Journal of Machine Learning Research 8 (2007) 2125-2167 Submitted 11/06; Revised 4/07; Published 9/07

Transfer Learning via Inter-Task Mappings for Temporal Difference Learning

Matthew E. Taylor MTAYLOR@CS.UTEXAS.EDU
Peter Stone PSTONE@CS.UTEXAS.EDU
Yaxin Liu YXLIU@CS.UTEXAS.EDU
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

Editor: Michael L. Littman

Abstract

Temporal difference (TD) learning (Sutton and Barto, 1998) has become a popular reinforcement learning technique in recent years. TD methods, relying on function approximators to generalize learning to novel situations, have had some experimental successes and have been shown to exhibit some desirable properties in theory, but the most basic algorithms have often been found slow in practice. This empirical result has motivated the development of many methods that speed up reinforcement learning by modifying a task for the learner or helping the learner better generalize to novel situations. This article focuses on generalizing *across tasks*, thereby speeding up learning, via a novel form of transfer using handcoded task relationships. We compare learning on a complex task with three function approximators, a cerebellar model arithmetic computer (CMAC), an artificial neural network (ANN), and a radial basis function (RBF), and empirically demonstrate that directly transferring the *action-value function* can lead to a dramatic speedup in learning with all three. Using *transfer via inter-task mapping* (TVITM), agents are able to learn one task and then markedly reduce the time it takes to learn a more complex task. Our algorithms are fully implemented and tested in the RoboCup soccer Keepaway domain.

This article contains and extends material published in two conference papers (Taylor and Stone, 2005; Taylor et al., 2005).

Keywords: transfer learning, reinforcement learning, temporal difference methods, value function approximation, inter-task mapping

1. Introduction

Machine learning has traditionally been limited to training and testing on the same distribution of problem instances. However, humans are able to learn to perform well in complex tasks by utilizing principles learned in previous tasks. Few current machine learning methods are able to transfer knowledge between pairs of tasks, and none are able to transfer between a broad range of tasks to the extent that humans are. This article presents a new method for *transfer learning* in the *reinforcement learning* (RL) framework using *temporal difference* (TD) learning methods (Sutton and Barto, 1998), whereby an agent can learn faster in a *target task* after training on a different, typically less complex, *source task*.

TD learning methods have shown some success in many reinforcement learning tasks because of their ability to learn where there is limited prior knowledge and minimal environmental feedback.

©2007 Matthew E. Taylor, Peter Stone and Yaxin Liu.

However, the basic unenhanced TD algorithms, such as Q-Learning (Watkins, 1989) and Sarsa (Rummery and Niranjan, 1994; Singh and Sutton, 1996), have been found slow to produce near-optimal behaviors in practice. Many techniques exist (Selfridge et al., 1985; Colombetti and Dorigo, 1993; Asada et al., 1994) which attempt, with more or less success, to speed up the learning process. Section 9 will discuss in depth how our transfer learning method differs from other existing methods and can potentially be combined with them if desired.

In this article we introduce *transfer via inter-task mapping* (TVITM), whereby a TD learner trained on one task with *action-value function* RL can learn faster when training on another task with related, but different, state and action spaces. TVITM thus enables faster TD learning in situations where there are two or more similar tasks. This transfer formulation is analogous to a human being told how a novel task is related to a known task, and then using this relation to decide how to perform the novel task. The key technical challenge is mapping an action-value function—the expected return or value of taking a particular action in a particular state—in one representation to a meaningful action-value function in another, typically larger, representation. It is this transfer functional which defines transfer in the TVITM framework.

In stochastic domains with continuous state spaces, agents will rarely (if ever) visit the same state twice. It is therefore necessary for learning agents to use *function approximation* when estimating the action-value function. Without some form of approximation, an agent would only be able to predict a value for states that it had previously visited. In this work we are primarily concerned with a different kind of generalization. Instead of finding similarities between different states, we focus on exploiting similarities between different *tasks*.

The primary contribution of this article is an existence proof that there are domains in which it is possible to construct a mapping between tasks and thereby speed up learning by transferring an action-value function. This approach may seem counterintuitive initially: the action-value function is the learned information which is directly tied to the particular task it was learned in. Nevertheless, we will demonstrate the efficacy of using TVITM to speed up learning in agents across tasks, irrespective of the representation used by the function approximator. Three different function approximators (as defined in Section 4.3), a CMAC, an ANN, and an RBF, are used to learn a single reinforcement learning problem. We will compare their effectiveness and demonstrate why TVITM is promising for future transfer studies.

The remainder of this article is organized as follows. Section 2 formally defines TVITM. Section 3 gives an overview of the tasks over which we quantitatively test our transfer method. Section 4 gives details of learning in our primary domain, robot soccer Keepaway. Section 5 describes how we perform transfer in our selected tasks. Sections 6 and 7 present the results of our experiments. Section 8 discusses some of their implications and future work. Section 9 details other related work while contrasting our methods and Section 10 concludes.

2. Transfer via Inter-Task Mapping

TVITM is defined for value function reinforcement learners. Thus, to formally define how to use our transfer method we first briefly review the general reinforcement learning framework that conforms to the generally accepted notation for *Markov decision processes* (MDP) (Puterman, 1994).

In an MDP, there is some set of possible perceptions of the current state of the world, S, and a learner has an initial starting state, $s_{initial}$. An agent's knowledge of the current state of its environment from observation, $s \in S$ is a vector of k *state variables*, so that $s = \langle x_1, x_2, \dots, x_k \rangle$. There is a

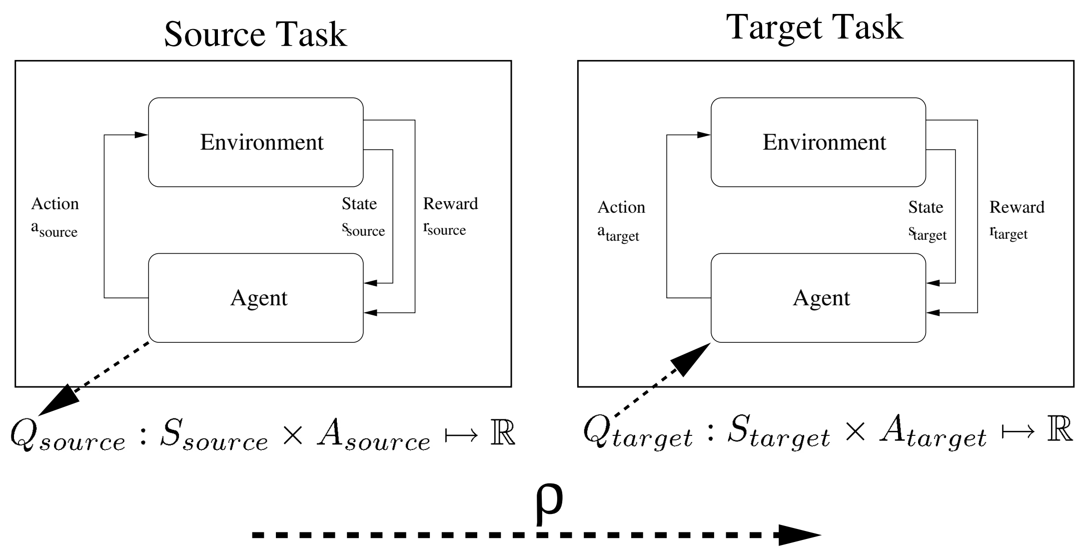


Figure 1: ρ is a functional that transforms a state-action function Q from one task so that it is applicable in a second task with different state and action spaces.

set of actions, A, which the agent can perform. The reward function, $R : S \mapsto \mathbb{R}$, maps each state of the environment to a single number which is the instantaneous reward achieved for reaching the state. The transition function, $T : S \times A \mapsto S$, takes a state and an action and returns the state of the environment after the action is performed. If transitions are non-deterministic the transition function is a probability distribution function. A learner is able to sense the current state, s, and typically knows A and what state variables comprise S. However, it does not know R, how it is rewarded for moving between states, or T, how actions move the agent between states.

A learner chooses which action to take in a given perceived environmental state by using a policy, $\pi : S \mapsto A$. π is modified by the learner over time to improve performance, the expected total reward accumulated, and it completely defines the behavior of the learner in an environment. In the general case the policy can be stochastic. The success of an agent is determined by how well it maximizes the total reward it receives in the long run while acting under some policy π. An *optimal policy*, π^*, is a policy that maximizes the expectation of this value. Any reasonable learning algorithm attempts to modify π over time so that the agent's performance approaches that of π^* in the limit. Value function reinforcement learning relies on learning a value function $V : S \mapsto \mathbb{R}$ so that the learner is able to estimate the total discounted reward that would be accumulated from moving to state s and then following the current policy π. In practice, the action-value function $Q : S \times A \mapsto \mathbb{R}$ is often learned, which frees the learner from having to explicitly model the transition function. If the action-value function is optimal (i.e., $Q = Q^*$), π^* can be followed by always selecting the optimal action a, which is the action with the largest value of $Q(s,a)$ in the current state.

In this article we consider the general case where the state features in the source and target tasks are different ($S_{source} \neq S_{target}$), and/or the actions in the source and target tasks are different ($A_{source} \neq A_{target}$). To use the learned action-value function from the source task $Q_{(source,final)}$ as the initial action-value function for a TD learner in a target task, we must transform the action-value function so that it can be directly applied to the new state and action space. This transformed action-value function may not provide immediate improvement over acting randomly in the target task, but it should bias the learner so that it is able to learn the target task faster than if it were learning without transfer.

A transfer functional $\rho(Q)$ will allow us to apply a policy in a new task (see Figure 1). The policy transform functional ρ needs to modify the action-value function so that it accepts S_{target} as inputs and allows for A_{target} to be outputs. A policy generally selects the action which is believed to accumulate the largest expected total reward; the problem of transforming a policy between two tasks therefore reduces to transforming the action-value function. Defining ρ to do this correctly is the key technical challenge to enable general TVITM.

2.1 Constructing a Transfer Functional

Given an arbitrary pair of unknown tasks and no experience in the pair of tasks, one could not hope to correctly define ρ, the transfer functional (for example, there are certainly pairs of tasks which have no relationship and thus mastery in one task would not lead to improved performance in the other). For our transfer method to succeed, not only must the two tasks be related, but we should be able to characterize *how* they are related. We represent these relations as a pair of inter-task mappings, denoted χ_X and χ_A. State variables in the target task are mapped via χ_X to the most similar state variable in the source task:

$$\chi_X(x_{i,target}) = x_{j,source}.$$

Similarly, χ_A maps each action in the target task to the most similar action in the source tasks:

$$\chi_A(a_{i,target}) = a_{j,source}.$$

χ_X and χ_A, mappings from the target task to the source task, are used to construct ρ, a transfer functional from the source task to the target task (see Figure 2). Note that χ_X and χ_A are defined only once for a pair of tasks, while multiple ρs (one for each type of function approximator employed by our learning agents), are constructed from this single pair of inter-task mappings. In this article we take χ_X and χ_A as given; learning them autonomously is an important goal of future work.

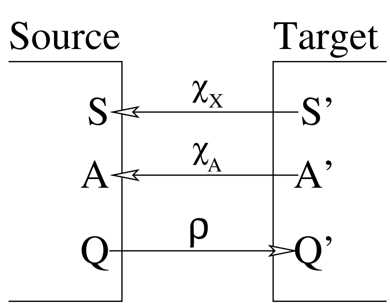


Figure 2: χ_X and χ_A are mappings from a target to a source task; ρ maps an action-value function from a source to a target task.

Thus, given χ_X, χ_A, and a learned action-value function Q_{source}, we can create an initial action-value function Q_{target}. The details of ρ depend on the particular function approximators used in the source and target task. In Sections 5.3 and 5.4 we construct three different ρ functionals from χ_X and χ_A for the RoboCup Soccer Keepaway domain.

It may seem counterintuitive that low-level action-value function information is able to speed up learning across different tasks. Often transfer techniques attempt to abstract knowledge so that it is applicable to more general tasks. For instance, an agent could be trained to balance a pole on a cart and then be asked to balance a pair of poles on a cart. An example of abstract knowledge in this domain would be things like “avoid hitting the end of the track,” “it is better to have the pole near vertical,” etc. Instead of trying to transfer higher level information about a source task into a target task, we instead focus on information contained in individual weights within function approximators. In this example, such weights which would contain specific information such as how fast to move the cart to the left when a pole was at a particular angle. Weights that encode this type of low level knowledge are the most task-specific part of the learner’s knowledge, but it is exactly these domain-dependant details that allow us to achieve significant speedups on similar tasks.

2.2 Evaluation of Transfer

There are many possible ways to measure the effectiveness of transfer, including:

1. Asymptotic Performance: Measure the performance after convergence in the target task.

2. Initial Performance: Measure the initial performance in the target task.

3. Total Reward: Measure the total accumulated reward during training in the target task.

4. Area Ratio: Measure the area between the transfer and non-transfer learning curves.

5. Time-to-Threshold: Measure the time needed to reach a performance threshold in the target task.

This section discusses these five different testing criteria and argues that the time-to-threshold metric is most appropriate for evaluating TVITM in our experimental domain.

One could examine the asymptotic performance of a learned policy. Such a metric would compare the average reward achieved after learning both with and without transfer. Leveraging source task knowledge may allow a learner to reach a higher asymptote, but it may be difficult to tell when the learner has converged, and convergence may take prohibitively long. Additionally, in applications of reinforcement learning we are often interested in the time required, not simply the performance of a learner with infinite time. Lastly, it is not uncommon for different learners to converge to the same asymptotic performance on a given task, making them indistinguishable in terms of the asymptotic performance metric.

A second measure of transfer is to look at the initial performance in a target task. Learned source task knowledge may be able to improve initial target task performance relative to learning the target task without transfer. While such an initial performance boost is appealing, we argue in Section 6 that this goal may often be infeasible to achieve in practice. Further, because we are primarily interested in the learning process of agents in pairs of tasks, it makes sense to concentrate on the rate of learning in the target task.

A third possible measure is that of the total reward accumulated during training. By measuring the total reward over some amount of training, we are able to quantify how much reward the agent accumulates in a certain amount of time. Transfer may allow an agent to accumulate more reward in the target task when compared to the non-transfer case; better initial performance and faster learning would help agents achieve more on-line reward. TD methods are not guaranteed to converge with function approximation and even when they do, learners do not always converge to the same performance levels. If the time considered is long enough, a learning method which achieves very fast learning will "lose" to a learning method which learns very slowly but eventually plateaus at a slightly higher performance level. Thus this metric is most appropriate for tasks that have a defined time limit for learning. However, it is more common to think of learning until some performance is reached (if ever), rather than specifying the amount of time, computational complexity, or sample complexity *a priori*.

A fourth measure of transfer efficacy is that of the ratio of the areas defined by two learning curves. Consider two learning curves: one that uses transfer, and one that does not. Assuming that the transfer learner is able to learn faster or reach a higher performance, the area under the transfer curve will be greater than the area under the non-transfer curve. The ratio

$$r = \frac{\textit{area under curve with transfer - area under curve without transfer}}{\textit{area under curve without transfer}}$$

gives us a metric for how much transfer improves learning. This metric is most appropriate if the same eventual performance is achieved, or there is a predetermined time for the task. Otherwise the ratio will directly depend on the length of time considered for the two curves. In the tasks we consider, the learners that use transfer and the learners that learn without transfer do not always plateau to the same performance, nor is there a defined task length.

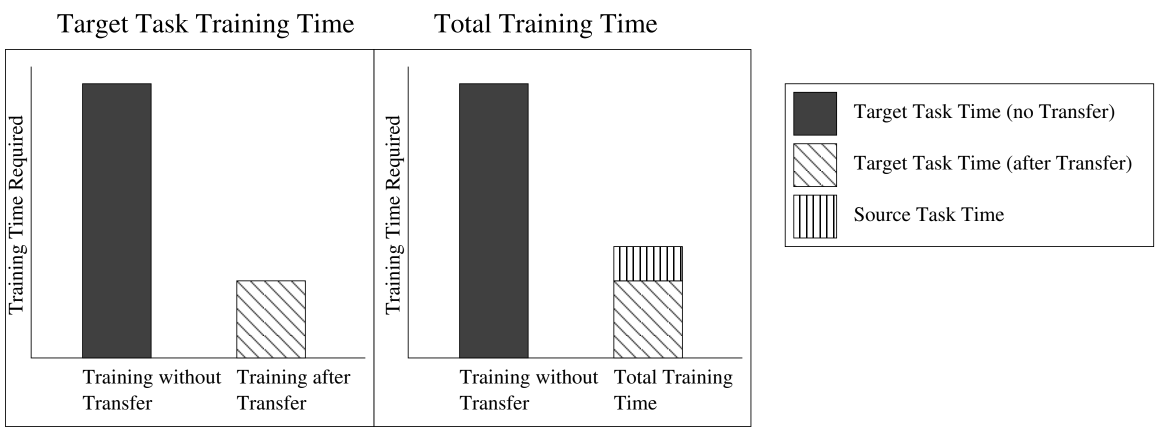


Figure 3: In this article we evaluate transfer by both considering the training time in the target task (left) and by considering the total time spent training in both tasks (right).

For these reasons we use the time-to-threshold metric. After preliminary experiments are conducted, thresholds for analysis are chosen such that all trials must learn for some amount of time before reaching the performance threshold, and most trials are able to eventually reach the threshold. We will show in Section 6 that given a $Q_{(source,final)}$, the training time for the learner in the target task to reach some performance threshold decreases when initializing $Q_{(target,initial)}$ with $\rho(Q_{(source,final)})$. This criterion is relevant when the source task is given and is of interest in its own right or if $Q_{(source,final)}$ can be used repeatedly to speed up multiple related tasks (see Figure 3).

A stronger measure of success that we will also use is that the training time for *both* tasks using TVITM is shorter than the training time to learn just the target task without transfer. This criterion is relevant when the source task is created for the sole purpose of speeding up learning with transfer and $Q_{(source,final)}$ is not reused.

3. Testbed Domains

This section introduces the Keepaway task, the testbed domain where we empirically evaluate our transfer method, and use as a running example throughout the rest of the article. We also introduce the Knight Joust, a task which we will later use as a supplemental source task from which to transfer into Keepaway.

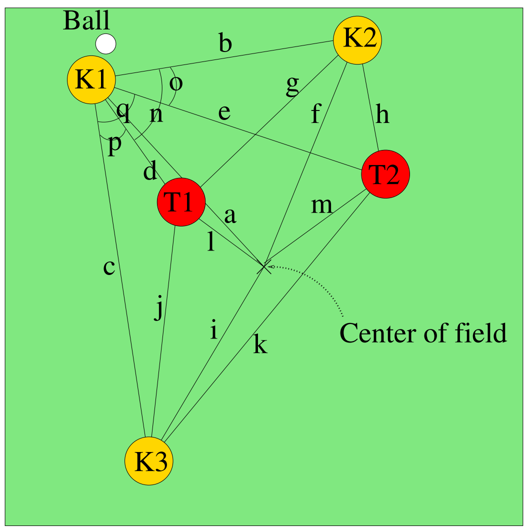
3.1 The Keepaway Task

RoboCup simulated soccer is well understood, as it has been the basis of multiple international competitions and research challenges. The multiagent domain incorporates noisy sensors and actuators, as well as enforcing a hidden state so that agents only have a partial world view at any given time. While previous work has attempted to use machine learning to learn the full simulated soccer problem (Andre and Teller, 1999; Riedmiller et al., 2001), the complexity and size of the problem have so far proven intractable. However, many of the RoboCup subproblems have been isolated and solved using machine learning techniques, including the task of playing Keepaway. By focusing on the smaller task of Keepaway we are able to use reinforcement learning to learn an action-value function for a more complex task, establish that TVITM provides considerable benefit, and hold the required computational resources to manageable levels.

Since late 2002, the *Keepaway* task has been part of the official release of the open source RoboCup Soccer Server used at RoboCup (starting with version 9.1.0). Agents in the simulator (Noda et al., 1998) receive visual perceptions every 150 *msec* indicating the relative distance and angle to visible objects in the world, such as the ball and other agents. They may execute a primitive, parameterized action such as `turn(`*angle*`)`, `dash(`*power*`)`, or `kick(`*power*`,`*angle*`)` every 100 *msec*. Thus the agents must sense and act asynchronously. Random noise is injected into all sensations and actions. Individual agents must be controlled by separate processes, with no inter-agent communication permitted other than via the simulator itself, which enforces communication bandwidth and range constraints. Full details of the simulator are presented in the server manual (Chen et al., 2003).

When started in a special mode, the simulator enforces the rules of the Keepaway task, as described below, instead of the rules of full soccer. In particular, the simulator places the players at their initial positions at the start of each *episode* and ends an episode when the ball leaves the play region or is taken away. In this mode, the simulator also informs the players when an episode has ended and produces a log file with the duration of each episode.

Keepaway is a subproblem of RoboCup simulated soccer in which one team—the *keepers*—attempts to maintain possession of the ball within a limited region while another team—the *takers*—attempts to steal the ball or force it out of bounds, ending an episode. Whenever the takers take possession or the ball leaves the region, the episode ends and the players are reset for another episode (with the keepers being given possession of the ball again). Standard parameters of the task include the size of the region, the number of keepers, and the number of takers. Other parameters such as player speed, player kick speed, player vision capabilities, sensor noise, and actuator noise, are all adjustable. This paper will use standard settings with the exception of a set of experiments in Section 7.1 that uses different kick speed actuators. Figure 4 shows a diagram of 3 keepers and 2 takers (3 vs. 2).[1]



a. dist(K1,C)
b. dist(K1,K2)
c. dist(K1,K3)
d. dist(K1,T1)
e. dist(K1,T2)
f. dist(K2,C)
g. dist(K2.T1)
h. dist(K2,T2)
i. dist(K3,C)
j. dist(K3,T1)
k. dist(K3,T2)
l. dist(T1,C)
m. dist(T2,C)
n. ang(K2,K1,T1)
o. ang(K2,K1,T2)
p. ang(K3,K1,T1)
q. ang(K3,K1,T2)

Figure 4: This diagram depicts the distances and angles used to construct the 13 state variables used for learning with 3 keepers and 2 takers. Relevant objects are the 5 players and the center of the field, C. All 13 state variables are enumerated later in Table 1.

When Keepaway was introduced as a testbed (Stone and Sutton, 2002), a standard task was defined. All our experiments are run on a code base derived from version 0.6 of the benchmark Keepaway implementation[2] (Stone et al., 2006) and the RoboCup Soccer Server version 9.4.5.

Our setup is similar to past research in Keepaway (Stone et al., 2005), which showed that Sarsa with CMAC function approximation can learn well in this domain. On a $25m \times 25m$ field, three keepers are initially placed near three corners of the field and a ball is placed near one of the keepers. The two takers are placed in the fourth corner. When the episode starts, the three keepers attempt to keep control of the ball by passing among themselves and moving to open positions. The keeper with the ball has the option to either pass the ball to one of its two teammates or to hold the ball. In this task $A = \{$*hold, pass to closest teammate, pass to second closest teammate*$\}$. S is defined by 13 state variables, as shown in Figure 4. When a taker gains control of the ball or the ball is kicked out of the field's bounds the episode is finished. The reward to the learning algorithm is the number of time steps the ball remains in play after an action is taken. After an episode ends, the next starts with a random keeper placed near the ball.

3.2 Knight Joust

Knight Joust is a variation on a previously introduced task (Taylor and Stone, 2007) situated in the grid world domain. In this task the player begins on one end of a $25m \times 25m$ board, the opponent begins on the other, and the players alternate moves. The player's goal is to reach the opposite

1. Flash files illustrating the task are available at `http://www.cs.utexas.edu/~AustinVilla/sim/Keepaway/`.
2. Released players are available at `http://www.cs.utexas.edu/~AustinVilla/sim/Keepaway/`.

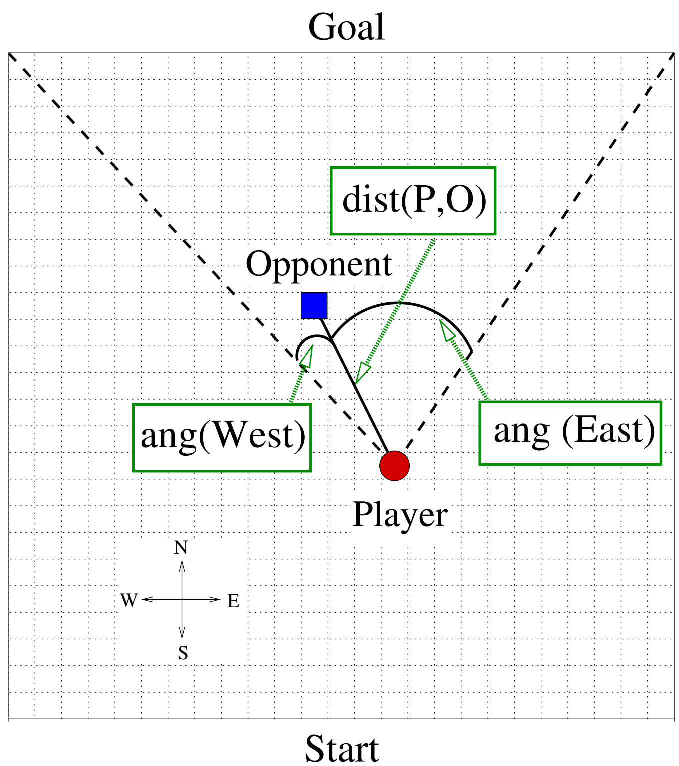


Figure 5: Knight Joust: The player attempts to reach the goal end of a a 25×25 grid-world while the opponent attempts to touch the player.

end of the board without being touched by the opponent (see Figure 5); the episode ends if the player reaches the goal line or the opponent is on the same square as the player. The state space is discretized into $1m$ squares and there is no noise in the perception. The player's state variables are composed of the distance from the player to the opponent, and two angles which describe how much of the goal line is viewable by the player.

The player may deterministically move one square North or perform a knight's jump where the player moves one square North and two West or two East: $A = \{Forward, Jump_W, Jump_E\}$. The opponent follows a fixed stochastic policy which allows it to move in any of the 8 directions. Given the start state and size of the board, an opponent that acted optimally would always prevent the player from reaching the goal line. In order to allow the player to reach the goal line with a learned policy, we restrict the opponent's motion so that 10% of the time, when it attempts to move East or West, it fails. 20% of the time, when it attempts to move North, it fails (the opponent never fails when it attempts to move South). These movement failure probabilities were selected after initial experiments showed that this opponent policy generally prevented the player from reaching the goal before training but allowed the player to reach the goal line with a high probability after learning with Sarsa. The opponent's policy is as follows:

if opponent is E of player **then**
  Move W with probability 0.9
else if opponent is W of player **then**
  Move E with probability 0.9
if opponent is N of player **then**
  Move S with probability 1.0
else if opponent is S of player **then**
  Move N with probability 0.8

The player receives a reward of +20 every time it takes the forward action, 0 if either knight jump action is taken, and an additional +20 upon reaching the goal line. The player uses Sarsa with a Q-value table to learn in this task. While this task is quite different from Keepaway, there are some similarities, such as favoring larger distances between player and opponent. This domain is much simpler than Keepaway and an agent takes roughly 20 seconds of wall-clock time (roughly 50,000 episodes) to plateau in our Java-based simulation.

4. Learning Keepaway

TVITM aims to improve learning in the target task based on prior learning in the source, and therefore a prerequisite is that both source and target tasks are learnable. In this section we outline how tasks in the Keepaway domain are learned using Sarsa.

4.1 Sarsa

Sarsa is a TD method that learns to estimate the action-value function by backing up the received rewards through time. Sarsa is an acronym for State Action Reward State Action, describing the 5-tuple needed to perform the update: $(s_t, a_t, r, s_{t+1}, a_{t+1})$, where s_t, a_t are the the agent's current state and action, r is the immediate reward the agent receives from the environment, and s_{t+1}, a_{t+1} are the agent's subsequent state and chosen action. After each action, action values are updated according to the following rule:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r + Q(s_{t+1}, a_{t+1})) \tag{1}$$

where α is the learning rate. Note that if the task is non-episodic we need to include an extra discount factor to weigh immediate rewards more heavily than future rewards.

Like other TD methods, Sarsa estimates the value of a given state-action pair by bootstrapping off the estimates of other such pairs. In particular, the value of a given state-action pair (s_t, a_t) can be estimated as $r + Q(s_{t+1}, a_{t+1})$, which is the value of the subsequent state-action pair (s_{t+1}, a_{t+1}) plus the immediate reward received during the transition. Sarsa's update rule takes the old action-value estimate $Q(s_t, a_t)$, and moves it incrementally closer towards this new estimate. The learning rate parameter α controls the size of these increments. Ideally, these action-value estimates will become more accurate over time and the agent's policy will steadily improve.

4.2 Framing the RL Problem

As described by Stone et al. (2005), the Keepaway problem maps fairly directly onto the discrete-time, episodic, reinforcement-learning framework. As a way of incorporating domain knowledge, the learners choose not from the simulator's primitive actions but from a set of higher-level macro-actions implemented as part of the player. These macro-actions can last more than one time step and the keepers have opportunities to make decisions only when an on-going macro-action terminates. To handle such situations, it is convenient to treat the problem as a *semi-Markov decision process*, or SMDP (Puterman, 1994; Bradtke and Duff, 1995). The agents make decisions at discrete SMDP time steps (when macro-actions are initiated and terminated).

The keepers learn in a constrained policy space: they have the freedom to decide which action to take only when in possession of the ball. A keeper in possession may either hold the ball or pass to one of its teammates. Therefore the number of actions from which the keeper with the ball may choose is equal to the number of keepers in the task. Keepers not in possession of the ball are required to execute the Receive macro-action in which the player who can reach the ball the fastest goes to the ball and the remaining players follow a handcoded strategy to try to get open for a pass.

When training the keepers, the behavior of the takers is "hard-wired" and relatively simple. The two takers that are closest to the ball go directly toward it. Note that a single keeper can hold the ball indefinitely from a single taker by constantly keeping its body between the ball and the taker. The remaining takers, if present, try to block open passing lanes.

The keepers learn which action to take when in possession of the ball by using episodic SMDP Sarsa(λ) (Sutton and Barto, 1998), to learn their task.[3] The episode consists of a sequence of states, macro-actions, and rewards. We choose episode duration as the performance measure for this task: the keepers attempt to maximize it while the the takers try to minimize it. Since we want the keepers to maintain possession of the ball for as long as possible, the reward in the Keepaway task is simply the number of time steps the ball remains in play after a macro-action is initiated. Learning attempts to discover an optimal action-value function that maps state-action pairs to expected time steps until the episode will end.

As more players are added to the task, Keepaway becomes harder for the keepers because the field becomes more crowded. As more takers are added there are more players to block passing lanes and chase down any errant passes. As more keepers are added, the keeper with the ball has more passing options but the average pass distance is shorter. This reduced distance forces more passes and often leads to more errors because of the noisy actuators and sensors. For this reason, keepers in 4 vs. 3 (i.e., 4 keepers and 3 takers) take longer to learn an optimal control policy than in 3 vs. 2. The average episode length of the best policy for a constant field size also decreases when adding an equal number of keepers and takers. The time needed to learn a policy with performance roughly equal to a handcoded solution roughly doubles as each additional keeper and taker is added (Stone et al., 2005). In our experiments we set the agents to have a 360° field of view. Although agents do also learn with a more realistic 90° field of view, allowing the agents to see 360° speeds up the rate of learning, enabling more experiments. Additionally, 360° vision also increases the learned hold times in comparison to learning with the limited 90° vision.

For the purposes of this article, it is particularly important to note the state variables and action possibilities used by the learners. The keepers' states consist of distances and angles of the keepers $K_1 - K_n$, the takers $T_1 - T_m$, and the center of the playing region C (see Figure 4 and Table 1). Keepers and takers are ordered by increasing distance from the ball, leading to an indexical representation. Note that as the number of keepers n and the number of takers m increase, the number of state variables also increases so that the more complex state can be fully described. S must

3. In previous experiments we found that setting $\lambda = 0$ produced the best learning results and "Sarsa(0)" is synonymous with "Sarsa."

3 vs. 2 State Variables

State Variable	Description
$dist(K_1, C)$	Distance from keeper with ball to center of field
$dist(K_1, K_2)$	Distance from keeper with ball to closest teammate
$dist(K_1, K_3)$	Distance from keeper with ball to second closest teammate
$dist(K_1, T_1)$	Distance from keeper with ball to closest taker
$dist(K_1, T_2)$	Distance from keeper with ball to second closest taker
$dist(K_2, C)$	Distance from closest teammate to center of field
$dist(K_3, C)$	Distance from second closest teammate to center of field
$dist(T_1, C)$	Distance from closest taker to center of field
$dist(T_2, C)$	Distance from second closest taker to center of field
Min($dist(K_2, T_1)$, $dist(K_2, T_2)$)	Distance from nearest teammate to its nearest taker
Min($dist(K_3, T_1)$, $dist(K_3, T_2)$)	Distance from second nearest teammate to its nearest taker
Min($ang(K_2, K_1, T_1)$, $ang(K_2, K_1, T_2)$)	Angle of passing lane from keeper with ball to closest teammate
Min($ang(K_3, K_1, T_1)$, $ang(K_3, K_1, T_2)$)	Angle of passing lane from keeper with ball to second closest teammate

Table 1: This table lists all state variables used for representing the state of 3 vs. 2 Keepaway. Note that the state is ego-centric for the keeper with the ball and rotationally invariant.

change (e.g., there are more distances to players to account for) and $|A|$ increases as there are more teammates for the keeper with possession of the ball to pass to.

4.3 Function Approximation

Continuous state variables combined with noise necessitate some form of function approximation for the action-value function: an agent will rarely visit the same state twice, with the possible exception of an initial start state. In this article we use three distinct function approximators and show that all are able to learn Keepaway, as well as use our transfer methodology (see Figure 6). In one implementation, we use linear tile-coding function approximation, also known as a CMAC (cerebellar model arithmetic computer), which has been successfully used in many reinforcement learning systems (Albus, 1981), including past Keepaway research (Stone et al., 2005). A second uses radial basis function approximation (RBF) (Sutton and Barto, 1998). The third implementation uses artificial neural networks (ANN), another method for function approximation that has had some notable past successes (Tesauro, 1994; Crites and Barto, 1996).

A CMAC takes arbitrary groups of continuous state variables and lays infinite, axis-parallel tilings over them. Using this method we are able to discretize the continuous state space by using tilings while maintaining the capability to generalize via multiple overlapping tilings. Increasing the tile widths allows better generalization while increasing the number of tilings allows more accurate representations of smaller details. The number of tiles and width of the tilings are handcoded: this sets the center, c_i, of each tile and dictates which state values will activate which tiles. The function approximation is learned by changing how much each tile contributes to the output of the function approximator. Thus, the output from the CMAC is the computed sum:

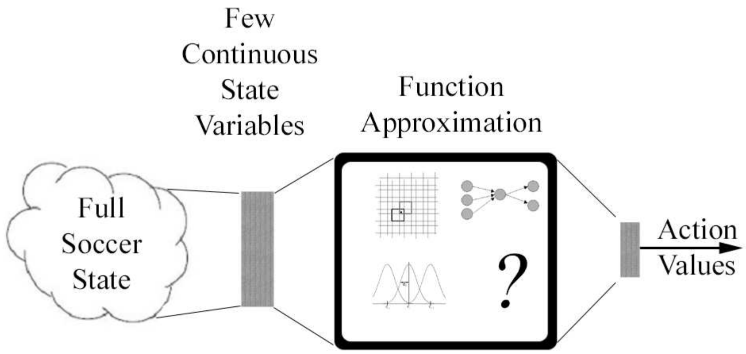


Figure 6: Function approximation is necessary for agents interacting with a continuous world. This article examines three different function approximators for Keepaway but many different methods could in principle be used by a transfer learner.

$$\hat{f}(x) = \sum_i w_i f_i(x) \tag{2}$$

but only tiles which are activated by the current state feature contribute to the sum:

$$f_i(x) = \begin{cases} 1, & \text{if tile i is activated} \\ 0, & \text{otherwise.} \end{cases}$$

By default, all the CMAC's weights are initialized to zero. This approach to function approximation in the RoboCup soccer domain has been detailed previously (Stone et al., 2005). We use one-dimensional tilings so that each state variable is tiled independently, but the principles apply in the n-dimensional case. For each variable, 32 tilings were overlaid, each offset from the others by by $\frac{1}{32}$ of a tile width. For each tiling, the current state activates a single tile. In 3 vs. 2, there are 32 tiles active for each state variable and $13 \times 32 = 416$ tiles activated in total. The tile widths are defined so that the distance state features have a width of roughly 3.0 meters and tiles for angle state features are roughly 10.0 degrees. In this work we do not vary these settings but set them to agree with past work.

RBF function approximation is a generalization of the tile coding idea to continuous functions (Sutton and Barto, 1998) and their application in Keepaway have been introduced elsewhere (Stone et al., 2006). When considering a single state variable, an RBF approximator is a linear function approximator:

$$\hat{f}(x) = \sum_i w_i f_i(x) \tag{3}$$

where the basis functions have the form:

$$f_i(x) = \phi(|x - c_i|) \tag{4}$$

x is the value of the current state variable, c_i is the center of feature i (which is unchanged from the CMAC, Equation 2), and w_i represents weights that can be modified over time by a learning algorithm. Here we set the features to be evenly spaced Gaussian radial basis functions, where:

$$\phi(x) = \exp(-\frac{x^2}{2\sigma^2}). \tag{5}$$

The σ parameter controls the width of the Gaussian function and therefore the amount of generalization over the state space. We set σ to 0.25, which roughly spans the width of three CMAC tiles, after running experiments with $\sigma = 1.0, 0.5, 0.25$ and observing that the learning rates were not dramatically effected.

As we did with the CMAC, we again assume that the state variables are independent and thus have one set of linearly tiled RBFs for each state variable. Similar to the CMAC implementation, all state variables are tiled independently and there are 32 tilings for each state variable. The RBFs in every tiling are spaced so that their centers correspond to the centers of CMAC tiles. We use Equations 3-5 to calculate Q-values of a state s. Because σ specifies that the spread of a RBF is roughly 3 CMAC tiles, each 3 vs. 2 state will thus be computed from approximately $3 \times 13 \times 32 = 1248$ weights in total. All weights w_i are initially set to zero, but over time learning updates changes the values of the weights so that the resulting Q-values more closely predict the true returns, as specified by Equation 1.

The ANN function approximator similarly allows a learner to approximate the action-value function, given a set of continuous, real valued, state variables. Each input to the ANN is set to the value of a state variable and the output corresponds to an action. Activations of the output nodes correspond to Q values. We use a fully-connected feedforward network with a single hidden layer of 20 sigmoid units for all our tasks. The output layer nodes are linear and return the currently predicted $Q(s,a)$ for each action. Weights were initialized with uniformly random numbers chosen from $[0, 1.0]$. We had also tried initializing the weights uniformly to 0 and from $[0, 0.01]$, with little effect on learning rates. This network topology was selected after testing 7 different sizes of hidden layers, from 5 to 30 hidden units. Again, the learning rate did not seem to be strongly affected by this parameter. The network is trained using standard backpropagation where the error signal to modify weights is generated by the Sarsa algorithm, as with the other function approximators.

4.4 Learning 3 vs. 2 Keepaway

To learn 3 vs. 2 Keepaway as a source task for transfer, all weights in the CMAC and RBF function approximators are initially set to zero; every initial state-action value is thus zero and our action-value function is uniform. All weights and biases in the 13-20-3 feedforward ANN are set to small random numbers to encourage faster backprop training (Mehrotra et al., 1997) but the initial action-value is still nearly uniform. As training progresses, the weights of the function approximators are changed by Sarsa so that the average hold time of the keepers increases.

In our experiments we set the learning rate, α, to be 0.1 for the CMAC function approximator, as in previous experiments. α was 0.05, and 0.125 for the RBF and ANN function approximators, respectively. These values were determined after trying approximately five different learning rates for each function approximator. The exploration rate, ϵ, was set to 0.01 (1%) in all experiments and λ was set to 0, which we selected to be consistent with past work (Stone et al., 2005).

4.5 Learning 4 vs. 3 Keepaway and 5 vs. 4 Keepaway without Transfer

Holding the field size constant we now add an additional keeper and an additional taker to generate the 4 vs. 3 task. All three takers still start in a single corner. Three keepers start in each of the other three corners and the fourth keeper begins an episode at the center of the field. R and T are effectively unchanged from 3 vs. 2 Keepaway, but now A = {*hold, pass to closest teammate, pass* *to second closest teammate, pass to third closest teammate*}, and S is made up of 19 state variables due to the added players.

It is also important to point out that the addition of an extra taker and keeper in 4 vs. 3 results in a qualitative change in the task. In 3 vs. 2 both takers must go towards the ball as two takers are needed to capture the ball from the keeper. However, the third taker is now free to roam the field and attempt to intercept passes. This necessarily changes the keeper behavior as one teammate is often blocked from receiving a pass by this new taker. Furthermore, adding a keeper in the center of the field changes the start state significantly as now the keeper that starts with the ball has a teammate that is closer to itself, but is also closer to the takers.

In order to quantify how fast an agent in 4 vs. 3 learns, we set a target performance of 10.0 seconds for ANN learners, while CMAC and RBF learners have a target of 11.5 seconds. These threshold times are chosen so that learners are able to consistently attain the performance level without transfer, but players using TVITM must also learn and do not initially perform above the threshold. CMAC and RBF learners are able to learn better policies than the ANN learners and thus have higher threshold values. When a group of four CMAC keepers has learned to hold the ball from the three takers for an average of 11.5 seconds over 1,000 episodes we say that the keepers have sufficiently learned the 4 vs. 3 task. Thus agents learn until the on-line reward of the keepers, averaged over 1,000 episodes, with exploration, passes a set threshold.[4] In 4 vs. 3, it takes a set of four keepers using CMAC function approximators 30.8 simulator hours (roughly 15 hours of wall-clock time, or 12,000 episodes) on average to learn to hold the ball for 11.5 seconds when training without transfer. By comparison, in 3 vs. 2, it takes a set of three keepers using CMAC function approximators 5.5 hours on average to learn to hold the ball for 11.5 seconds when training without transfer.

The ANN used in 4 vs. 3 is a 19-20-4 feedforward network.[5] The ANN learners do not learn as quickly nor achieve as high a performance before learning plateaus and therefore we use a threshold of 10.0 seconds. (After training four keepers using ANN function approximation without transfer in 4 vs. 3 for over 80 hours, the average hold time was only 10.3 seconds.)

5 vs. 4 is harder than 4 vs. 3 for the same reasons that 4 vs. 3 is more difficult than 3 vs. 2. In 5 vs. 4 three keepers are again placed in three corners and the two remaining keepers are placed in the middle of the $25m \times 25m$ field. All four takers are placed in the fourth corner. There are now five actions: {*hold, pass to closest teammate, pass to second closest teammate, pass to third closest teammate, pass to fourth closest teammate*}, and 25 state variables. In 5 vs. 4, it takes a set of five keepers using CMAC function approximators 59.9 hours (roughly 24,000 episodes) on average to learn to hold the ball for 11.5 seconds when training without transfer. In this paper we investigate the 5 vs. 4 problem only with the CMAC function approximator.

5. Transfer via Inter-Task Mapping in Keepaway

Having introduced our testbed domain and baseline learning approaches, we can now show how TVITM is performed in Keepaway, utilizing terminology described in Section 2. Recall that TVITM

4. We begin each trial by following the initial policy for 1,000 episodes without learning (and therefore without counting this time towards the learning time). This enables us to assign a well-defined initial performance when we begin learning because there already exist 1,000 episodes to average over.
5. Again, other networks with different numbers of hidden units were tried, but the differences in learning times were not significant.

relies on a functional ρ that is able to transfer an action-value function from a source task into a target task with different state and action spaces. ρ is built from the inter-task mappings χ_X and χ_A, and thus this section begins by defining these two mappings and then describing how they are used to generate different ρs.

In the Keepaway domain, A and S are determined by the current Keepaway task and thus differ from instance to instance. $s_{initial}$, R, and T, though formally different, are effectively constant across tasks. When S and A change, $s_{initial}$, R, and T change by definition because they are functions defined over S and A, but in practice R is always defined as $+1$ for every time step that the keepers maintain possession, and $s_{initial}$ and T are always defined by the RoboCup soccer simulation.

5.1 Defining χ_X and χ_A for 4 vs. 3 Keepaway and 3 vs. 2 Keepaway

In the Keepaway domain we are able to intuit the inter-task mappings between states and actions in the two tasks based on our knowledge of the domain. Our choice for the mappings is supported by empirical evidence in Section 6 showing that using these mappings do allow us to construct transfer functions that successfully reduce training time. In general, the transform may not be so straightforward, but experimenting in a domain where it is easily defined allows us to focus on showing the benefits of transfer. This article demonstrates that transfer can be successful when a mapping is available, while we leave it to future work to show how to best construct (or learn) such a transform.

We define χ_A, the inter-task mapping between actions in the two tasks, by identifying actions that have similar effects on the world state in both tasks. For the 3 vs. 2 and 4 vs. 3 tasks, the action "Hold ball" is equivalent because this action has a similar effect on the world in both tasks. Likewise, the action "Pass to closest keeper" is analogous in both tasks, as is "Pass to second closest keeper." We map the novel target action "Pass to third closest keeper" to "Pass to second closest keeper" in the source task.

The state variable mapping, χ_X, is handled with a similar strategy. Each of the 19 state variables in the 4 vs. 3 task is mapped to a similar state variable in the 3 vs. 2 task. For instance, "Distance to closest keeper" is the same in both tasks. "Distance to second closest keeper" in the target task is similar to "Distance to second closest keeper" in the source task. "Distance to third closest keeper" in the target task is also mapped to "Distance to second closest keeper" in the source task. See Table 2 for a full description of χ_X.

Now that χ_A and χ_X are defined, relating the state variables and actions in a target task to the state variables and actions in a source task, we can use them to construct ρs for different internal representations. The functionals will transfer the learned action-value function from the source task into the target task. We denote these functionals as ρ_{CMAC}, ρ_{RBF}, and ρ_{ANN} for the CMAC, RBF, and ANN function approximators, respectively.

5.2 Defining χ_X and χ_A for 4 vs. 3 Keepaway and Knight Joust

The Knight Joust task is less similar to 4 vs. 3 Keepaway than 3 vs. 2 Keepaway is. There are many fewer state variables, a less similar transition function, and a very different reward structure. However, we will show later that information from Knight Joust can significantly improve the performance of Keepaway players because very basic information, such as that it is desirable to maximize the distance to the opponent, will initially cause the players to perform better than acting randomly.

Description of χ_X Mapping from 4 vs. 3 to 3 vs. 2

4 vs. 3 state variable	3 vs. 2 state variable
$dist(K_1, C)$	$dist(K_1, C)$
$dist(K_1, K_2)$	$dist(K_1, K_2)$
$dist(K_1, K_3)$	$dist(K_1, K_3)$
$\mathbf{dist(K_1, K_4)}$	$dist(K_1, K_3)$
$dist(K_1, T_1)$	$dist(K_1, T_1)$
$dist(K_1, T_2)$	$dist(K_1, T_2)$
$\mathbf{dist(K_1, T_3)}$	$dist(K_1, T_2)$
$dist(K_2, C)$	$dist(K_2, C)$
$dist(K_3, C)$	$dist(K_3, C)$
$\mathbf{dist(K_4, C)}$	$dist(K_3, C)$
$dist(T_1, C)$	$dist(T_1, C)$
$dist(T_2, C)$	$dist(T_2, C)$
$\mathbf{dist(T_3, C)}$	$dist(T_2, C)$
Min($dist(K_2, T_1)$, $dist(K_2, T_2)$, $\mathbf{dist(K_2, T_3)}$)	Min($dist(K_2, T_1)$, $dist(K_2, T_2)$)
Min($dist(K_3, T_1)$, $dist(K_3, T_2)$, $\mathbf{dist(K_3, T_3)}$)	Min($dist(K_3, T_1)$, $dist(K_3, T_2)$)
Min($\mathbf{dist(K_4, T_1)}$, $\mathbf{dist(K_4, T_2)}$, $\mathbf{dist(K_4, T_3)}$)	Min($dist(K_3, T_1)$, $dist(K_3, T_2)$)
Min($ang(K_2, K_1, T_1)$, $ang(K_2, K_1, T_2)$, $\mathbf{ang(K_2, K_1, T_3)}$)	Min($ang(K_2, K_1, T_1)$, $ang(K_2, K_1, T_2)$)
Min($ang(K_3, K_1, T_1)$, $ang(K_3, K_1, T_2)$, $\mathbf{ang(K_3, K_1, T_3)}$)	Min($ang(K_3, K_1, T_1)$, $ang(K_3, K_1, T_2)$)
Min($\mathbf{ang(K_4, K_1, T_1)}$, $\mathbf{ang(K_4, K_1, T_2)}$, $\mathbf{ang(K_4, K_1, T_3)}$)	Min($ang(K_3, K_1, T_1)$, $ang(K_3, K_1, T_2)$)

Table 2: This table describes the mapping between states in 4 vs. 3 to states in 3 vs. 2. The distance between a and b is denoted as $dist(a, b)$; the angle made by a, b, and c, where b is the vertex, is denoted by $ang(a, b, c)$; and values not present in 3 vs. 2 are in bold. Relevant points are the center of the field C, keepers K_1-K_4, and takers T_1-T_3, where players are ordered by increasing distance from the ball.

Table 3 describes the inter-task mappings used to transfer between Knight Joust and 4 vs. 3 Keepaway. Our hypothesis was that the Knight Joust player would learn to move North when possible and jump to the side when necessary, which could be similar to holding the ball in Keepaway when possible and passing when necessary.

5.3 Constructing ρ_{CMAC} and ρ_{RBF}

The CMAC function approximator takes a state and an action and returns the expected long-term reward. The learner can evaluate each possible action for the current state and then use π to choose one. We construct a ρ_{CMAC} and use it so that when the learner considers a 4 vs. 3 action, the weights for the activated tiles are not zero but instead are initialized by $Q_{(3vs2,final)}$. To accomplish this, we copy weights learned in the source CMAC into weights in a newly initialized target CMAC, using χ_X and χ_A. Algorithm 1 describes the process in detail.

χ_X: 4 vs. 3 to Knight Joust

4 vs. 3 state variable	Knight Joust state variable
$dist(K_1,T_1)$	$dist(P,O)$
Min($ang(K_2,K_1,T_1)$, $ang(K_2,K_1,T_2)$, $ang(K_2,K_1,T_3)$)	$ang(West)$
Min($ang(K_3,K_1,T_1)$, $ang(K_3,K_1,T_2)$, $ang(K_3,K_1,T_3)$)	$ang(East)$
Min($ang(K_4,K_1,T_1)$, $ang(K_4,K_1,T_2)$, $ang(K_4,K_1,T_3)$)	$ang(East)$
All other Keepaway variables	ϕ

χ_A: 4 vs. 3 to Knight Joust

4 vs. 3 action	Knight Joust action
Hold Ball	Forward
Pass to closest teammate	$Jump_W$
Pass to second closest teammate	$Jump_E$
Pass to third closest teammate	$Jump_E$

Table 3: This table describes the mapping between state variables and actions from 4 vs. 3 to Knight Joust. Note that the we have made Jump West in the Knight Joust correspond to passing to K_2 and Jump East correspond to passing to K_3, but either is reasonable, as long as the state variables and actions are consistent.

Note that this target CMAC will initially be unable to distinguish between some states and actions because the inter-task mappings allow duplication of values. For instance, the weights corresponding to the tiles that are activated for the "Pass to second closest teammate" in the source task are copied into the weights for the tiles that are activated to evaluate the "Pass to second closest teammate" action and the "Pass to third closest teammate" in the target task. The 4 vs. 3 agents are initially unable to distinguish between these two actions. In other words, because the values for the weights corresponding to the two 4 vs. 3 actions are the same, $Q_{(4vs3,initial)}$ will evaluate both actions as having the same expected return. The 4 vs. 3 agents will therefore have to learn to differentiate these two actions as they learn in the target task.

Algorithm 1 APPLICATION OF ρ_{CMAC}

for each non-zero weight, w_i in the source CMAC **do**
  $x_{source} \leftarrow$ value of state variable corresponding to tile i
  $a_{source} \leftarrow$ action corresponding to i
  for each value x_{target} such that $\chi_X(x_{target}) = x_{source}$ **do**
    for each value a_{target} such that $\chi_A(a_{target}) = a_{source}$ **do**
      $j \leftarrow$ the tile in the target CMAC activated by x_{target}, a_{target}
      $w_j \leftarrow w_i$
$w_{Average} \leftarrow$ average value of all non-zero weights in the target CMAC
for each weight w_j in the target CMAC **do**
  if $w_j = 0$ **then**
    $w_j \leftarrow w_{Average}$

As a final step (Algorithm 1, lines 8–11), any weights which have not been initialized by ρ_{CMAC} are set to the average value of all initialized weights. The 3 vs. 2 training was likely not exhaustive and therefore some weights which may be used in 4 vs. 3 would otherwise remain uninitialized. Tiles which correspond to every value in the new 4 vs. 3 state vector have thus been initialized to values determined via training in 3 vs. 2 and can therefore be considered in the computation. This averaging effect is discussed further in Section 6 and has the effect of allowing agents in the target task to learn faster.

ρ_{RBF} is constructed similarly to ρ_{CMAC}. The main difference between the RBF and CMAC function approximators are how weights are summed together to produces values, but the weights have similar structure in both function approximators. For a given state variable, a CMAC sums one weight per tiling. An RBF differs in that it sums multiple weights for each tiling, where weights are multiplied by the Gaussian function $\phi(x - c_i)$. Thus when using ρ_{RBF} we copy weights following the same schema as in ρ_{CMAC} in Algorithm 1.

5.4 Constructing ρ_{ANN}

To construct a (fully connected, feedforward) neural network for the 4 vs. 3 target task, the 13-20-3 network from 3 vs. 2 is first augmented by adding 6 inputs and 1 output node. The weights connecting inputs 1–13 to the hidden nodes are copied over from the 13-20-3 network. Likewise, the weights from hidden nodes to outputs 1–3 are copied over to the 19-20-4 network. Weights from inputs 14-19 to the hidden nodes correspond to the new state variables and are copied over from the analogous 3 vs. 2 state variable, according to χ_X. The weights from the hidden nodes to the novel output are copied over from the analogous 3 vs. 2 action, according to χ_A. Every weight in the 19-20-4 network is therefore set to an initial value based on the trained 13-20-3 network. Algorithm 2 describes this process in detail. We define the function ψ to map nodes in the two networks:

$$\psi(n) = \begin{cases} \chi_X(n), & \text{if } n \text{ is an input} \\ \chi_A(n), & \text{if } n \text{ is an output} \\ \delta(n), & \text{if } n \text{ is a hidden node} \end{cases}$$

where a function δ represents the correspondence between these hidden nodes ($\delta(h_{target}) = h_{source}$). In our case the number of hidden nodes used are the same in both tasks. Therefore, in practice ψ("n^{th} hidden node in the source network") = "n^{th} hidden node in the target network."

Whereas ρ_{CMAC} and ρ_{RBF} copied many weights (hundreds or thousands, where increasing the amount of 3 vs. 2 training will increase the number of learned non-zero weights), ρ_{ANN} always copies the same number of weights regardless of training. In fact, ρ_{ANN} initializes only 140 new weights (in addition to the 320 weights that existed in 3 vs. 2) in the 4 vs. 3 representation and is therefore in some sense simpler than the other ρs.

Algorithm 2 APPLICATION OF ρ_{ANN}

```
1: for each pair of nodes n_i, n_j in ANN_target do
2:    if link(ψ(n_i), ψ(n_j)) exists in ANN_source then
3:       Set link(n_i, n_j) in ANN_target to have weight of link(ψ(n_i), ψ(n_j)) in ANN_source
```

5.5 Q-value Reuse

The three ρs previously introduced are specific to particular function approximators. In this section we introduce a different approach, *Q-value Reuse*, to transfer between a source and target. Rather than initialize a function approximator in the target task with values learned in the source task, we instead reuse the entire learned source task's Q-values. A copy of the source task's function approximator is retained so that it can calculate the source task's Q-values for any state, action pair: $Q_{sourceFA} : S \times A \mapsto \mathbb{R}$. When computing Q-values for the target task, we first map the target task state and action to the source task's state and action via the inter-task mappings. The computed Q-value is a combination of the output of the source task's saved function approximator and the target task's current function approximator:

$$Q(s,a) = Q_{sourceFA}(\chi_X(s), \chi_A(a)) + Q_{targetFA}(s,a)$$

Sarsa updates in the target task are computed as normal, but only the target function approximator's weights are eligible for updates. Note that if $\chi_X(s)$ or $\chi_A(a)$ were undefined for a certain s,a pair in the target task, $Q(s,a)$ would equal $Q_{targetFA}(s,a)$.

Q-value Reuse may be considered a type of *reward shaping* (Colombetti and Dorigo, 1993; Mataric, 1994): we are able to directly use the expected rewards from the source task to bias the learner in the target task. This method has two advantages. First, it is not function-approximator specific, and could, in theory, be used to transfer between different function approximators as well as between different tasks. Second, there is no initialization step needed between learning the two tasks. However, drawbacks include an increased lookup time and larger memory requirements. Such requirements will grow linearly in the number of transfer steps; while they are not substantial with a single source task, they may become prohibitive when using multiple source tasks or when performing doing multi-step transfer (such as shown later in Section 7.2).

6. Experimental Results: 3 vs. 2 Keepaway to 4 vs. 3 Keepaway

This section discusses the results of our transfer experiments between the 3 vs. 2 and 4 vs. 3 Keepaway tasks using our two metrics, training time reduction in the target task and total training time reduction. Section 6.1 shows the success of transfer when the 3 vs. 2 is used as a source task to learn 4 vs. 3. Section 6.2 includes additional analysis of these results. Section 6.3 demonstrates transfer between 3 vs. 2 and 4 vs. 3 CMAC players using Q-value Reuse.

6.1 Transferring via ρ from 3 vs. 2 Keepaway into 4 vs. 3 Keepaway

Having constructed three ρs that transform the learned action-value functions, we can now set $Q_{(4vs3,initial)} = \rho(Q_{(3vs2,final)})$ between Keepaway agents with CMAC, RBF, or ANN function approximation. We do not claim that these initial action-value functions are correct (and empirically they are not), but instead that the constructed action-value functions allow the learners to more quickly discover a better-performing policy.

In this section we show the results of learning 4 vs. 3 Keepaway, both without transfer and after using TVITM with varying amounts of 3 vs. 2 training. Analyses of learning times required to reach

CMAC Learning Results

# 3 vs. 2 Episodes	Ave. 3 vs. 2 Time	Ave. 4 vs. 3 Time	Ave. Total Time	Std. Dev.
0	0	30.84	30.84	4.72
10	0.03	24.99	25.02	4.23
50	0.12	19.51	19.63	3.65
100	0.25	17.71	17.96	4.70
250	0.67	16.98	**17.65**	4.82
500	1.44	17.74	19.18	4.16
1000	2.75	16.95	19.70	5.5
3000	9.67	9.12	18.79	2.73
6000	21.65	**8.56**	30.21	2.98

Table 4: Results showing that learning Keepaway with a CMAC and applying transfer via inter-task mapping reduces training time (in simulator hours) for CMAC players. Minimum learning times for reaching the 11.5 second threshold are bold. As source task training time increases, the required target task training time decreases. The total training time is minimized with a moderate amount of source task training.

threshold performance levels[6] show that agents utilizing CMAC, RBF, and ANN function approximation are all able to learn faster in the target task by using ρ_{CMAC}, ρ_{RBF}, and ρ_{ANN}, respectively.

Tables 4 and 5 show learning times to reach a threshold performance and verify that a CMAC, an RBF, and an ANN successfully allow independent players to learn to hold the ball from opponents when learning without transfer; agents utilizing these three function approximation methods are able to successfully attain the 4 vs. 3 threshold performance.[7]

This result shows that a CMAC is more efficient than an ANN trained with backprop, another obvious choice. We posit that this difference is due to the CMAC's property of *locality*. When a particular CMAC weight for one state variable is updated during training, the update affects the output value of the CMAC for other nearby state variable values. The width of the CMAC tiles determines the generalization effect and outside of this tile width, the change has no effect. Contrast this with the non-locality of an ANN. Every weight is used for the calculation of an action-value function, regardless of how close two inputs are in state space. Any update to a weight in the ANN must necessarily change the final output of the network for every set of inputs. Therefore it may take the ANN longer to settle into an effective configuration. Furthermore, the ANNs use many fewer weights than the CMAC and RBF learners, which may have allowed for faster learning at the cost of reduced performance of the final policy.

The RBF function approximator had the best performance of the three when learning without transfer (i.e., the top row of each table). The RBF shares the CMAC's locality benefits, but is also able to generalize more smoothly due to the Gaussian summation of weights.

To test the effect of using transfer with a learned 3 vs. 2 action-value function, we train a set of keepers for a number of 3 vs. 2 episodes, save the function approximator's weights ($Q_{(3vs2,final)}$)

6. Our results hold for other threshold times as well, provided that the threshold is not initially reached without training and that learning will enable the keepers' performance to eventually cross the threshold.
7. All times reported in this article refer to simulator time, which is roughly twice that of the wall clock time. We only report sample complexity and not computational complexity; the running time for our learning methods is negligible compared to that of the RoboCup Soccer Server.

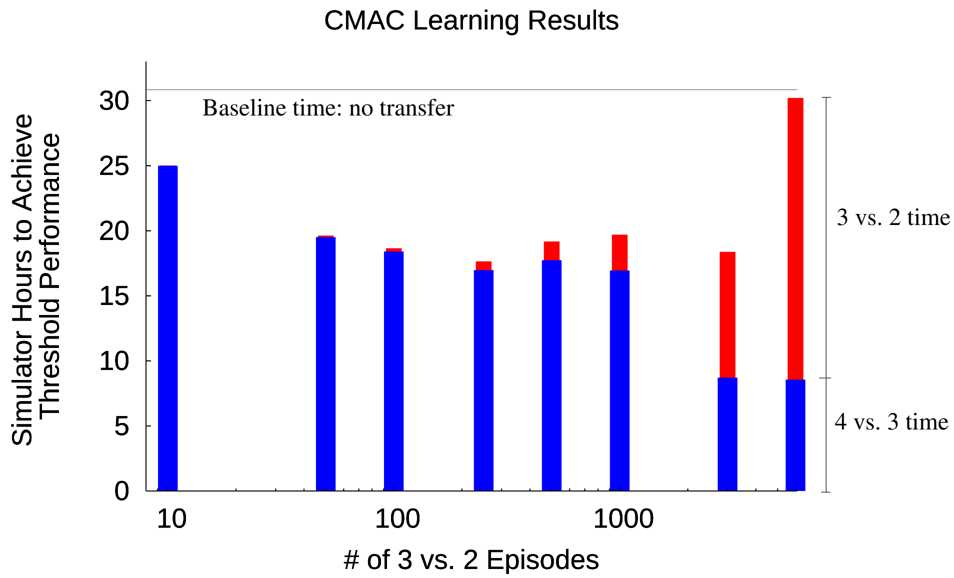


Figure 7: A graph of Table 4 where the x-axis uses a logarithmic scale. The thin bars show the amount of time spent training in the source task, the thick bars show the amount of time spent training in the target task, and their sum represents the total time. The target task training time is reduced as more time is spent training in the source task. The total time is minimized when using a moderate amount of source task training.

from a random 3 vs. 2 keeper, and use the weights to initialize all four keepers[8] in 4 vs. 3 so that $Q_{(4vs3,initial)} \leftarrow \rho(Q_{(3vs2,final)})$. Then we train on the 4 vs. 3 Keepaway task until the average hold time for 1,000 episodes is greater than some performance threshold. Recall that in section 4.5 we specify a threshold of 11.5 seconds in the case of CMAC and RBF function approximators and 10.0 seconds for ANNs as neural network agents were unable to learn as effectively.

To determine if Keepaway players using CMAC function approximation can benefit from transfer, we compare the time it takes agents to learn the target task after transferring from the source task with the time it takes to learn the target task without transfer. The result tables show different amounts of source task training time, where the minimal learning times are in bold. The top row of each table represents learning the task without transfer and thus any column with transfer times lower than the top row shows beneficial transfer. Our second goal of transfer would be met if the total training time in both tasks with transfer was less than learning without transfer in the target task. Table 4 reports the average time spent training in 4 vs. 3 with CMAC function approximation to achieve an 11.5 second average hold time after different amounts of 3 vs. 2 training. Column two reports the time spent training on 4 vs. 3 while the third column shows the total time to train 3 vs. 2 and 4 vs. 3. As can be seen from the table, spending time training in the simpler 3 vs. 2 domain can cause the learning time for 4 vs. 3 to decrease. To overcome the high amounts of noise in our evaluation we run at least 25 independent trials for each data point reported.

8. We do so under the hypothesis that the policy of a single keeper represents all of the keepers' learned knowledge. Though in theory the keepers could be learning different policies that interact well with one another, so far there is no evidence that they do. One pressure against such specialization is that the keepers' start positions are randomized. There appears to be specialization when each keeper starts in the same location every episode.

RBF and ANN Learning Results

# of 3 vs. 2 Episodes	Ave. RBF 4 vs. 3 Time	Ave. RBF Total Time	Standard Deviation	Ave. ANN 4 vs. 3 Time	Ave. ANN Total Time	Standard Deviation
0	19.52	19.52	6.03	33.08	33.08	16.14
10	18.99	19.01	6.88	19.28	**19.31**	9.37
50	19.22	19.36	5.27	22.24	22.39	11.13
100	18.00	18.27	5.59	23.73	24.04	9.47
250	18.00	18.72	7.57	22.80	23.60	12.42
500	16.56	18.12	5.94	19.12	20.73	8.81
1,000	**14.30**	**17.63**	3.34	**16.99**	20.19	9.53
3,000	14.48	26.34	5.71	17.18	27.19	10.68

Table 5: Results from learning Keepaway with different amounts of 3 vs. 2 training time (in simulator hours) indicates that ρ_{RBF} and ρ_{ANN} can reduce training time for RBF players (11.5 second threshold) and ANN players (10.0 second threshold). Minimum learning times for each method are in bold.

The potential of TVITM is evident in Table 4 and Figure 7. To analyze these results, we conduct a number of Student's t-tests to determine if the differences between the distributions of learning times for the different settings are significant. These tests confirm that the differences in the distributions of 4 vs. 3 training times when using TVITM are statistically significant ($p < 0.05$) when compared to training 4 vs. 3 without transfer. Not only is the time to train the 4 vs. 3 task decreased when we first train on 3 vs. 2, but the total training time is less than the time to train 4 vs. 3 without transfer. We can therefore conclude that in the Keepaway domain, training first on a simpler source task can increase the rate of learning enough that the total training time is decreased when using a CMAC function approximator. It is not obvious how to choose the amount of time to spend learning the source task to minimize the total time and this an optimization will be left for future work (see Section 8).

Analogous experiments for Keepaway players using RBF and neural network function approximation are presented in Table 5. Again, successful transfer is demonstrated as both the transfer agents' target task training time and the transfer agent's total training time are less than the time required to learn the target task without transfer. All numbers reported are averaged over at least 25 independent trials; both 4 vs. 3 time and total time can be reduced with TVITM. For the RBF players, all TVITM 4 vs. 3 results using at least 500 3 vs. 2 episodes show a statistically significant difference from those that learn without transfer ($p < 0.05$), while the learning trials that used less than 500 source task episodes did not significantly reduce the target task training time. The difference in all 4 vs. 3 training times for the ANN players between using TVITM and training without transfer is statistically significant ($p < 0.05$).

The RBF function approximator yielded the best learning rates for 3 vs. 2 Keepaway, followed by the CMAC function approximator, and lastly the ANN trained with backpropagation. However, TVITM provided the least percentage speedup to the RBF agents. One possible hypothesis is that transfer is less useful to the best learners. One explanation is that if a particular representation is poorly suited for a task, transfer may be able to provide proportionally more speedup because it is that much further from an "optimal learner." Nonetheless, while some function approximators get more or less benefit from TVITM, it is clear that all three are able learn the target task faster with the

Ablation Studies with ρ_{CMAC}

Transfer Functional	# of 3 vs. 2 Episodes	Ave. 4 vs. 3 Time	Standard Deviation
No Transfer	0	30.84	4.72
ρ_{CMAC}	100	17.71	4.70
ρ_{CMAC}	1000	16.95	5.5
ρ_{CMAC}	3000	9.12	2.73
$\rho_{CMAC,\ No\ Averaging}$	100	25.68	4.21
$\rho_{CMAC,\ No\ Averaging}$	3000	9.53	2.28
only averaging	100	19.06	6.85
only averaging	3000	10.26	2.42
$\rho_{CMAC,\ Ave\ Source}$	1000	15.67	4.31

Table 6: Results showing that transfer with the full ρ_{CMAC} outperforms using ρ_{CMAC} without the final averaging step, using only the averaging step of ρ_{CMAC}, and when averaging weights in the source task before transferring the weights.

technique, and that more training in the source task generally reduces the time needed to learn the target task.

6.2 Understanding ρ_{CMAC}'s Benefit

To better understand how TVITM uses ρ_{CMAC} to reduce the required training time in the target task, and to isolate the effects of its various components, this section details a number of supplemental experiments.[9]

To help understand how ρ_{CMAC} enables transfer we isolate its two components. We first ablate the functional so that the final averaging step (Algorithm 1, lines 8–11), which places the average weight into all zero weights, is removed. We anticipated that the benefit from transfer would be increasingly degraded, relative to using the actual ρ_{CMAC}, as fewer numbers of training episodes in the source task were used. The resulting 4 vs. 3 training times were all shorter than training without transfer, but longer than when the averaging step was incorporated. The relative benefit of our ablated ρ_{CMAC} is greater after greater numbers of source task episodes; the averaging step appears to have given initial values to weights in the state/action space that have never been visited with low numbers of source episodes and thus imparts some bias in the target task even with very little 3 vs. 2 training. Over time more of the state space in the source task is explored and thus our ablated functional performs quite well. This result shows that the averaging step is most useful with less source task training, but becomes less so as more source experience is accumulated (see Table 6 for result details).

If we perform *only* the averaging step from ρ_{CMAC} on learners trained in the *target* task, we can determine how important this step is to our method's effectiveness. Applying the averaging step

9. Informal experiments showed that the CMAC and RBF transfer results were qualitatively similar, which is reasonable given the two function approximator's many similarities. Thus we expect that the supplemental experiments in this section would yield qualitatively similar results if we used RBFs rather than CMACs. While our results demonstrate that all three function approximators can successfully transfer knowledge, we focus our supplementary experiments on CMAC function approximation so that our transfer work can be directly comparable to previous work in Keepaway, which also used CMACs (Stone et al., 2005).

Time required for CMAC 4 vs. 3 players to reach 11.5 sec. hold time

Initial CMAC weight	Ave. Learning Time	Standard Deviation
0	30.84	4.72
0.5	35.03	8.68
1.0	N/A	N/A
Each weight randomly selected from the uniform distribution from [0,1.0]	28.01	6.93

Table 7: 10 independent trials are averaged for different values for initial CMAC weights. None of the trials with initial weights of 1.0 were able to reach the 11.5 threshold within 45 hours, and thus are shown as N/A above.

causes the total training time to decrease below that of training 4 vs. 3 without transfer, but again the training times are longer than running ρ_{CMAC} on weights trained in 3 vs. 2. This result confirms that both parts of ρ_{CMAC} contribute to reducing 4 vs. 3 training time and that training on 3 vs. 2 is more beneficial for reducing the required 4 vs. 3 training time than training on 4 vs. 3 and applying ρ_{CMAC} (see Table 6 for result details).

The averaging step in ρ_{CMAC} is defined so that the average weight in the *target* CMAC overwrites all zero-weights. We also conducted a set of 30 trials which modified ρ_{CMAC} so that the average weight in the *source* CMAC is put into all zero-weights in the target CMAC, which is possible when agents in the source task know that their saved weights will be used for TVITM. Table 6 shows that when the weights are averaged in the source task ($\rho_{CMAC,\ Ave\ Source}$) the performance is not statistically different ($p < 0.05$ from TVITM when averaging in the target task (See Table 4).

To verify that the 4 vs. 3 CMAC players were benefiting from TVITM and not from having non-zero initial weights, we initialized CMAC weights uniformly to 0.5 in one set of experiments, 1.0 uniformly in a second set of experiments, and then to random numbers uniformly distributed from 0.0-1.0 in a third set of experiments. We do so under the assumption that 0.0, 0.5, and 1.0 are all reasonable initial values for weights (although in practice 0.0 is most common). The learning time was never statistically better than learning with weights initialized to zero, and in some experiments the non-zero initial weights decreased the speed of learning. Haphazardly initializing CMAC weights may hurt the learner but systematically setting them through TVITM is beneficial. Thus we conclude that the benefit of transfer is not a byproduct of our initial setting of weights in the CMAC (see Table 7 for result details).

To further test the sensitivity of the ρ_{CMAC} function, we change it in two different ways. We first defined $\rho_{modified}$ by modifying χ_A so that instead of mapping the novel target task action "Pass to second third keeper" into the action "Pass to second closest keeper," we instead map the novel action into "Hold ball." Now $Q_{4vs3,initial}$ will initially evaluate "pass to third closest keeper" and "hold ball" as equivalent for all states. Second, we modify χ_A and χ_X so that state variables and actions not present in 3 vs. 2 are not initialized in the target task. Using these new inter-task mappings, we construct ρ_{3vs2}, a functional which copies over information learned in 3 vs. 2 exactly but assigns the average weight to all novel state variables and actions in 4 vs. 3.

When using this $\rho_{modified}$ to initialize weights in 4 vs. 3, the total training time increased relative to the normal ρ_{CMAC} but still outperformed training without transfer. Similarly, ρ_{3vs2} is able to outperform learning without transfer, but underperforms the full ρ_{CMAC}, particularly for higher amounts of training in the source task.

Testing Sub-optimal Inter-task Mappings

Transfer Functional	# of 3 vs. 2 Episodes	Ave. 4 vs. 3 Time	Standard Deviation
ρ_{CMAC}	100	17.71	4.70
ρ_{CMAC}	3000	9.12	2.73
$\rho_{modified}$	100	21.74	6.91
$\rho_{modified}$	3000	10.33	3.21
ρ_{3vs2}	100	18.90	3.73
ρ_{3vs2}	3000	12.00	5.38

Table 8: Results showing that transfer with the full ρ_{CMAC} outperforms using sub-optimal or incomplete inter-task mappings.

Choosing non-optimal inter-task mappings when constructing ρ seems to have a detrimental, but not necessarily disastrous, effect on the training time. This result shows that the structure of ρ is indeed important to the success of transfer (see Table 8 for result details).

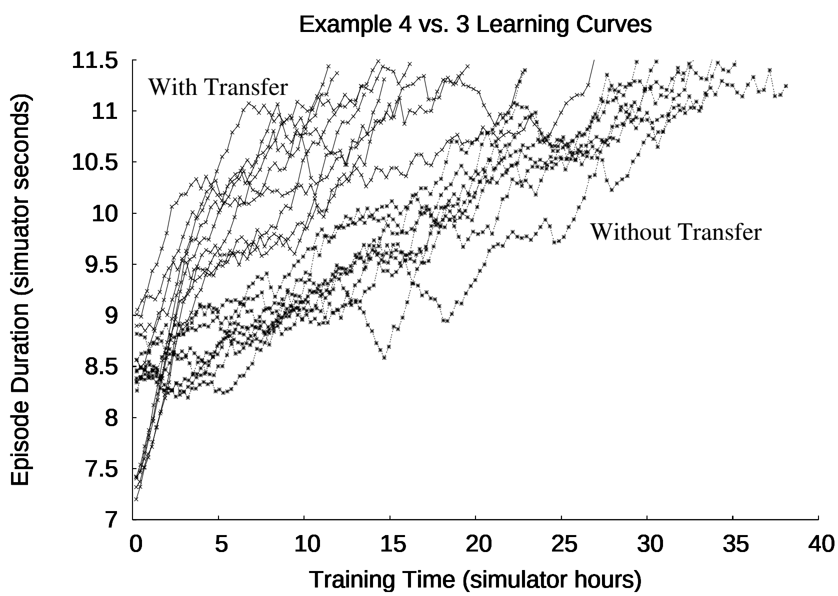


Figure 8: Representative learning curves, showing that transfer via inter-task mapping does not significantly increase the performance of the initial policy in 4 vs. 3, but enables faster learning by biasing the learner towards a productive part of the function approximator's weight space. Eight 4 vs. 3 learning curves without transfer are compared to eight learning curves in 4 vs. 3 after transferring from 250 episodes of 3 vs. 2.

Interestingly, when the CMACs' weights are loaded into the keepers in 4 vs. 3, the initial hold times of the keepers do not differ significantly from those of keepers with uninitialized CMACs (i.e., CMACs where all weights are initially set to zero). The information contained in the function approximators' weights prime the 4 vs. 3 keepers to more quickly learn their task by biasing

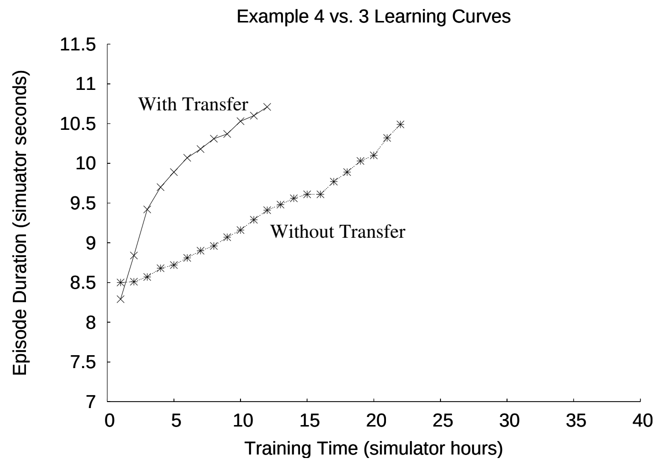


Figure 9: The average performance from the learners in Figure 8 shows a clear benefit from using transfer.

Initial Performance in 4 vs. 3 with CMAC Function Approximation

# of 3 vs. 2 Episodes	Ave. Performance (sec.)	Standard Deviation
0	8.46	0.17
1000	8.92	1.48
6000	9.24	1.15

Table 9: This graph shows the difference in initial performance between 4 vs. 3 players with and without transfer. 40 independent trials are averaged for each setting and the differences in initial performance (i.e., initial episode lengths) are small. A Student's t-test shows that 8.46 and 8.92 are not statistically different while 8.46 and 9.24 are ($p < 0.05$).

their search, even though the knowledge we transfer is of limited initial value. See Figure 8 for representative learning curves and Table 9 for result details.[10]

TVITM relies on effectively reusing learned data in the target task. We hypothesized that successfully leveraging this data may be effected by ε, Sarsa's exploration parameter, which balances exploration with exploitation. Recall that we had initially chosen an exploration rate of 0.01 (1%) to be consistent with past research. Table 10 shows the results of learning 3 vs. 2 Keepaway with $\varepsilon = 0.01$ for 1,000 episodes, utilizing ρ_{CMAC}, and then learning 4 vs. 3 Keepaway with various settings for ε. The results show that of these 4 additional settings for ε, only $\varepsilon = 0.05$ is statistically better than the default rate of 0.01. To further explore this last result we ran a series of 30 trials of learning 4 vs. 3 from scratch with the value of $\varepsilon = 0.05$ and found that there was a significant difference from learning 4 vs. 3 from scratch with $\varepsilon = 0.01$. Thus the speedup for this particular setting of ε in transfer, relative to the default value, is explained by the increased learning speed

10. The more similar the source and target tasks are, the more of an immediate performance improvement we would expect to see. For example, in the degenerate case where the source and target task are identical, the initial performance in the target task will be equivalent to the final performance in the source task. However, in such a situation, reducing the total time—our more difficult transfer goal—would prove *impossible*.

Varying the Exploration in 4 vs. 3

ϵ in 4 vs. 3	# 3 vs. 2 Episodes	Ave. 4 vs. 3 Time	Standard Deviation
0.001	1000	22.06	10.52
0.005	1000	19.22	8.31
0.01 (default)	1000	16.95	5.5
0.05	1000	12.84	2.55
0.1	1000	18.40	5.70
0.01 (default)	0	30.84	4.72
0.05	0	17.57	2.59

Table 10: The first five rows detail experiments where 3 vs. 2 is first learned with $\epsilon = 0.01$ and then transfer is used to speed up learning in 4 vs. 3. 30 independent trials are averaged for each setting of ϵ in the target task. The last two rows show the results of learning 4 vs. 3 without transfer for two settings of ϵ. These results show that the amount of exploration in the target task affects learning speed both with and without transfer.

without transfer. This experiment does suggest, however, that the previously determined value of $\epsilon = 0.01$ is not optimal for Sarsa with CMAC function approximation in the Keepaway domain.

From these supplemental results we conclude:

1. Both parts of ρ_{CMAC}—copying weights based on χ_X and χ_A, and the final averaging step—contribute to the success of TVITM. The former gives more benefit after more training is completed in the source task and the second helps when less knowledge is gained in the source task before transfer.

2. Using ρ_{CMAC} is superior to weights initialized to zero (training without transfer), as well as weights initialized to 0.5, 1.0 and [0,1.0], three other reasonable initial settings.

3. A suboptimal or incomplete transfer functional, such as $\rho_{modified}$ and ρ_{3vs2}, allows TVITM to speed up learning, but not as much as the more correct ρ_{CMAC}.

4. Players initialized by TVITM in the source task do not initially outperform uninitialized players in the target task, but are able to learn faster.

6.3 Transferring via Q-value Reuse from 3 vs. 2 Keepaway into 4 vs. 3 Keepaway

In the previous sections we showed that TVITM was capable of transferring from 3 vs. 2 into 4 vs. 3 by using ρ_{CMAC}, ρ_{RBF}, and ρ_{ANN}. In this section we use TVITM with Q-value Reuse (Section 5.5) between CMAC players in 3 vs. 2 and 4 vs. 3. Recall that Q-value Reuse directly uses a learned function approximator from the source task when calculating Q-values in the target task.

Table 11 shows the results of using Q-value Reuse. Each transfer experiment shows the average of 30 independent trials. Both the 4 vs. 3 and total times are statistically different from learning without transfer ($p < 0.05$, via Student's t-tests). As when using ρ_{CMAC} for transfer (Table 4), more 3 vs. 2 episodes correspond to a decrease in the time required for 4 vs. 3 players to reach the 11.5 second threshold performance.

The reduction in transfer efficacy, relative to using ρ_{CMAC}, is due to the averaging step in ρ_{CMAC}. As we showed in the previous section, this averaging step has an impact on the target task learning

Q-value Reuse between CMAC players

# of 3 vs. 2 Episodes	Ave. 4 vs. 3 Time	Ave. Total Time	Standard Deviation
0	30.84	30.84	4.72
10	28.18	28.21	5.04
50	28.0	28.13	5.18
100	26.8	27.06	5.88
250	24.02	24.69	6.53
500	22.94	24.39	4.36
1,000	22.21	**24.05**	4.52
3,000	**17.82**	27.39	3.67

Table 11: Results from learning 3 vs. 2 with CMAC players for different numbers of episodes and then utilizing the learned 3 vs. 2 CMAC directly while learning 4 vs. 3. Minimum learning times for reaching the 11.5 second threshold are bold.

times. However, in Q-value Reuse we treat the source task function approximator as a "black box" and thus do not permute its values, nor use it to set the initial values of the target task's function approximator. These results suggest that if the source and target function approximators are different, Q-value Reuse may be appropriate. However, if memory is limited, running time is critical, and/or multiple transfer steps are involved (such as transferring from 3 vs. 2 to 4 vs. 3, and then from 4 vs. 3 to 5 vs. 4), then using a ρ is preferable.

7. Experimental Results: Different Transfer Tasks

In this section of the article we show that TVITM can work between a variety of different source/target task pairs. Section 7.1 presents results of transfer between 3 vs. 2 and 4 vs. 3 agents with different abilities. Section 7.2 demonstrates that transfer can also be used to reduce both target and total training time for 5 vs. 4 Keepaway, and gives some initial results for 6 vs. 5 Keepaway, demonstrating that TVITM can scale to more complex tasks. Section 7.3 shows the results of transferring from two variants of 3 vs. 2 into 4 vs. 3 to demonstrate how the relatedness of source and target tasks effect the efficacy of TVITM. Lastly, Section 7.4 shows that TVITM can successfully transfer between the Knight Joust task and 4 vs. 3, two tasks with very different characteristics.

7.1 3 vs. 2 Keepaway and 4 vs. 3 Keepaway with Differing Player Abilities

The results in Section 6.1 show that Q-values learned in 3 vs. 2 can be successfully used to speed up learning in 4 vs. 3. In this section we test how robust TVITM is to changes in the agent's abilities. In addition to changing the number of players between the source and target tasks, other variations such as the size of the field, wind, and player ability can be modified. It is a qualitatively different challenge to use TVITM to speed up learning between two tasks where the agents' actions have different effects (i.e., T has been modified so that the actions are qualitatively different) in addition to different state and action spaces. We choose to test the robustness of TVITM by changing the passing

Learning Results with Different Actuators

# of 3 vs. 2 Episodes	3 vs. 2 Time	3 vs. 2 Actuator Accurate?	Ave. 4 vs. 3 Time	Standard Deviation	4 vs. 3 Actuator Accurate?
0	0	N/A	30.84	16.14	Yes
500	1.44	Yes	17.74	4.16	Yes
3000	9.67	Yes	9.12	2.73	Yes
0	0	N/A	54.15	6.13	No
500	1.23	No	37.3	9.24	No
3000	8.36	No	29.86	9.20	No
500	1.37	Yes	37.54	7.48	No
3000	9.45	Yes	24.17	5.54	No
500	1.3	No	18.46	3.93	Yes
3000	8.21	No	13.57	3.64	Yes

Table 12: Results showing transfer via inter-task mapping benefits CMAC players utilizing ρ_{CMAC} with two kinds of actuators. These results demonstrate that transfer can succeed even when actions in the source and target tasks are qualitatively different. The results in rows 1–3 are from Table 4.

actuators on some sets of agents so that the passes are less accurate.[11] We show in this section that TVITM speeds up learning, relative to learning without transfer, in the following scenarios:

1. Learning 4 vs. 3 with damaged passing actuators after transferring from 3 vs. 2 players with damaged passing actuators.

2. Learning 4 vs. 3 with a normal passing actuators after transferring from 3 vs. 2 players with damaged passing actuators.

3. Learning 4 vs. 3 with damaged passing actuators after transferring from 3 vs. 2 players with normal passing actuators.

Accurate CMAC players learning without transfer in 4 vs. 3 take only 30.1 hours to reach the threshold performance level (row 1 of Table 12). When we allow sets of CMAC keepers to learn 4 vs. 3 without transfer while using the less accurate pass mechanism, the average time to reach an average performance of 11.5 seconds is 54.2 hours (row 4 of Table 12). We are also able to use the same ρ_{CMAC} to speed up learning in the target task when both the target and source tasks have inaccurate actuators (rows 5 and 6). These two results, as well as all other 4 vs. 3 transfer learning times in this table, are statistically significant when compared to learning the relevant 4 vs. 3 task without transfer ($p < 0.05$).

Now consider that we would like to learn the 4 vs. 3 target task with inaccurate passing, but that we have already trained some 3 vs. 2 keepers that learned using an accurate pass action in the source task. As we can see in the third group (i.e., rows 7 and 8) of Table 12, even though the players in the source task have different actuators than in the target task, transfer is able to significantly speed up learning compared to not using transfer.

11. Actuators are changed in the benchmark players by changing the pass action from the default “PassNormal” to “PassFast” which increases the speed of the pass by 50%, reducing accuracy.

This result confirms that the same ρ will allow TVITM to transfer between tasks where not only have S and A changed, but the effect of the actions have also changed qualitatively. This situation is of practical import as well, as many robotic systems experience gradual degradation in performance over time due to wear and tear. If a set of robots with worn down actuators are available, they may still be able to benefit from action-value function transfer of Q-values from learners that have fresh actuators. Alternately, if a set of agents have learned a task and then later want to learn another task but have damaged their actuators since learning the source task, transfer may still increase the speed of learning.

We also perform the inverse experiment where agents in the source task have inaccurate actuators and agents in the target task have normal actuators. We perform TVITM after 500 and 3,000 episodes of 3 vs. 2 with inaccurate passing to initialize the Q-values of agents in 4 vs. 3 with accurate passing. The final two rows in Table 12 again shows using this transfer is a significant improvement over learning without transfer. Thus a fielded agent with worn down actuators would be able to successfully transfer its learned action-value function to agents whose actuators were undamaged. Interestingly, transferring from source keepers that have accurate actuators is more effective than transferring from source keepers that have inaccurate actuators both when the target task has accurate actuators and when it has inaccurate actuators. We posit that this is because it is easier to learn with accurate actuators, which means that more useful information exists to be transfered.

We first showed in Section 6 that transfer from 3 vs. 2 keepers with accurate pass actuators to 4 vs. 3 keepers with accurate pass actuators was successful. In this subsection we demonstrate that transfer works when actuators are inaccurate in both the source and target tasks. It also speeds learning in the target task when transferring from inaccurate 3 vs. 2 players to accurate 4 vs. 3 players or from accurate 3 vs. 2 players to inaccurate 4 vs. 3 players.

Combined, our results show that TVITM is able to speed up learning in multiple target tasks with different state and action spaces, and even when the agents have somewhat different actuators in the two tasks.

7.2 Scaling up to Larger Keepaway Tasks

In this section we show that our method can also be used to speed up the 5 vs. 4 Keepaway task, which provides evidence for scalability to larger tasks. The 5 vs. 4 task is more difficult than the 4 vs. 3 task, as discussed in Section 4.5. In addition to using 4 vs. 3 to speed up learning in 5 vs. 4, we show that TVITM can be used *twice* to learn the 3 vs. 2, 4 vs. 3, and 5 vs. 4 tasks in succession.

Results in Table 13 show that TVITM scales to the 5 vs. 4 Keepaway task. In 5 vs. 4 we say that the task has been learned when the 5 keepers are able to hold the ball for an average of 9.0 seconds over 1,000 episodes. ρ_{CMAC} can be formulated by extending χ_X and χ_A so that they can transfer the action-value function from 4 vs. 3 to 5 vs. 4, analogous to the way it transfers values from 3 vs. 2 to 4 vs. 3. These results are shown in rows 2 and 3 of Table 13.

χ_X and χ_A can also be formulated so that we can use TVITM to speed up 5 vs. 4 after learning 3 vs. 2. For instance, the the target task actions “Pass to Second Closest Teammate”, “Pass to Third Closest Teammate”, and “Pass to Fourth Closest Teammate” are mapped to the source task action “Pass to Second Closest Teammate.” Table 13, rows 4 and 5, show that this mapping formulation is successful. In fact, there is more benefit than transferring from 4 vs. 3. We posit that this is due to the fact that it is easier to learn more in the simpler target task and this outweighs the fact that it is less related to 5 vs. 4 than to 4 vs. 3. Another way to understand this is that in a fixed amount of

CMAC Learning Results in 5 vs. 4

# of 3 vs. 2 Episodes	# of 4 vs. 3 Episodes	Ave. 5 vs. 4 Time	Ave. Total Time	Standard Deviation
0	0	22.58	22.58	3.46
0	500	13.44	14.60	7.82
0	1000	9.66	12.02	4.50
500	0	6.76	**8.18**	1.90
1000	0	6.70	9.66	2.12
500	500	**6.19**	8.86	1.26

Table 13: Results showing that learning Keepaway with a CMAC and applying transfer via inter-task mapping can reduce training time (in simulator hours) for CMAC in 5 vs. 4 with a target performance of 9.0 seconds. All numbers are averaged over at least 25 independent trials.

experience, players in 3 vs. 2 are able to update more weights than 4 vs. 3, measured as a percentage of the total possible number of weights used in the task.

A final refinement is to use a two-step application of TVITM so that 3 vs. 2 runs first. This learned action-value function is used as the initial action-value function in 4 vs. 3 after applying ρ_{CMAC}, and after training the final 4 vs. 3 action-value function is used as the initial action-value function for 5 vs. 4. Using this procedure (Table 13, bottom row) we find that the time to learn 5 vs. 4 is reduced to roughly 27% of learning without transfer. A t-test confirms that the differences between all 5 vs. 4 training times shown are statistically significant ($p < 0.05$) when compared to learning without transfer.

These results clearly show that TVITM allows 5 vs. 4 Keepaway to be learned faster after training on 4 vs. 3 and/or 3 vs. 2. They also suggest that a multi-step process where tasks are made incrementally more challenging may produce faster learning times than a single application of TVITM. As a final result, a similar χ_X, χ_A, and ρ_{CMAC} can be constructed to significantly speed up learning in 6 vs. 5 as well (which also takes place on a $25m \times 25m$ field), as shown in Table 14.

Transfer in 6 vs. 5 with CMAC Function Approximation

# of 5 vs. 4 Episodes	Ave. 6 vs. 5 Time	Ave Total Time	Standard Deviation
0	22.85	22.85	1.71
1000	9.38	11.53	2.38

Table 14: 10 independent trials are averaged for learning 6 vs. 5 with and without transfer from 5 vs. 4. The threshold performance time is 8.0 seconds. A Student's t-test confirms that the difference is statistically significant ($p < 0.05$).

7.3 Variants of 3 vs. 2 Keepaway for Transfer into 4 vs. 3 Keepaway

In this section we introduce two novel variants of the 3 vs. 2 Keepaway task to show how TVITM with ρ_{CMAC} can fail to improve performance relative to learning without transfer.

We first modify 3 vs. 2 so that the reward is defined as $+1$ for each action, rather than $+1$ for each timestep. Players in the *3 vs. 2 Flat Reward* task can still learn to increase the average episode time. We hypothesized that the changes in reward structure would prevent TVITM from successfully improving performance in the standard 4 vs. 3 task because of the different reward structure.

We next modify 3 vs. 2 so that the reward is defined to be -1 for each timestep. The task of *3 vs. 2 Giveaway* is thus very different from Keepaway. Given the available actions, the optimal action for players is for the player closest to the takers to hold the ball until the takers captures it. We hypothesized that using TVITM from Giveaway to 4 vs. 3 Keepaway would produce *negative transfer*, where the required target task training time is increased by using transfer.

Table 15 shows the results of using these two 3 vs. 2 source task variants and compares them to using Keepaway as a source task and to learning without transfer. Transfer from the Flat Reward tasks gives a benefit relative to learning without transfer, but not nearly as much as transferring from 3 vs. 2 Keepaway. Students t-tests determine that transfer after 3,000 episodes of Giveaway is significantly slower than learning without transfer.

Transfer into 4 vs. 3 with ρ_{CMAC}: different source tasks

Source Task	# of 3 vs. 2 Episodes	Ave. 4 vs. 3 Time	Ave. Total Time	Std. Dev.
none	0	30.84	30.84	4.72
Keepaway	1000	16.95	19.70	5.5
Keepaway	3000	9.12	18.79	2.73
Flat Reward	1000	25.11	27.62	6.31
Flat Reward	3000	19.42	28.03	8.62
Giveaway	1000	27.05	28.58	10.71
Giveaway	3000	32.94	37.10	8.96

Table 15: Results compare transferring from three different source tasks. Each line is an average of 30 independent trials. The 3 vs. 2 Flat Reward task improves performance relative to learning without transfer, but less than when transferring from Keepaway. The 3 vs. 2 Giveaway task can decrease 4 vs. 3 performance when it is used as a source task.

7.4 Transferring from Knight Joust to 4 vs. 3 Keepaway

In this section we show that TVITM can successfully transfer between the gridworld Knight Joust task and 4 vs. 3 Keepaway. We use a variant of ρ_{CMAC} to transfer the learned weights, because Knight Joust is learned with a tabular function approximator rather than a CMAC. This representation choice results in changes to the syntax of ρ_{CMAC}, as described in Algorithm 3. Note that this new variant of the transfer functional is not necessitated by the novel target task and that if we learned Knight Joust with a CMAC, the original ρ_{CMAC} would be sufficient for transfer between Knight Joust and 4 vs. 3.

The results in Table 16 report the average of 30 independent trials. The 4 vs. 3 transfer times (in simulator hours) are statistically different from learning without transfer (determined via Student's t-tests). Recall that the wall-clock time of the Knight Joust simulator is negligible and thus, in practice, the 4 vs. 3 time is the same as the total time.

The reader will notice that the number of source task episodes used in these experiments is much larger than other experiments in this paper. The reason for this is two-fold. First, we learn Knight Joust with tabular function approximation, which is significantly slower to learn than a CMAC, for instance, because there is no generalization. Secondly, because the wall-clock time requirements

Algorithm 3 APPLICATION OF ρ_{CMAC} FROM TABULAR FUNCTION APPROXIMATION

$n_{source} \leftarrow$ number of variables in source task
for each non-zero Q-value, q_i in the source task's Q-table **do**
  $a_{source} \leftarrow$ action corresponding to q_i
  for each state variable, x_{source}, in source task **do**
    for each value x_{target} such that $\chi_X(x_{target}) = x_{source}$ **do**
      for each value a_{target} such that $\chi_A(a_{target}) = a_{source}$ **do**
        $j \leftarrow$ the tile in the target CMAC activated by x_{target}, a_{target}
        $w_j \leftarrow (q_i/n_{source})$
$w_{Average}$ = average value of all non-zero weights in the target CMAC
for each weight w_j in the target CMAC **do**
  if $w_j = 0$ **then**
    $w_j \leftarrow w_{Average}$

Transfer from Knight Joust into 4 vs. 3

# of Knight Joust Episodes	Ave. 4 vs. 3 Time	Standard Deviation
0	30.84	4.72
25,000	24.24	16.18
50,000	18.90	13.20

Table 16: Results from using Knight Joust to speed up learning in 4 vs. 3 Keepaway. Knight Joust is learned with Q-learning and tabular function approximation and Keepaway players are learned using Sarsa with CMAC function approximation. Both transfer times are significantly less than learning without transfer, as determined via Student's t-tests ($p < 0.05$).

for this domain were so small, we felt justified in allowing the source task learners run until learning plateaued (which takes roughly 50,000 episodes).

The main importance of these results is showing that TVITM can successfully transfer between different tasks with very different dynamics. Keepaway has stochastic actions, is partially observable, and uses a continuous state space. In contrast, Knight Joust has no stochasticity in the player's actions, is fully observable, and has a discrete state space.

8. Discussion and Future Work

We consider the research reported in this article to be a first step towards autonomous transfer learning. In particular, the results presented in this article serve mainly as an existence proof that TVITM *can* be effective for speeding up learning in a target task after training in a source task. As such, it opens up the door for future research that is necessary to build it into a fully general autonomous transfer method. Specifically,

1. Can χ_X and χ_A be learned?

2. When concerned with the total training time of both tasks, what is the optimal amount of training time to spend in the source task?

3. How can an agent determine if two tasks are related so that transfer will be able to impart some advantage?

In this work we construct ρs for learners from a pair of χ_X, χ_A inter-task mappings for a given pair of tasks. There has been initial progress in learning such mappings, as discussed in the next section. In the future we intend to make this process more automatic or completely autonomous.

Another question that arises from this work is: if the ultimate goal is to learn the target task in the minimal amount of time, how can one determine the optimal amount of training in the source task automatically based on task characteristics? While it is clear from our results that spending more time learning 3 vs. 2 often decreases the amount of time it takes to learn 4 vs. 3 Keepaway, it is unclear how to determine the number of 3 vs. 2 episodes to use *a priori* when the goal is to minimize overall training time. It is likely that such a calculation or heuristic will have to consider the structure of the two tasks, how they are related, and the specifics of the ρ used.

A fundamental difficulty of transfer learning is determining whether and how two tasks are similar. Here we only consider the case where the agents are directed to use previous experience to speed up learning in a new task. In practice, it may be the case that the agent has no experience that is relevant and the optimal approach is to simply learn the target task without transfer.

Consider the problem of learners that have trained in multiple tasks and have built up different action-value functions for each of those tasks. When a new task is presented to the learners, they must now decide from which task to perform the transfer. This situation is analogous to presenting a human a solution to a problem and then ask how to solve a related problem. Research has shown (Gick and Holyoak, 1980) that humans are not good at this type of *analogy* problem. For instance, Gick and Holyoak presented subjects with a source task and a solution. When they later presented a similar target task, most people were initially unable to solve the target problem. However, told to use a strategy similar to that used for a past problem, 90% of the subjects were able to discover the analogy and solve the target problem. It is likely that computerized learners will have similar problems deciding if two tasks are related at all, particularly when agents have built up experience on many different kinds of problems.

Our TVITM methodology relies on being able to find an inter-task mapping between similar states and actions. The mapping should identify state variables and actions that have similar effects on the long-term discounted reward. If, for instance, the Keepaway task were changed so that instead of receiving a reward of $+1$ at every time step, you received a $+10$, the ρ could be trivially modified so that all the weights were multiplied by 10. However, if the reward structure is more significantly changed, to that of *Giveaway* for instance, ρ would need to be dramatically changed, if it could be formulated at all.[12]

We hypothesize that the main requirement for TVITM to successfully transfer is that, on average, at least one of the following is true:

1. The best learned actions in the source task, for a given state, be mapped to the best action in the target task via the inter-task mappings.

2. The average Q-values learned for states are of the correct magnitude in the trained target task's function approximator.

12. The "obvious" solution of multiplying all weights by -1 would not work, for instance. In Keepaway a keeper typically learns to hold the ball until a taker comes within roughly $6m$. Thus, if all weights from this policy were multiplied by -1, the keepers would continually pass the ball until a taker came within $6m$. These Giveaway episodes would last much longer than simply forcing the first keeper to the ball to always hold, which is very easily learned.

The first condition will work to bias the learner so that the best actions in the target task are chosen more often, even if these actions' Q-values are incorrect. The second condition will make learning faster because smaller adjustments to the function approximators' weights will be needed to reach their optimal values, even if the optimal actions are not initially chosen. In this work, an example of the first condition being met is that a keeper learns to hold the ball in the source task until forced to pass. Hold is often the correct action in both 3 vs. 2 and 4 vs. 3 when the takers are far away from the ball. The second condition is also met between 3 vs. 2 and 4 vs. 3 by virtue of similar reward structures and roughly similar episode lengths. If either of these conditions were not true, the transfer functional we employed would have to account for the differences (or suffer from reduced transfer efficacy).

It is important to recognize that domain knowledge contained in χ_X and χ_A is required to generate an effective ρ. As our experiments show, simply copying weights without respecting the inter-task mapping is not a viable method of transfer, as our function approximator representations necessarily differ between the two tasks due to changes in S and A. Simply putting the average value of the 3 vs. 2 weights into the 4 vs. 3 function approximator does not give nearly as much benefit as using a ρ which explicitly handles the different state and action values. Likewise, when we used $\rho_{modified}$ (introduced in Section 6.2), which copied the values for the weights corresponding to the some of the state variables incorrectly, learning in 4 vs. 3 was significantly slower. These results suggest that a ρ which is able to leverage inter-task similarities will outperform more simpleminded ρs.

In this work we have defined χ_X and χ_A so that the state variables and actions are mapped independently. This formulation was sufficient for all the source task and target task pairs considered in this work. However, there are likely tasks where these two mappings are intertwined. For instance, it could be that the actions map differently depending on the agent's current location in state space. We would like to investigate how often such an interdependence would be beneficial, and how our formulation could be enhanced to account for such added complexity.

It is almost certainly possible to find pairs of tasks for which no ρ exists, where transfer would provide no benefit or even hinder learning. It is also possible to think of a pair of tasks for which transferring knowledge should be able to provide a benefit but that an intuitive ρ that allows speedup in learning cannot be found due to the complexity of the domain. This article focuses on providing an existence proof: we show that we are able to construct ρs for the Keepaway domain and that they provide significant benefits to learning. A main goal for our future research is to allow ρ to be constructed automatically between a given pair of tasks, potentially by learning χ_X and χ_A.

9. Related Work

The concept of seeding a learned behavior with some initial simple behavior is not new. The psychological concept of shaping (Skinner, 1953) is well understood and many researchers have applied the idea to machine learning as well as animal training. There have been approaches to simplifying reinforcement learning by manipulating the transition function, the agent's initial state, and/or the reward function, as reviewed in the following paragraphs.

Past research confirms that if two tasks are closely related the learned policy from a source task can be used to provide a good initial policy for a target task. For example, Selfridge et al. (1985) showed that the 1-D pole balancing task could be made harder over time by shortening the length of the pole and increasing its mass; when the learner was first trained on a longer and lighter pole it learned to succeed faster in the harder task with different dynamics and transition function. This method thus changes the transition function T between pairs of tasks but leaves S, the velocity and angle of the pole, and A, move right or move left, unmodified.

Learning from easy missions (Asada et al., 1994) allows a human to change the start state of the learner, $s_{initial}$, making the task incrementally harder. Starting the learner near the exit of a maze and gradually allowing the learner to start further and further from the goal is a demonstration of this. This kind of direction allows the learner to spend less total time learning to perform the final task. Our work differs from these two methods because we allow the modification of S and A between tasks, rather than only T or $s_{initial}$.

Transfer of learning (Singh, 1992) applies specifically to temporally sequential subtasks. Using compositional learning, a large task may be broken down into subtasks that are easier to learn and have distinct beginning and termination conditions. However, the subtasks must all be very similar in that they have the same state spaces, action spaces, and environment dynamics. The reward functions R are allowed to differ. TVITM does not have the restriction that tasks must be divided into subtasks with these characteristics and, again, we have the additional flexibility of changing S and A between the source and target tasks.

Another successful idea, *reward shaping* (Colombetti and Dorigo, 1993; Mataric, 1994), also contrasts with TVITM. In reward shaping, learners are given an artificial problem which will allow the learner to train faster than if they had trained on the actual problem with different environmental rewards, R. However, the policy that is learned is designed to work on the original task as well as the artificial one. TVITM differs in intent in that we aim to transfer behaviors from existing, relevant tasks which can have different state and action spaces, rather than creating artificial problems which are easier for the agent to learn. In the RoboCup soccer domain, all the different Keepaway tasks may occur during the full task of simulated soccer. For instance, there may be times when three teammates on defense must keep the ball from two opponent forwards until another player comes into passing range. We therefore argue that we train on useful tasks rather than just simpler variations of a real task. Using TVITM, learners are able to take previously learned behaviors from related tasks and apply that behavior to harder tasks that can have different state and action spaces.

While these four methods allow the learner to spend less total time training, they rely on modification of the transition function, the initial start state, or the reward function to create artificial problems to train on. We contrast this with TVITM where we allow the state and/or action spaces to change between actual tasks. This added flexibility permits TVITM to be applied to a wider range of domains and tasks than the other aforementioned methods. Furthermore, TVITM does not preclude the modification of the transition function, the start state, or the reward function and can therefore be combined with other methods if desired.

In some problems where subtasks are clearly defined by state features, the subtasks can be automatically identified (Drummond, 2002) and leveraged to increase learning rates. This method is only directly applicable to tasks in which features clearly define subtasks. Furthermore, if the shape of the various regions in the value function are too complex and the smoothness assumption is violated too often, the algorithm to automatically detect subtasks will fail.

Learned subroutines have been successfully transfered in a hierarchical reinforcement learning framework (Andre and Russell, 2002). By analyzing two tasks, subroutines may be identified which can be directly reused in a target task that has a slightly modified state space. The learning rate for the target task can be substantially increased by duplicating the local sub-policy. This work can be thought of as another example in which ρ has been successfully constructed, but in a very different way.

Imitation is another technique which may transfer knowledge from one learner to another (Price and Boutilier, 2003). However, there is the assumption that "the mentor and observer have similar abilities" and thus may not be directly applicable when the number of dimensions of the state space changes or the agents have a qualitatively different action set. Other research (Fern et al., 2004) has shown that it is possible to learn policies for large-scale planning tasks that generalize across different tasks in the same domain. Using this method, researchers are able to speed up learning in different tasks without explicitly transferring any knowledge, as the policy is defined for the planning domain rather than a specific task.

Another related approach (Guestrin et al., 2003) uses linear programming to determine value functions for classes of similar agents. Rather than treating the different agents independently, all agents in the same class use a single value function. The target task is assumed to have similar transition functions and rewards for each class of agent. Thus the authors can directly insert the class-based value subfunctions into agents in the new task, even though there are a different number of objects (and thus different state and action spaces). Although no learning is performed in the new world, the previously learned value functions may still perform better than a baseline hand-coded strategy. However, as the authors themselves state, the technique will not perform well in heterogeneous environments or domains with "strong and constant interactions between many objects (e.g., RoboCup)." Our work is further differentiated as we continue learning in the target task after performing transfer. While the initial performance in the new domain may be increased after loading learned action-value functions compared to learning without transfer, we have found that the primary benefit is an increased learning rate.

The technique of *autonomous shaping* (Konidaris and Barto, 2006) may prove to be useful for transfer learning, particularly in agents that have many sensors. In this work the authors show that reward shaping may be learned on-line by the agent. If a later task has a similar reward structure and actions, the learned reward shaping will help the agent initially have a much higher performance than if it were learning without transfer. For instance, if a signal device (a beacon) is near the goal state, the agent may learn a shaping reward that gives an internal reward for approaching the beacon even if the environmental reward is zero. This work does not directly address how to handle novel actions (specifically, actions which are not in the source task's *agent space*). Additionally, while the authors limit themselves to transfer between "reward-linked" tasks, no method is given for determining if a sequence of tasks are reward-linked and a learned shaping function will not necessarily be useful in a given target task.

Policies from different tasks can also be learned and then used to speed up the current task (Fernandez and Veloso, 2006). This technique relies on having a set of tasks where S, A, and T are constant, but the goal state moves. When the agent is placed in a new task, it can learn to either exploit a past policy, exploit the policy that it is currently learning, or explore. This technique is currently restrictive in that it requires S, A, and T to be unchanged between the set of tasks, that there be one goal state, and that all rewards other than the goal state be zero. However, the idea of building a library of policies may be a critical one for transfer learning. For instance, once an agent has trained in multiple Keepaway tasks it would ideally be able to recall any of these policies if it were placed in the same task or use the most similar of them to speed up the current task.

Automatically generated advice can also be used to speed up learning in transfer (Torrey et al., 2005). This method allows a RL learner to build up a model for the source task and then extract general advice. A human then provides a translation for this advice into the new task, similar to our ρ. The modified advice is then used as constraints in a knowledge-based support vector regression method. Instead of setting the initial Q-values as in our work, the advice sets relative preferences for different actions in different states and thus may work when the reward structure of the tasks change. One apparent shortcoming of this work is that the initial performance and learning rate of the agents is only slightly improved relative to learning without transfer, but the results suggest that advice from one task may be able to help learn a second, related, task. It is also worth noting that while we cannot directly compare the performance of our two methods because we have not implemented the "Breakaway" domain, the relative speedup in learning from TVITM is much greater than that obtained from automatically generating advice.

Wilson et al. (2007) take a different approach by showing that a prior may be transferred in a hierarchical Bayesian reinforcement learning setting. In this work, the authors consider a multitask setting where the goal is to be able to learn quickly on an MDP drawn from a fixed but unknown distribution of MDPs. Learning on subsequent tasks shows a clear performance on a novel task drawn from this distribution but no attempt is made to reduce the total training time.

This work has demonstrated that inter-task mappings can be used to transfer action-value functions. Taylor et al. (2007) demonstrates that the same mappings can be used to transfer policies, learned with a genetic algorithm, between tasks. Rules have also been successfully used (Taylor and Stone, 2007) in conjunction with inter-task mappings as a way of transferring between tasks that used different RL learning methods. Using such a translation mechanism allows, for instance, transfer between a source task policy search learner and a value function target task learner.

There have also been recent advances in learning relationships between related RL tasks, a topic beyond the scope of the current article. Liu and Stone (2006) define *qualitative dynamic Bayes networks* which summarize the effect of actions on different state variables. After networks for the source and target tasks are defined by hand, a graph mapping technique can be used to automatically find inter-task mappings between the two tasks. Taylor et al. (2007) show that it is also possible to learn both χ_A and χ_X by using a classification technique. To enable such a method, the method must be provided *task-independent objects* which describe an object in a task with a set number of state variables. This assumption is also leveraged in other work (Soni and Singh, 2006), but only the state-feature mapping is learned (the action mapping χ_A is hand-coded). *AtEase* (Talvitie and Singh, 2007) is an algorithm that generates a number of possible state-feature mappings and then uses a multi-armed bandit approach to select the best mapping. However, the action mapping (χ_A) is assumed, and learning is not allowed in the target task after an appropriate mapping is selected.

10. Conclusions

This article describes the implementation and results from learning Keepaway with Sarsa, a standard TD method, and three different function approximators. We introduce the transfer via inter-task mapping method for speeding up reinforcement learning and give empirical evidence in the Keepaway domain of its usefulness. Rather than utilizing abstract knowledge, this transfer method is able to leverage the weights from function approximators specifying action-value functions, a very task-specific form of knowledge. We first give formulations of how to define transfer functionals for the different function approximators, or re-use learned weights via Q-value Reuse, from a single pair of inter-task mappings. We proceed to show that agents using all three function approximation methods can learn to reach a target performance faster in the target task. Additionally, we show that the total training time can be reduced using TVITM when compared to simply learning the final task without transfer.

We give further evidence that TVITM is useful for speeding up learning by utilizing the 5 vs. 4 Keepaway task, which suggests that this method will scale up to even more complex problems. We have shown that the TVITM method is robust to some changes in the transition function, such as when the effectiveness of actuators in the two tasks differ. This flexibility may prove critical when transferring behavior between agents situated in the real world, where environmental conditions may cause sensors and actuators to have different behaviors at different times.

We introduce a novel variant of Knight Joust, a gridworld task, and demonstrate that transfer between it and Keepaway is effective despite substantial qualitative differences in the two tasks. We also show how transfer efficacy is reduced when the source task and target task are less related, such as when using 3 vs. 2 Flat Reward or 3 vs. 2 Giveaway as source tasks.

When considered as a whole, the experiments presented in this article establish that TVITM can be used successfully for transferring action-value functions between tasks and reducing training time. This article therefore constitutes a first step towards a fully general and autonomous transfer method within the RL framework.

Acknowledgments

We would like to thank Gregory Kuhlmann for his help with Keepaway experiments described in this article, Cynthia Matuszek and Shimon Whiteson for useful discussions, and the anonymous reviewers for their detailed and constructed comments. This research was supported in part by NSF CAREER award IIS-0237699, NSF award EIA-0303609, and DARPA grant HR0011-04-1-0035.

References

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

David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *Proc. of the Eighteenth National Conference on Artificial Intelligence*, pages 119–125, 2002.

David Andre and Astro Teller. Evolving team Darwin United. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, pages 346–351. Springer Verlag, Berlin, 1999.

Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Vision-based behavior acquisition for a shooting robot by using a reinforcement learning. In *Proc. of IAPR/IEEE Workshop on Visual Behaviors-1994*, pages 112–118, 1994.

Steven J. Bradtke and Michael O. Duff. Reinforcement learning methods for continuous-time Markov decision problems. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 393–400, San Mateo, CA, 1995. Morgan Kaufmann.

Mao Chen, Ehsan Foroughi, Fredrik Heintz, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Patrick Riley, Timo Steffens, Yi Wang, and Xiang Yin. Users manual: RoboCup soccer server manual for soccer server version 7.07 and later, 2003. Available at `http://sourceforge.net/projects/sserver/`.

Marco Colombetti and Marco Dorigo. Robot Shaping: Developing Situated Agents through Learning. Technical Report TR-92-040, International Computer Science Institute, Berkeley, CA, 1993.

Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1017–1023, Cambridge, MA, 1996. MIT Press.

Chris Drummond. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research*, 16:59–104, 2002.

Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Fernando Fernandez and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems*, pages 720–727, 2006.

Mary L. Gick and Keith J. Holyoak. Analogical problem-solving. *Cognitive Psychology*, 12:306–355, 1980.

Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. Generalizing plans to new environments in relational mdps. In *International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, August 2003.

George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 489–496, 2006.

Yaxin Liu and Peter 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.

Maja J. Mataric. Reward functions for accelerated learning. In *International Conference on Machine Learning*, pages 181–189, 1994.

Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka. *Elements of Artificial Neural Networks*. MIT Press, Cambridge, MA, USA, 1997. ISBN 0-262-13328-8.

Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.

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

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994. ISBN 0471619779.

Martin Riedmiller, Author Merke, David Meier, Andreas Hoffman, Alex Sinner, Ortwin Thate, and Ralf Ehrmann. Karlsruhe brainstormers—a reinforcement learning approach to robotic soccer. In Peter Stone, Tucker Balch, and Gerhard Kraetszchmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, pages 367–372. Springer Verlag, Berlin, 2001.

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

Oliver G. Selfridge, Richard S. Sutton, and Andrew G. Barto. Training and tracking in robotics. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 670–672, 1985.

Satinder P. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8:323–339, 1992.

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

Burrhus F. Skinner. *Science and Human Behavior*. Colliler-Macmillian, 1953. ISBN 0029290406.

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

Peter Stone and Richard S. Sutton. Keepaway soccer: a machine learning testbed. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup-2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Artificial Intelligence*, pages 214–223. Springer Verlag, Berlin, 2002.

Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

Peter Stone, Gregory Kuhlmann, Matthew E. Taylor, and Yaxin Liu. Keepaway soccer: From machine learning testbed to benchmark. In Itsuki Noda, Adam Jacoff, Ansgar Bredenfeld, and Yasutake Takahashi, editors, *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020, pages 93–105. Springer Verlag, Berlin, 2006.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998. ISBN 0262193981.

Erik Talvitie and Satinder Singh. An experts algorithm for transfer learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.

Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59, New York, NY, July 2005. ACM Press.

Matthew E. Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, June 2007.

Matthew E. Taylor, Peter Stone, and Yaxin Liu. Value functions for RL-based behavior transfer: A comparative study. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005.

Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2007.

Gerald Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

Lisa Torrey, Trevor Walker, Jude Shavlik, and Richard Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Proceedings of the Sixteenth European Conference on Machine Learning*, 2005.

Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 1015–1022, New York, NY, USA, 2007. ACM Press.