In *The AAMAS workshop on Adaptive and Learning Agents* (ALA-10), Toronto, Canada, May 2010.

Transfer Learning for Reinforcement Learning on a Physical Robot

Samuel Barrett
Dept. of Computer Science
University of Texas at Austin
Austin, TX 78712 USA
sbarrett@cs.utexas.edu

Matthew E. Taylor
Dept. of Computer Science
University of Southern California
Los Angeles, CA 90089
taylorm@usc.edu

Peter Stone
Dept. of Computer Science
University of Texas at Austin
Austin, TX 78712 USA
pstone@cs.utexas.edu

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Experimentation

Keywords

Transfer Learning, Robotics, Reinforcement Learning, Artificial Intelligence

ABSTRACT

As robots become more widely available, many capabilities that were once only practical to develop and test in simulation are becoming feasible on real, physically grounded, robots. This newfound feasibility is important because simulators rarely represent the world with sufficient fidelity that developed behaviors will work as desired in the real world. However, development and testing on robots remains difficult and time consuming, so it is desirable to minimize the number of trials needed when developing robot behaviors.

This paper focuses on reinforcement learning (RL) on physically grounded robots. A few noteworthy exceptions notwithstanding, RL has typically been done purely in simulation, or, at best, initially in simulation with the eventual learned behaviors run on a real robot. However, some recent RL methods exhibit sufficiently low sample complexity to enable learning entirely on robots. One such method is transfer learning for RL. The main contribution of this paper is the first empirical demonstration that transfer learning can significantly speed up and even improve asymptotic performance of RL done entirely on a physical robot. In addition, we show that transferring information learned in simulation can bolster additional learning on the robot.

1. INTRODUCTION

Physically grounded robots need to be able to learn from their experience, both in order to deal with changing environments and to adapt to new problems. For the purpose of online learning of sequential decision making tasks with limited feedback, value-function-based reinforcement learning (RL) [15] is an appealing paradigm, because of the well-defined semantics of the value function and its elegant theoretical properties. However, a few notable successes notwithstanding (e.g., flying RC helicopters [3, 9] and quadruped walking [8]), RL algorithms have typically been applied only in simulation, or at best trained in simulation with the eventual learned behaviors run on a real robot (e.g., [6], [10], and [5]).

Learning on physically grounded robots is difficult for several reasons, including environmental and sensor noise, high costs of failure (such as a crashed helicopter), the large amount of time it takes to perform tasks, and the fact the robots' dynamics are often not constant due to wear and tear on their motors. Thus, to the extent possible, it is desirable to train robots in a controlled environment before sending them out into the world. Doing so can reduce damage to the robots and prepare them to deal with *expected* situations. However, when encountering *unexpected* situations after "deployment" in the real world, the robot will have to continue to adapt. Such unexpected situations can even arise from the dynamics of the robot itself changing as its joints break, or as repairs are made. It is *conceivable* to relearn tasks from scratch each time a change happens, but due to the time and cost of learning, it is not *practical*. Instead, it is desirable for the robot to reuse prior information in order to learn faster. The concept of reusing information from past learning is the idea behind *transfer learning*.

Transfer learning for RL tasks has been shown to be effective in simulation [18], but no prior work has been done on transfer learning on physically grounded robots. The main contribution of this paper is the first empirical demonstration that transfer learning for RL can significantly speed up and even improve asymptotic performance of RL with learning done entirely on a physical robot, specifically using Q-value reuse for the Sarsa(λ) algorithm [19]. In addition, we show that transferring information from learning in simulation can improve subsequent learning learning on the robot.

To this end, we introduce a novel reinforcement learning task for humanoid robots and demonstrate that transfer learning can be effective for this task. The results additionally represent one of the first successful applications of reinforcement learning on the Nao humanoid platform developed by Aldebaran[1]. A limited amount of previous work has been done using the Nao, but this work focused on simulation work, with only a single run on a physical robot [7].

The remainder of the paper is organized as follows. Section 2 presents the main algorithms used in our experiments, namely Sarsa(λ) and Q-value reuse. Section 3 introduces our experimental testbed and fully specifies the task to be learned. Sections 4 and 5 present the results of our experiments. Section 6 further situates the results in the literature, and Section 7 concludes.

2. BACKGROUND

Reinforcement learning (RL) is a framework for learning sequential decisions with delayed rewards [15]. RL is promising for robotics because it handles online learning with limited feedback where actions taken affect the environment. RL has been extensively studied in many domains, with positive results. However,

[1]http://www.aldebaran-robotics.com/eng

RL techniques can require long training times. Therefore, especially on robots, it can be useful to reuse knowledge learned from similar problems to speed up training times via transfer learning.

Value-function-based RL algorithms assume that the task to be learned can be modeled as a Markov Decision Process (MDP). An MDP is a four-tuple of (S, A, T, R) where S is a state space, A is an action space, T is a transition function specifying the effects of actions, and R is an immediate reward function specifying the value of state transitions. The complete formulation is given by $T(s, a) = s'$ with $s, s' \in S$ and $a \in A$ and $R(s, s') = r$ where $r \in \mathbb{R}$. However, the agent typically does not start with any knowledge about T or R, so it must learn what actions should be taken in the states it encounters. One way of doing so is via an intermediate data structure called a *state-action value function* (Q) that stores the expected long-term reward of executing action a from state s.

Taylor et al. [19] recently demonstrated that state-action values from one RL problem can be effectively reused in a related, but different sequential decision making problem. This result is surprising because the state-action values are intuitively the most problem-specific data structure of an RL algorithm: they represent the expected long-term reward from a given state-action pair in a single, specific problem. However, it turns out that there are useful patterns encoded in the state-action value function that can speed up and even improve asymptotic learning on related tasks. Their algorithm of Q-value reuse, which we adopt in this paper, is based on the standard RL framework.

2.1 Sarsa(λ)

This research uses the Sarsa(λ) learning algorithm as the base RL algorithm. We choose to use Sarsa because it is compatible with Q-value reuse, and because it is among the simplest of RL algorithms: our focus is on speed-up due to transfer rather than on the learning algorithm itself.

Sarsa is an on-policy temporal difference learning algorithm first proposed by Rummery and Niranjan [11] and later augmented by Sutton [14]. Specifically, Sutton's work describes using cerebellar model arithmetic computers (CMACs) [2] as a function approximator for generalizing learning, allowing the agent to generalize across similar states and handle larger (or infinite) state spaces. This approach has been shown to be successful in a number of domains.

The Sarsa(λ) algorithm is a well-known approach to solving an MDP. It learns a value function over state-action pairs, $Q(s, a) = r$, and actions are chosen ϵ-greedily with respect to Q. The value function is changed via a Bellman (TD) update:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha[R(s, s') + \gamma e(s, a)Q(s', a')]$$

given the state s, action a, reward r, next state s', next action a', the discount factor γ, the eligibility trace $e(s, a)$ (representing how recently the state-action pair was visited), and the decay parameter λ.

We use CMACs for their discretization and generalization abilities, deriving from their infinite, axis-parallel tilings over a continuous state space. These tiles are discrete features, and there are a constant number active for each point in the space. Several tilings are used and each is offset from the others (by a random amount in our implementation). The value function is generalized over each tile, but the overlapping tiles allow for better resolution. A CMAC's value for each action is computed by summing the weights from each activated tile:

$$f(x) = \sum_i w_i f_i(x)$$

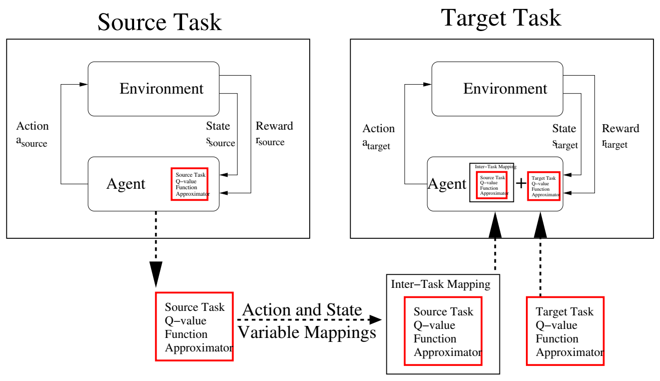

Source Task
Target Task
Environment
Action
a_{source}
State
s_{source}
Reward
r_{source}
Agent
Source Task Q–value Function Approximator
Environment
Action
a_{target}
State
s_{target}
Reward
r_{target}
Agent
Inter–Task Mapping
Source Task Q–value Function Approximator
+
Target Task Q–value Function Approximator
Source Task Q–value Function Approximator
Action and State Variable Mappings
Inter–Task Mapping
Source Task Q–value Function Approximator
Target Task Q–value Function Approximator

Figure 1: Q-Value Reuse

where

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

For any state, the result is a vector of values with length equal to the number of actions, and the lengths may not be the same for different tasks.

2.2 Q-value Reuse

Transfer learning involves reusing knowledge learned from earlier tasks to learn new problems more effectively. The task learned previously is called the *source task* and the new task is called the *target task*. We use Q-value reuse for the transfer, where the value function, Q_{source}, learned from an earlier task is used as a starting point for the new problem, and a new value function, Q_{target}, is learned to correct errors in the source value function. However, the source state and action spaces may not coincide with the target state and action spaces. Therefore, the agent must be given a mapping between the source and target tasks: $\chi_X(s_{\text{target}}) = s_{\text{source}}$ and $\chi_A(a_{\text{target}}) = a_{\text{source}}$. In this work, this mapping is provided to the agent rather than learned. Therefore, the agent's new value function is given by

$$Q(s, a) = Q_{\text{source}}(\chi_X(s), \chi_A(a)) + Q_{\text{target}}(s, a)$$

Figure 1 shows how Q-value reuse works, reusing the source task's state-action value function approximator in the target.

Sarsa updates are calculated the same way as previously, but only the target's function approximator weights ($Q_{\text{target}(s,a)}$) are updated. In some cases, there may be no corresponding action in the source task so a default value is given to these actions. In this paper, we initialize these actions to the average action values across all possible states in the source domain [19]. We also tried initializing new actions to an intermediate value picked by hand (0) and to the average action value for the current state, but the average action values of all states performed better in initial experiments.

3. EXPERIMENTAL SETUP

For all experiments in this paper, we use a novel task on the Nao humanoid platform developed by Aldebaran. We chose a task with the robot seated to reduce the possibility of damaging the robot and for easier control of the robot's start state. We emphasize the physical groundedness of our experiments by requiring that the robot calculate its own reward signal from observations, accepting any resulting inaccuracies.

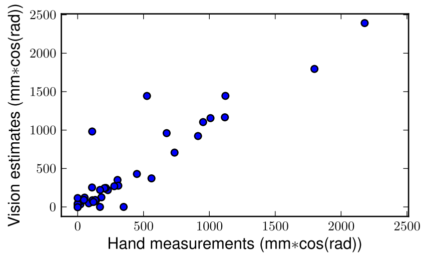


Figure 2: Estimates of episode rewards

Specifically, the robot's task was to hit an orange ball as far as possible at a 45° angle with its right hand. It used its onboard camera to observe the result of each trial and calculate the reward signal. The robot is seated with the ball 80 mm in front of the center of the robot and 170 mm to its right. Note that the robot is not given ball's location except for the information in the reward signal. Every 75 ms, the robot is given the current positions of the joints and their velocities as observations.

The reward signal is given by $r = d * \cos(\theta)$, where d is the distance that the ball moved, and θ is the angle between the ball's trajectory and the 45° target angle. If the ball was not seen for sufficiently long, it was assumed to have been hit backwards, and the action was assigned reward -100. All other steps were given a reward of -1 to encourage the agent to find a fast action sequence to hit the ball.

The reward from vision can be inaccurate, due to the ball moving outside the sight range of the robot, the arm obscuring the sight of the ball, and noisy distance estimates of the ball. However, we measured these effects, and found that they were not very significant. Figure 2 compares the robot's estimate of the reward with the measurements taken by hand using a tape measure and a protractor. Out of 50 episodes, only two successful hits were not seen by the robot and incorrectly assumed to be backward hits with reward -100. The R-squared value of the robot's estimations was 0.86.

As shown in Figure 3 (supplied by Aldebaran[1]), the robot can use four joints to help it hit the ball: shoulder pitch, shoulder roll, elbow pitch, and elbow yaw. For each episode, these joints start at a fixed position with no initial velocity; these values are given in Table 1 and depicted in the left-most frames of Figure 4. Also, the ball starts in the same position for every episode, as shown in Figure 4. At each time step, the robot can accelerate one joint in either direction or leave all the joints alone. Therefore, the robot has nine actions: {no change, accelerate the shoulder pitch upwards, accelerate the shoulder pitch downwards, accelerate the shoulder roll clockwise, or accelerate the shoulder roll counter-clockwise, etc.}. Furthermore, it has eight observations: the position and velocity for each joint. The velocities are kept in the range $[-100^\circ/s, +100^\circ/s]$ and the actions are taken every 75ms (more than 13 times per second) to change the velocity by $50^\circ/s$.

It is possible to learn this task without any prior information, but the process can be slow and the robot converges to a mediocre policy. Our work focuses on improving this learning, specifically by using a related source task as prior information. For this simpler task, the robot only has control of the two shoulder joints, with the elbow roll and yaw fixed at 0° and 0°. Therefore, the robot will only have five actions and four observations. We will refer to this simpler task as the source task and the original task as the target task. The keyframes of the robot performing the two tasks can be

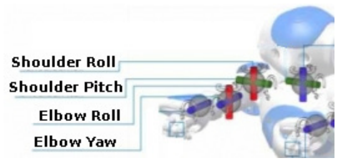
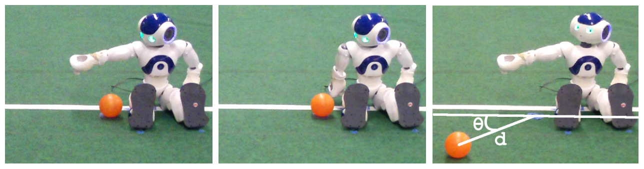
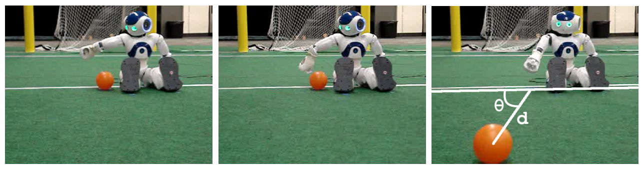


Figure 3: Joint movements possible for the task



(a) Source task



(b) Target task

Figure 4: Keyframes of robot tasks

seen in Figure 4. The robot has less control in this source task, and therefore cannot hit the ball as far as in the target task, but it can learn faster as the problem is simpler. Our central hypothesis is that using Q-value reuse to transfer information from this source task will enable the robot to learn faster on the target task.

As this work focuses on transfer learning on robots, so the main task considered was transferring from the source task to the target task on the robot, compared to learning the target task with no prior information. We also replicated both tasks in the Webots simulator[2] to test our algorithm in a different, though similar environment (as the dynamics of the simulator do not entirely match the physical robot). We do not assume that a useful simulator will be available in all cases, which is why we focus on transfer on the robot itself. In this case, the simulator allows us to better evaluate the effectiveness and robustness of the algorithm and to run many more experiments than physical robots allow. However, we emphasize that for the main result of the paper, both the source and target tasks were learned on the physical robot.

We refer to the source task on the robot as SOURCEROB, the target task on the robot as TARGETROB, the source task in the simulator as SOURCESIM, and the target task in the simulator as TARGETSIM. The main test of our algorithm is in how the trans-

[2]Cyberbotics Ltd. http://www.cyberbotics.com

Joint	Min	Max	Start
Shoulder pitch	0°	115°	115°
Shoulder roll	-90°	5°	-75°
Elbow roll	0°	120°	45°
Elbow yaw	-90°	90°	-45°

Table 1: Joint angle ranges and starting positions

fer from SourceRob to TargetRob and from SourceSim to TargetSim performs. However, the use of the simulator allows for several other paths for transfer information, and we discuss this idea further in Section 5

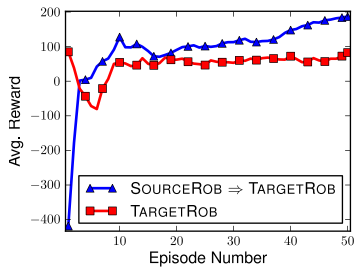
A significant part of the work was done using the Webots simulator[2], and this work relies heavily on the code developed by the UT Austin Villa robot soccer team[3]. This code base provides the interface between the learning agent and the robot's actions, as well as providing visual detection of the ball.

4. RESULTS

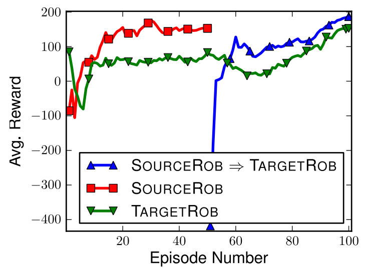
Transfer learning can be evaluated in many different ways [18]. In this paper, our main focus is on "weak" transfer, meaning that we assume that the time spent in the source task does not count against the learner in the target. This is the case when the robot has already learned the source task, so this training time is not a new cost. For example, if a robot was trained in a lab before being sent out, we might be interested in the time it would take the robot to learn a new task, and less interested in how long the robot was trained in the lab. We also show one "strong" transfer result, where time spent in the source does count.

For all experiments, we plot the running average reward for each approach, taken with a 25 episode moving window for the robot tests and a 50 episode window for the simulation tests. Each test on the robot represents five runs, each lasting 50 episodes. In the simulator, each test averages 50 runs, each lasting 1,000 episodes. These 50 episodes on a robot takes approximately 30 minutes, and 1,000 episodes in simulation takes approximately three hours. This data allows us to draw conclusions with statistical significance and reason about the convergence of each approach.

The baseline that we use is learning TargetRob with no prior information. Figure 5a shows that transfer from SourceRob to TargetRob is helpful, improving the reward throughout the entire test. The initial few episodes of each algorithm are very noisy, so the initial positive performance of TargetRob is not significant, just the effect of a few outliers. This graph is an evaluation of weak transfer: we do not depict training time in the source task.



(a) Weak Transfer

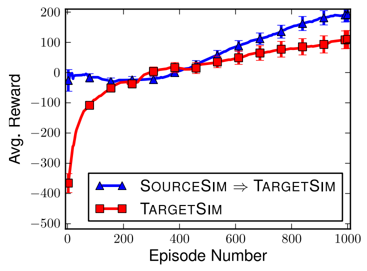


(b) Strong Transfer

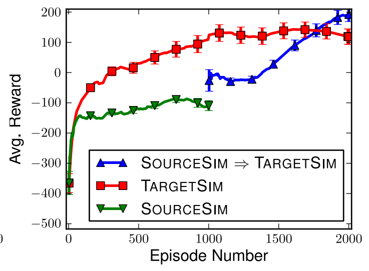
Figure 5: Transfer on the robot to the target task

Figure 5b shifts the transfer plot 50 episodes to the right to represent the strong transfer scenario. Though not as dramatic, the result is still positive, thus demonstrating that it can be useful to break a robot task into robot subtasks, and then transfer from the subtasks to the target task. In this test, the robot performs about as well in the source task as in the target task, because it does not have enough trials to completely explore the target task and find a good behavior.

Unfortunately, the small number of tests on the robots means that we cannot draw statistical conclusions about the performance of the methods. However, the tests were also replicated in simulation with good results. Figure 6a shows that the transfer from SourceSim to TargetSim is helpful, even after a large number of episodes. The differences between the final rewards of each method are statistically significant with a confidence of 99%, and the error bars in the diagram show the standard deviation of the average rewards. Figure 6b shows that our results for strong transfer hold in simulation. Overall, Figures 5–6 suggest that transfer learning works on robots, and can greatly speed up learning and reach better end behaviors.



(a) Weak Transfer



(b) Strong Transfer

Figure 6: Transfer in the simulator

5. ADDITIONAL EXPERIMENTS

In addition to providing statistically significant results, the use of the simulator opens several other paths for transferring knowledge

[3]http://www.cs.utexas.edu/users/austinvilla

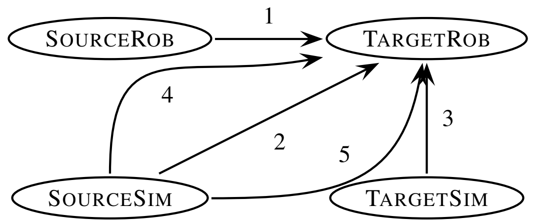


Figure 7: Paths for transferring experience

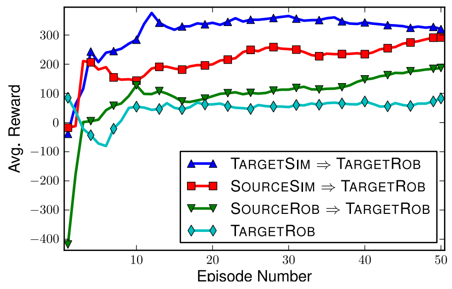


Figure 8: One-step transfer to the robot target task

between tasks, including two-step transfer, where we learn sequentially from multiple source tasks. Two-step transfer is performed as described in Section 2, with the value function:

$$\begin{aligned} Q(s,a) &= Q_1(\chi_{X_1}(s), \chi_{A_1}(a)) \\ &\quad + Q_2(\chi_{X_2}(s), \chi_{A_2}(a)) + Q_3(s,a) \end{aligned}$$

We consider TARGETROB to be the target for all of the tests, and we continue using 1,000 episodes in simulation and 50 on the physical robot. Figure 7 shows all the ways to transfer information to TARGETROB, with numbers corresponding to the following tests:

1. SOURCEROB → TARGETROB
2. SOURCESIM → TARGETROB
3. TARGETSIM → TARGETROB
4. SOURCESIM → SOURCEROB → TARGETROB
5. SOURCESIM → TARGETSIM → TARGETROB

Test 1 is further investigated in Section 4, and the results of the three one-step transfer tests (tests 1, 2, and 3) are displayed in Figure 8. Transferring from TARGETSIM produces the biggest improvement in the early episodes due to it having already learned about the entire state-action space. However, the agent does have to learn about the differences between the simulated and real robots. Also, transferring from SOURCESIM performs better than transferring from SOURCEROB, probably due to the higher number of runs in SOURCESIM, which allow the agent to explore the state-action space more completely. In the end, all of the transfer methods end up with similar performance, and all perform much better than starting with no prior information.

The two types of two-step transfer were also tested (tests 4 and 5), and the results are shown in Figure 9. Both methods show a substantial boost to early episodes but later plateau, achieving similar results to the other transfer methods. The results of the two-step transfer are not better than some of the one-step transfers, but Figure 10 shows that multi-step transfer can be beneficial, giving a large early boost.

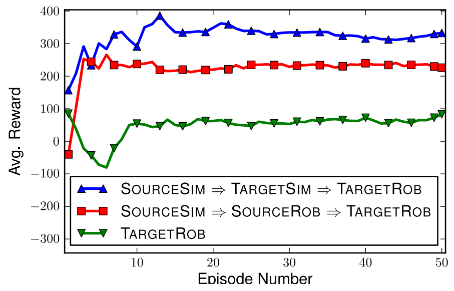


Figure 9: Two-step transfer to the robot target task

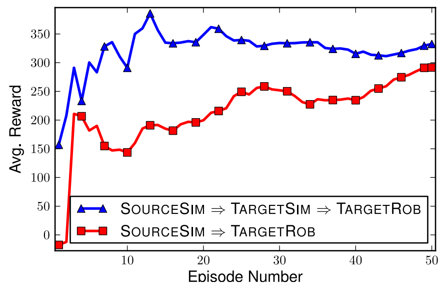


Figure 10: Comparison of one and two-step transfer.

Though all of these results are for weak transfer, we speculate that these trends will hold for strong transfer (as they did in both one-step transfer cases). Furthermore, transferring from simulation to a physical robot raises the possibility of having different costs for training spent in the simulator than on the robot. For example, if we consider simulation time to be insignificant, then tests 2, 3, and 5 are all evidence of strong transfer.

6. RELATED WORK

One of the earliest uses of transfer learning for reinforcement learning was done by Selfridge et al. [12] in the familiar cart-pole domain. In this work, the function approximator was reused for poles of different sizes and weights, with good effect.

Taylor and Stone [18] recently surveyed the use of transfer learning in reinforcement learning. Significant prior work in this area has been performed, with good results. However, little work has been done in applying transfer learning to the area of robotics. Taylor and Stone discuss several approaches to transfer learning, and point out several ways to evaluate the effects of the transfer. Our research focuses on Q-value reuse with supervised task transfer.

Taylor et al. [19] explored Q-value reuse in temporal difference learning with good results. They specifically evaluate a Sarsa agent using CMAC for function approximation. However, this work focuses on the simulated domain of keepaway for soccer. Our work applies this research to a physical robot, and has a greater difference between the source and target tasks.

One interesting approach to transfer learning is to extract higher level strategies from the policy learned by the agent. Torrey et al. [20] explored this idea using relational macros to represent the strategies learned by induction logic programming (ILP) in the RoboCup breakaway domain, but this requires the domain to be translated into first-order logic. It is also possible to break a single problem into a series of smaller tasks. Then, the agent learns each of these sub-tasks and combines the learned knowledge for the full task.

In the target task, the state space, actions, and transition function are the same as the sub-tasks, and the information is transferred via Q-value transfer. Singh [13] also explored this area, naming it "compositional learning."

It is possible to learn a mapping between source and target tasks autonomously (e.g., when a human is unable or unwilling to provide such a mapping). Talvitie and Singh [16] developed an algorithm to generate possible state variable mappings and learn which mapping is best as an n-armed bandit problem. Further work has been done by Taylor et al. [17] using a model-based approach to reduce the samples needed, and they transfer observed (s, a, r, s') instances, which allows the source and target agents to have different representations for the task. However, these methods are not as reliable as hand-mapping and can be unnecessary for smaller domains.

Unfortunately, tests on robots can be slow, and most learning algorithms require a large amount of training data to perform well. Therefore, it can be useful to train an agent in simulation and transfer these behaviors to a robot [6, 10, 5]. However, we cannot assume that a simulator will accurately model complex perception or manipulation tasks, so it is often useful to tune the behavior from the simulator by running more tests on a robot. This requires combining information about a source simulation task and a target robot task, but no work we know of treats this as a transfer learning problem.

Another way to speed up learning is to use prior demonstrations. Researchers have shown that sub-optimal demonstrations can be sufficient to teach an agent to control an autonomous helicopter [1, 4]. Unfortunately, this requires on an expert in the domain to perform the demonstrations, which is not always possible.

7. CONCLUSIONS AND FUTURE WORK

This paper empirically tests transfer learning for RL on physical robots. The results show that model-free RL can be effective on a robot, and that transfer learning can speed up learning on physical robots.

Furthermore, this prior information can be learned in simulation, even if the simulator does not completely capture the dynamics of the robot. For example, the simulator does not model collisions between the robot's different parts, so dynamics of the arm hitting the body are never learned in the simulator. However, the behaviors learned in the simulator serve as good starting points for learning on the robot. This result is useful when a simulator is available, since simulator tests are significantly easier to run than robot tests: it suggests that only a relatively small amount of tuning is necessary to adapt behaviors learned in the simulator to the real robot. The main motivation for this work is that in some situations learning must be performed entirely on a physical platform, and the positive results in that setting are the main contribution of this paper.

This work opens up several interesting directions for future work. For example, it is worth investigating if other learning algorithms can learn this task faster than Sarsa, and if so, whether Q-value reuse (if applicable) can show similar benefits with these other algorithms. It would also be interesting to see how different methods for transfer learning perform on this task. In the long run, we view the research reported in this paper as just the first of many possible applications of transfer learning for RL to physical robots.

8. REFERENCES

[1] P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *ICML '05*, pages 1–8. ACM, 2005.

[2] J. S. Albus. *Brains, Behavior, and Robotics*. Byte Books, Peterborough, NH, 1981.

[3] J. A. Bagnell and J. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *ICRA '01*, pages 1615–1620. IEEE Press, 2001.

[4] A. Coates, P. Abbeel, and A. Y. Ng. Learning for control from multiple demonstrations. In *ICML '08*, pages 144–151. ACM, 2008.

[5] Y. Davidor. *Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization*. World Scientific Publishing Co., Inc., 1991.

[6] E. Gat. On the role of simulation in the study of autonomous mobile robots. In *AAAI-95 Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents.*, March 1995.

[7] T. Hester, M. Quinlan, and P. Stone. Generalized model learning for reinforcement learning on a humanoid robot. In *ICRA '10*, 2010.

[8] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *ICRA '04*, May 2004.

[9] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Inverted autonomous helicopter flight via reinforcement learning. In *In International Symposium on Experimental Robotics*. MIT Press, 2004.

[10] J. M. Porta and E. Celaya. Efficient gait generation using reinforcement learning. In *Proceedings of the Fourth International Conference on Climbing and Walking Robots*, pages 411–418, 2001.

[11] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUEF/F-INFENG/TR 166, Cambridge University Engineering Dept., 1994.

[12] O. G. Selfridge, R. S. Sutton, and A. G. Barto. Training and tracking in robotics. In *IJCAI*, pages 670–672, 1985.

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

[14] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *NIPS '96*, 1996.

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[16] E. Talvitie and S. Singh. An experts algorithm for transfer learning. In *IJCAI*, pages 1065–1070, 2007.

[17] M. E. Taylor, N. K. Jong, and P. Stone. Transferring instances for model-based reinforcement learning. In *ECML PKDD*, pages 488–505, September 2008.

[18] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *JMLR*, 10(1):1633–1685, 2009.

[19] M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *JMLR*, 8(1):2125–2167, 2007.

[20] L. Torrey, J. W. Shavlik, T. Walker, and R. Maclin. Relational macros for transfer in reinforcement learning. In *ILP '07*, 2007.