,000 episodes to more accurately graph the learned policy performances. Figure 5 shows that NEAT players utilizing RT outperform NEAT players learning without transfer. This result is particularly dramatic because TD-RBF players initially train much faster than NEAT players. The 20 hours of simulator time spent training the RBF players and the roughly 0.1 simulator hours to collect the 1,000 tuples are not reflected in this graph.

The difference between learning with and without transfer is statistically significant for all points graphed (except for 490 simulator hours) and the total training time needed to reach a pre-determined performance threshold in the tar-

[2]Recall that the only information we have regarding the value of non-chosen actions are that they should be lower valued than than selected actions. However, setting those values too low may disrupt the FA so that it does not generalize well to unseen states. 0.9 was chosen after informally testing three different parameter values.

[3]NEAT trains offline with a fitness function that sums the number of times the action predicted by NEAT from a given state matches that action that had been recorded.

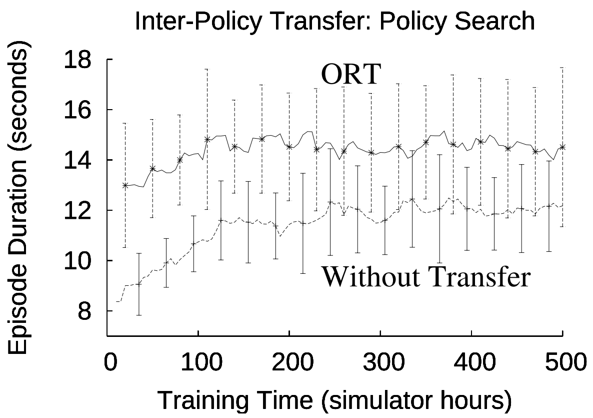

Figure 5: ORT can initialize NEAT players from RBF players to significantly outperforms learning without transfer.

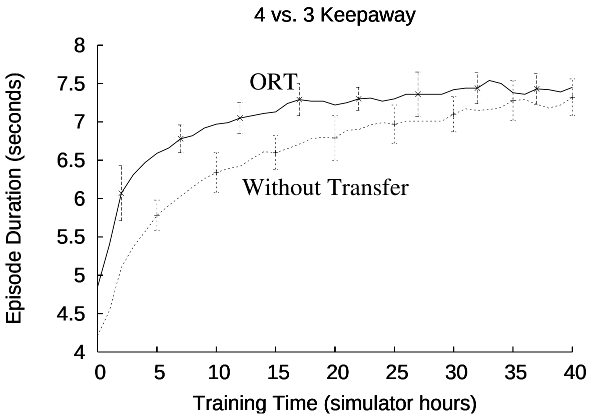

Figure 6: ORT successfully reduces training time for task transfer between 3 vs. 2 and 4 vs. 3 Keepaway.

get task has been reduced. For instance, if the goal is to train a set of agents to hold the ball in 3 vs. 2 Keepaway for 14.0 seconds via NEAT, it takes approximately 700 simulator hours to learn without transfer (not shown). The total simulator time needed to reach the same threshold using ORT is less than 100 simulator hours. Additionally, the best learned average performance of 15.0 seconds is better than the best performance achieved by NEAT learning without transfer in 1000 simulator hours (Taylor, Whiteson, & Stone 2006).

This paper focuses on sample complexity, assuming that agents operating in a physical world are most affected by slow sample gathering. If computational complexity were taken into account, RT would still show significant improvement. Although we did not optimize for it, the wall clock time for RT's offline training was only 4.3 hours per trial. Therefore, RT would still successfully improve performance if our goal had been to minimize wall clock time.

Offline RT for Task Transfer

ORT is able to meet both transfer scenario goals when the source and target are 3 vs. 2 and 4 vs. 3 Keepaway, successfully performing task transfer. This result suggests both that ORT is a general algorithm that may be applied to both RT and task transfer and that other RT algorithms may work for both types of transfer.

To transfer between 3 vs. 2 and 4 vs. 3, we use ρ_X and ρ_A used previously in this pair of tasks (Taylor, Stone, & Liu 2005). 3 vs. 2 players learning with Sarsa and RBF FAs are trained for 5 simulator hours. The final 20,000 tuples are saved at the end of training (taking roughly 2 simulator hours). 4 vs. 3 players, also using Sarsa and RBF FAs, are initialized by training offline using Algorithm 2d, where the inter-task mappings are used to transform the experience from 3 vs. 2 so that the states and actions are applicable in 4 vs. 3. The batch training over all tuples is repeated 5 times.

Figure 6 shows that ORT reduces the target task training time, meeting the goal of transfer in the first scenario. The performance of the learners using ORT is better than that of learning without transfer until a time of 31 simulator hours. Furthermore, the total time is reduced when accounting for the 5 hours of training in 3 vs. 2. In this case, the ORT agents statistically outperform agents training without transfer during hours 10 – 25. Put another way, it will take agents learning without transfer an average of 26 simulator hours to reach a hold time of 7.0 seconds, but agents using ORT will use a total time of only 17 simulator hours to reach the same performance level.

Related Work

Using multiple representations to solve a problem is not a new idea. For instance, SOAR (Laird, Newell, & Rosenbloom 1987) uses multiple descriptions of planning problems to help with search and learning. Kaplan's production system (1989) was able to simulate the representation shift that humans often undergo when solving the *mutilated checkerboard* (McCarthy 1964) problem. Other work (Fink 1999) used libraries of problem solving and "problem description improvement" algorithms to automatically change representations in planning problems. *Implicit imitation* (Price & Boutilier 2003) allows an RL agent to train while watching a mentor with similar actions, but this method does not directly address internal representation differences. Additionally, all training is done on-line; agents using imitation do not initially perform better than learning without transfer.

None of these methods directly address the problem of transferring knowledge between different representations in an RL setting. By using RT methods like Complexification and ORT, different representations can be leveraged so that better performance can be more quickly learned, possibly in conjunction with existing RL speedup methods.

Our work shows the application of ORT to task transfer between 3 vs. 2 and 4 vs. 3. When the Complexification algorithm is used for task transfer between 3 vs. 2 and 4 vs. 3, it can make use of ρ_X and ρ_A analogously. However, our previous value-function transfer algorithm (Taylor, Stone, & Liu 2005) is very similar and has been shown to reduce total training time as well as target task training time. The main difference is that we perform the weight transfer, via Complexification, on-line while the agent interacts with the target task, while they transferred *after* learning the source but *before* learning the target task. Other recent work (Ahmadi, Taylor, & Stone 2007) uses an algorithm similar to Complexification, but concentrates on adding state variables over time, rather than shifting between different FA parameterizations.

Work by Maclin et. al. (2005) and Soni and Singh (2006) address similar transfer learning problems with different methods. Note that the change in state variables is necessitated by differences in the source and target tasks, but such an internal change could also be considered a type of representation transfer.

Future Work

This paper presents algorithms for transfer between different internal representations. We have presented five different scenarios in which RT improves agent performance relative to learning without transfer. Two of these scenarios show that RT can significantly reduce the total training time as well. In addition to representation transfer, we show that RT algorithms can be directly used to reduce both target and total training times for task transfer, a related but distinct problem. We have tested our algorithms in three versions of robot soccer Keepaway, using Sarsa and NEAT as representative learning algorithms and CMAC, RBFs, and neural networks as representative function approximators.

In the future we would like to test RT in more domains and with more representations. The experiments presented in this paper were chosen to be representative of the power of RT but are not exhaustive. For example, we would like to show that ORT can be used to transfer between policy search learners. We would also like to test ORT when the source and targets differ *both* in representation and task. We believe this will be possible as both Complexification and ORT may effectively transfer between tasks as well as representations.

This paper has introduced three situations where transfer reduces the total training time, but it would be useful to be able to *a priori* know if a given task could be learned faster by using multiple representations. We have also left open the questions of how different amounts of saved experience effect the efficacy of RT and if the initial dip in performance (e.g. Figure 4c) is caused by overfitting. Lastly, we intend to further explore the relationship between task and RT by developing, and analyzing, more methods which are able to perform both kinds of transfer.

Conclusion

This paper presents algorithms for RT to transfer knowledge between internal representations. We have presented five different scenarios in which RT improves agent performance relative to learning from scratch. Two of these scenarios show that RT can significantly reduce the total training time as well. In addition to representation transfer, we show that RT algorithms can be directly used to reduce both target and total training times for task transfer, a related but distinct problem. We have tested our algorithms in three versions of robot soccer Keepaway, using Sarsa and NEAT as representative learning algorithms and CMAC, RBFs, and neural networks as representative function approximators.

Acknowledgments

We would like to thank Cynthia Matuszek, Shimon Whiteson, Andrew Dreher, Bryan Klimt, and Nate Kohl for helpful comments and suggestions. This research was supported in part by DARPA grant HR0011-04-1-0035, NSF CAREER award IIS-0237699, and NSF award EIA-0303609.

References

Ahmadi, M.; Taylor, M. E.; and Stone, P. 2007. IFSA: Incremental feature-set augmentation for reinforcement learning tasks. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*.

Albus, J. S. 1981. *Brains, Behavior, and Robotics*. Peterborough, NH: Byte Books.

Fink, E. 1999. Automatic representation changes in problem solving. Technical Report CMU-CS-99-150, Depart. of Computer Science, Carnegie Mellon University.

Kaplan, C. A. 1989. Switch: A simulation of representational change in the mutilated checkboard problem. Technical Report C.I.P. 477, Department of Psychology, Carnegie Mellon University.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. SOAR: An architecture for general intelligence. *Artificial Intelligence* 33(1):1–64.

Maclin, R.; Shavlik, J.; Torrey, L.; Walker, T.; and Wild, E. 2005. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proceedings of the 20th National Conference on Artificial Intelligence*.

McCarthy, J. 1964. A tough nut for proof procedures. Technical Report Sail AI Memo 16, Computer Science Department, Stanford University.

Price, B., and Boutilier, C. 2003. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research* 19:569–629.

Rummery, G., and Niranjan, M. 1994. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG-RT 116, Engineering Department, Cambridge University.

Simon, H. A. 1975. The functional equivalence of problem solving skills. *Cognitive Psychology* 7:268–288.

Singh, S. P., and Sutton, R. S. 1996. Reinforcement learning with replaceing eligibility traces. *Machine Learning* 22:123–158.

Soni, V., and Singh, S. 2006. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of the Twenty First National Conference on Artificial Intelligence*.

Stanley, K. O., and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation* 10(2):99–127.

Stone, P.; Kuhlmann, G.; Taylor, M. E.; and Liu, Y. 2006. Keepaway soccer: From machine learning testbed to benchmark. In Noda, I.; Jacoff, A.; Bredenfeld, A.; and Takahashi, Y., eds., *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020. Berlin: Springer Verlag. 93–105.

Stone, P.; Sutton, R. S.; and Kuhlmann, G. 2005. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior* 13(3):165–188.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. MIT Press.

Taylor, M. E.; Stone, P.; and Liu, Y. 2005. Value functions for RL-based behavior transfer: A comparative study. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*.

Taylor, M. E.; Whiteson, S.; and Stone, P. 2006. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1321–28.