To appear in *Proceedings of the Twenty-Third International Conference on Machine Learning* (ICML 06), Pittsburgh, PA, June 2006.

Transfer Learning for Policy Search Methods

Matthew E. Taylor MTAYLOR@CS.UTEXAS.EDU
Shimon Whiteson SHIMON@CS.UTEXAS.EDU
Peter Stone PSTONE@CS.UTEXAS.EDU
Department of Computer Sciences, University of Texas at Austin, 1 University Station, C0500, Austin, TX 78712-0233

Abstract

An ambitious goal of *transfer learning* is to learn a task faster after training on a different, but related, task. In this paper we extend a previously successful *temporal difference* (Sutton & Barto, 1998) approach to transfer in *reinforcement learning* (Sutton & Barto, 1998) tasks to work with policy search. In particular, we show how to construct a mapping to translate a population of policies trained via genetic algorithms (GAs) (Goldberg, 1989) from a *source* task to a *target* task. Empirical results in robot soccer Keepaway, a standard RL benchmark domain (Stone et al., 2006), demonstrate that *transfer via inter-task mapping* can markedly reduce the time required to learn a second, more complex, task.

1. Introduction

Reinforcement learning (RL) (Sutton & Barto, 1998) problems are characterized by agents making sequential decisions with the goal of maximizing total (possibly delayed) reward. Recent work has focused on speeding up learning across tasks with different state and action spaces via *transfer learning*. Transfer learning allows agents to first learn an initial *source task* and then in a second, typically more complex, *target task*. Transfer learning is successful if the target task can be learned faster (or with better final performance) after using knowledge learned in the source task than if learners train only on the target task.

One approach to transfer learning is to construct a translation functional that maps the final value function learned in the source task to an initial value function in the target task (Taylor et al., 2005). This method, which works even when the two tasks have different state and action spaces, effectively transferred knowledge learned with *temporal difference* (TD) methods (Sutton & Barto, 1998). While it has succeeded with several different kinds of function approximators, it has not been shown to work with RL methods that do not learn value functions.

In this paper, we extend the approach of *transfer via inter-task mapping* and show how to construct a mapping to translate a set of policies (a *population* of *organisms*) trained via a *genetic algorithm* (GA) (Goldberg, 1989) on a source task to form the initial population for training on a target task.

We evaluate this method in robot soccer Keepaway, a standard RL benchmark domain (Stone et al., 2006). Results demonstrate that transfer via inter-task mapping reduces the time a GA takes to learn a target task. Furthermore, the total training time of both source and target tasks is less than the training time needed to learn the target task from scratch.

Appearing in *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, Pittsburgh, PA, 2006. Copyright 2006 by the author(s)/owner(s).

2. NEAT

GAs are search and optimization methods with significant empirical success evolving policies to solve RL tasks. This paper uses NeuroEvolution of Augmenting Topologies (NEAT) (Stanley & Miikkulainen, 2002) as a representative GA. NEAT, which trains populations of neural networks, is an appropriate choice because of past empirical successes on difficult RL tasks such as pole balancing (Stanley & Miikkulainen, 2002), robot control (Stanley & Miikkulainen, 2004), and Keepaway (Taylor et al., 2006). Additionally, unlike many other optimization techniques, NEAT automatically learns appropriate representations for the solution, often discovering small networks that learn relatively quickly.

Since NEAT is a general purpose optimization technique, it can be applied to a wide variety of problems. When used for policy search in RL problems, NEAT typically evolves *action selectors*, which directly map states (input nodes) to the action (an output node) the agent should take in that state. The agent performs the action whose corresponding output has the highest activation.

3. MDP Terminology

Transfer via inter-task mapping relies on leveraging relationships between pairs of RL tasks. To define such a relationship, we use standard notation for Markov decision processes (MDPs) (Puterman, 1994). An agent's knowledge of the state of its environment, $s \in S$ is a vector of k *state features*, so that $s = x_1, x_2, \ldots, x_k$. The agent has a set of actions, A, from which to choose. A reward function, $R : s \mapsto \mathbb{R}$, defines the instantaneous environmental reward of a state. A policy $\pi : S \mapsto A$ defines how an agent interacts with the environment. The success of an agent's policy is defined by how well it maximizes the total reward it receives in the long run while following that policy. The action selectors evolved by NEAT thus have k inputs, one for each state feature, and $|A|$ outputs, one for each action.

4. Constructing an Inter-Task Mapping

To perform transfer via inter-task mapping with NEAT, we need a way to convert the population[1] of networks trained on the source task into a population of networks suitable

[1]Transferring a population, rather than a single policy, allows search to begin in the target task from a variety of locations in policy space, which increases the chances of finding a good starting point for learning. Informal results showed transferring a population was more effective than transferring a few of the best policies.

for training on the target task. However, we cannot simply transfer the policies unaltered because, in the general case, the state and actions spaces may be different for the two tasks. In this section we define a translation mapping ρ such that $\rho(\pi_{source}) \mapsto \pi_{target}$ to properly perform this transfer.

Given an arbitrary pair of unknown tasks, one could not hope to correctly define ρ, the inter-task mapping. For transfer to succeed, not only must the two tasks be related, but the human designer must understand *how* they are related. Hence, when constructing ρ we assume that a human familiar with the two tasks has provided two mappings, γ and β. γ maps each state feature in the target task to the most similar state feature in the source task: $\gamma(x_{i,source}) = x_{j,target}$. Similarly, β maps each action in the target task to the most similar action in the source tasks: $\beta(a_{i,source}) = a_{j,target}$. In this paper we will be applying γ and β to neural network action selectors and thus we substitute network input nodes for the state features and output nodes for actions. Note that ρ is a mapping from the source task to the target task, while β and γ are mappings from the target task to the source task.

Given γ, β, and a network π_{source} trained by NEAT, we can create a new network π_{target} using the following procedure. π_{target} begins with no links but has one input for each state feature in the target task, one output for each action in the target task, and one hidden node for each such node in π_{source}. If a function δ represents the correspondence between these hidden nodes ($\delta(h_{target}) = h_{source}$), then each node $n \in \pi_{target}$ can be mapped back to a node in π_{source} via ψ:

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

By using ψ, we can now add links to π_{target} by copying the links that connect the corresponding nodes in π_{source}. For every pair of nodes n_i, n_j, in π_{target}, if a link exists between $\psi(n_i)$ and $\psi(n_j)$ in π_{source}, a new link with the same weight is created between n_i and n_j. By applying this method to all policies in the source population, we can initialize a population of policies in the target task. This allows all networks in the target task to be given structure and link weights learned from the source task; this knowledge biases the population in policy space so that NEAT can learn faster in the target task than when learning from scratch. Algorithm 1 summarizes this domain independent process.

Algorithm 1 APPLICATION OF ρ

```
1: for each network $\pi_{source} \in population_{source}$ do
2:   Construct a network $\pi_{target} \in population_{target}$ where #
     of input and output nodes are determined by the target task.
3:   Add the same number of hidden nodes to $\pi_{target}$ as
     $\pi_{source}$.
4:   for each pair of nodes $n_i, n_j \in \pi_{target}$ do
5:     if link($\psi(n_i), \psi(n_j)$) $\in \pi_{source}$ then
6:       Add link($n_i, n_j$) to $\pi_{target}$ with weight identical to
         link($\psi(n_i), \psi(n_j)$)
```

5. Testbed Domain: Keepaway

To test the efficacy of transfer via inter-task mapping we consider the RoboCup simulated soccer Keepaway domain using a setup similar to past research (Stone et al., 2005; Taylor et al., 2006). The agents on one team – the *keepers* – choose from a set of macro-actions so as to maintain control of the ball. Macro-actions can last more than one time step and agents make decisions only when a macro-action terminates. The macro-actions are Hold Ball, Get Open, Receive, and Pass (Stone et al., 2005). The opposite team – the *takers* – do not learn and follow a static hand-coded policy while attempting to steal the ball.

As more players are added to the task, Keepaway becomes harder for the keepers because the field is more crowded and the average pass distance is shorter, forcing more errors due to noisy sensors and actuators. As more takers are added there are more players to block passing lanes and chase down any errant passes. For these reasons, keepers in 4 vs. 3 Keepaway (i.e. 4 keepers and 3 takers) take longer to learn an optimal control policy than in 3 vs. 2 and the best policies in 4 vs. 3 have lower performance than the best 3 vs. 2 policies. The addition of an extra taker and keeper to the 3 vs. 2 task also results in a qualitative change: in 4 vs. 3 a third taker is now free to roam the field and attempt to intercept passes. See our past work (Taylor et al., 2006) for further details.

6. Learning Keepaway with NEAT

Our Keepaway players are based on version 0.6 of the benchmark players distributed by UT-Austin[2] (Stone et al., 2006). 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, in 3 vs. 2 Keepaway, a keeper with the ball may choose from 3 actions, A = {hold, passToTeammate1, passToTeammate2}.

The keepers' states comprise distances and angles of the keepers $K_1 - K_n$, the takers $T_1 - T_m$, and the center of the playing region, C. Keepers and takers are ordered by increasing distance from the ball and states are rotationally invariant. Note that as the number of keepers *n* and the number of takers *m* increase, the number of state features also increases so that the more complex state can be fully described. S must 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. Full details of the Keepaway domain are documented elsewhere (Stone et al., 2005).

When playing 3 vs. 2 Keepaway, keepers' states are defined by 13 features, comprised of distances and angles to other players. Keepers receive a reward of +1 for every time step the ball remains in play. The episode finishes when a taker gains control of the ball or the ball is kicked out of bounds. We used NEAT to evolve teams of homogeneous agents: in any given episode, copies of the same neural network is used to control all three autonomous keepers on the field.

4 vs. 3 Keepaway has the same size field but an additional keeper and taker. Hence, A = {hold, passToTeammate1, passToTeammate2, passToTeammate3} and S is composed

[2]Flash file demonstrations, source code, documentation, and mailing list are located at *http://www.cs.utexas.edu/users/AustinVilla/sim/keepaway/*.

Partial Description of γ

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

Table 1. This table describes seven example correspondences between state features in Keepaway. We denote the distance between a and b as $dist(a, b)$. Relevant points are the center of the field C, keepers K_1-K_4, and takers T_1-T_3. Keepers and takers are ordered in increasing distance from the ball and state values not present in 3 vs. 2 are bold.

of 19 state features due to the added players. Every network needs to have 19 inputs, 1 bias input, and 4 outputs.

7. Transfer via Inter-Task Mapping in Keepaway

In this section we define the mappings γ and β, used for transferring between 3 vs. 2 and 4 vs. 3 Keepaway. A and S change when the number of players is increased but we are able to easily[3] define these mappings between states and actions to transfer knowledge between the two tasks. The definitions of these mappings is the same as used by our past transfer work in this domain (Taylor et al., 2005).

We define β, the 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, i.e. 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 feature mapping, γ, is handled with a similar strategy. Each of the 19 state features in the 4 vs. 3 task is mapped to a similar state feature 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 source task is similar to "Distance to second closest keeper" in the target task, and "Distance to third closest keeper" in the target task. See Table 1 for more examples of state feature mappings.

8. Results and Discussion

One measure of success for evaluating transfer learning is the time required to learn in the target task. By setting a threshold level of performance in the target task, we are able to measure the amount of training time needed to achieve this performance. By this measure, transfer learning is effective if we learn the target task faster by utilizing policies trained in the source task than by learning from scratch. A stronger criterion of transfer success is that the training time for the source and target tasks combined is shorter than the training time to learn the target task from scratch. We will show that transfer via inter-task mapping with NEAT is able to meet both of these transfer goals.

[3]Note that other domains may not have such straightforward mappings between tasks of different complexity.

Training Times for Performance Thresholds

Threshold	Scratch	ρ_5	ρ_{10}	Total ρ_5	Total ρ_{10}
7.0	87.3	30.2	50.1	96.3	199.3
7.5	152.1	40.2	80.9	106.4	230.2
8.0	286.7	64.8	116.7	130.9	250.9
8.5	416.6	124.3	168.4	183.0	302.6
9.0	474.4	229.0	229.5	281.2	363.8

Table 2. This table shows the average source and total training times (in hours) for players learning from scratch, via transfer after 5 source generations, and via transfer after 10 source generations.

To quantify how fast the agents learn, we set threshold performance values for the 4 vs. 3 task. We analyze the champions of each generation after learning and determine when the organism identified as the best by NEAT has learned to hold the ball for at least the threshold value, averaged over 1,000 episodes. If a NEAT trial does not reach the threshold value within 500 hours, we assign it a time of 500 hours[4].

Table 2 shows that the time it takes sets of 4 agents to learn to hold the ball for some amount of time in the target task can be reduced by utilizing ρ_5 and ρ_{10}, which respectively represents applying the inter-task mapping ρ after five and ten generations of learning in the source task. Each result is averaged over ten independent runs. A Student's t-test confirms that the difference between ρ and scratch is statistically significant at the 95% level for all points graphed.

Table 2 also shows the total training time (3 vs. 2 plus 4 vs. 3), which is also reduced when using ρ. The difference between scratch and ρ from five 3 vs. 2 generations is significant for all test threshold times above 7.0 seconds when considering total training time. The difference between scratch and ρ using ten 3 vs. 2 generations is significant for all points graphed except 8.0 seconds.

Transferring from five 3 vs. 2 episodes was more beneficial than from ten 3 vs. 2 episodes and we hypothesize this is due to two factors. Networks for five generations in 3 vs. 2 had more links and nodes than those evolved for ten generations, and more complex networks will likely take longer to train. Secondly, training for ten generations in 3 vs. 2 may have also overfit the task; similar results were seen our previous work (Taylor et al., 2005) where training for less time in the source task was more beneficial to transfer.

9. Related Work

There have been previous approaches to simplifying reinforcement learning by manipulating the transition function, the agent's initial state, and/or the reward function. For instance, *Directed training* (Selfridge et al., 1985) allows

[4]500 hours of training time corresponds to approximately 30 generations (300,000 episodes) of Keepaway.

a researcher to modify the transition function over time and slowly make the task harder. *Learning from easy missions* (Asada et al., 1994) allows a gradual modification of the start state so that the learner is initially placed near a goal state and gradually placed further and further away form it. However, these methods rely on S and A remaining the same between pairs of tasks. Transfer via inter-task mapping permits this, which allows transfer to be applied to a larger set of tasks. Furthermore, transfer via inter-task mapping does not preclude human modification of the transition function, the start state, or the reward function to increase the speed of learning and can therefore be combined with these other methods if desired.

Learned subroutines have been successfully transfered in hierarchical RL (Andre & Russell, 2002) by analyzing subroutines to identify those that can be directly reused in a target task. If a task can be formulated in a relational reinforcement learning setting, it may also be mastered more quickly via transfer learning (Morales, 2003). *Imitation* (Price & Boutilier, 2003) is another technique which may transfer knowledge from one learner to another. 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 without explicit knowledge transfer.

Another related approach uses linear programming to determine value functions for classes of similar agents (Guestrin et al., 2003). Automatically generated advice can also be used to speed up learning in transfer (Torrey et al., 2005) by utilizing a human to provide a mapping for this advice into the new task, similar to ρ. Other work in transfer learning (Fernandez, 2005; Konidaris, 2005) allows speedup between tasks but does not allow S and A to differ between the two (or more) tasks.

10. Conclusions

We have extended transfer via inter-task mapping to policy search methods and empirically shown that it can significantly speed up learning in pairs of related RL tasks. We utilized NEAT, a popular GA method, to learn on pairs of Keepaway tasks with different state and action spaces. Transferring learned policies the two tasks reduces not only training time in the target task, but also total training time.

11. Acknowledgments

We would like to thank the anonymous reviewers, Nicholas K. Jong, Shivaram Kalyanakrishnan, and Joe Reisinger for helpful comments and suggestions. This research was supported in part by DARPA grant HR0011-04-1-0035, NSF CAREER award IIS-0237699, and NSF award EIA-0303609.

References

Andre, D., & Russell, S. J. (2002). State abstraction for programmable reinforcement learning agents. *Proceedings of the Eighteenth National Conference on Artificial Intelligence* (pp. 119–125).

Asada, M., Noda, S., Tawaratsumida, S., & Hosoda, K. (1994). Vision-based behavior acquisition for a shooting robot by using a reinforcement learning. *Proc. of IAPR/IEEE Workshop on Visual Behaviors-1994* (pp. 112–118).

Fern, A., Yoon, S., & Givan, R. (2004). Approximate policy iteration with a policy language bias. In S. Thrun, L. Saul and B. Schölkopf (Eds.), *Advances in neural information processing systems 16*. Cambridge, MA: MIT Press.

Fernandez, F. (2005). *Probabilistic reuse of past policies* (Technical Report CMU-CS-05-173). School of Computer Science, Carnegie Mellon University.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. (2003). Generalizing plans to new environments in relational mdps. *International Joint Conference on Artificial Intelligence (IJCAI-03)*. Acapulco, Mexico.

Konidaris, G. (2005). *Autonomous sharing: Learning to predict reward for novel states* (Technical Report UM-CS-2005-58). Department of Computer Science, University of Massachusetts - Amherst.

Morales, E. F. (2003). Scaling up reinforcement learning with a relational representation. *Proc. of the Workshop on Adaptability in Multi-agent Systems*.

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

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc.

Selfridge, O., Sutton, R. S., & Barto, A. G. (1985). Training and tracking in robotics. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 670–672).

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

Stanley, K. O., & Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, *21*, 63–100.

Stone, P., Kuhlmann, G., Taylor, M. E., & Liu, Y. (2006). Keepaway soccer: From machine learning testbed to benchmark. In I. Noda, A. Jacoff, A. Bredenfeld and Y. Takahashi (Eds.), *RoboCup-2005: Robot soccer world cup IX*, vol. 4020. Berlin: Springer Verlag. To appear.

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

Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning*. MIT Press.

Taylor, M., Whiteson, S., & Stone, P. (2006). Comparing evolutionary and temporal difference methods for reinforcement learning. *Proceedings of the Genetic and Evolutionary Computation Conference*. To appear.

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

Torrey, L., Walker, T., Shavlik, J., & Maclin, R. (2005). Using advice to transfer knowledge acquired in one reinforcement learning task to another. *Proceedings of the Sixteenth European Conference on Machine Learning*.