Rule Extraction for Transfer Learning

Lisa Torrey[1], Jude Shavlik[1], Trevor Walker[1] and Richard Maclin[2]

University of Wisconsin - Madison and University of Minnesota - Duluth

1 Introduction

Typically rule extraction is done for the purposes of human interpretation. However, there are other possible applications of rule extraction. One practical application is *transfer learning*, in which knowledge learned in one task is used to aid in learning a related task. The extracted rules, which explain the learned solution to the first task, can be considered *advice* on how to approach the second task.

Transfer learning, besides being desirable in its own right, could be viewed as another way to evaluate extracted rules. That is, how well extracted knowledge transfers to a related task is a potential way of judging the value of the rule extraction algorithm. Thus transfer can be used as an alternative to traditional measures such as complexity and faithfulness to the original model. While this method is more objective and more computational than some of the traditional measures, it requires a trusted algorithm for making use of extracted knowledge.

This chapter discusses transfer learning via advice taking, in particular for reinforcement learning (RL) tasks that use support vector machines (SVMs) as function approximators. After some background information on transfer, advice, and RL with SVMs, it describes two methods for rule extraction in this context and presents a case study from our recent research.

2 Transfer Learning and Advice Taking

Machine learning tasks are often addressed independently, under the implicit assumption that each new task has no relation to the tasks that came before. However, many machine learning domains contain several related tasks. Instead of learning each one from scratch, agents in such domains should be able to use knowledge learned in previous tasks to speed up learning in later ones. This is the goal of *transfer learning* (see Figure 1).

For example, consider the domain of simulated soccer (e.g., RoboCup [10]). Suppose an agent has learned a game of keeping the ball from its opponents by passing amongst its teammates, and the next game to learn is to score goals against opponents. Since these games have some similarities, the agent could benefit from using its knowledge from the first game while learning the second.

Appears as a chapter in *Rule Extraction from Support Vector Machines*, pp. 67–82, edited by J. Diederich, Springer 2008.

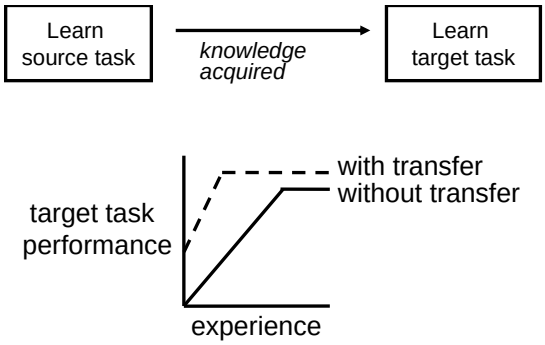

Fig. 1. With transfer from a related source task, the learning curve for the target task might improve in one or more of the ways shown above: higher initial performance, faster performance increase, and higher asymptotic performance.

In this case we refer to the first game as the *source task* and the second game as the *target task*.

There are several approaches to transfer learning in RL that do not involve rule extraction. Examples are Taylor et al. [15], who transfer Q-functions directly, and Soni and Singh [11], who transfer multi-step action sequences known as options. Here, however, we will focus on approaches that use rule extraction and apply those rules as advice for the target task.

Advice is a set of approximately correct instructions for a task. It may have some errors, and it usually does not provide a complete solution. Due to differences between the source and target tasks in transfer learning, extracted rules are likely to have these characteristics. In advice-taking RL algorithms, advice can be followed, refined, or ignored according to its value. In the target task, this means extracted rules can be obeyed if they lead to positive transfer, but quickly get refined or ignored if they lead to negative transfer.

For example, in the simulated soccer domain where the state features are distances and angles between players, some reasonable advice might look like:

IF distance(nearestOpponent, self) ≤ 5 AND
angle(teammate, self, minAngleOpponent(teammate)) ≥ 40
THEN prefer pass(teammate) over other actions

This advice tells an agent to pass to a teammate when two conditions hold: 1) an opponent is too close to the agent, and 2) the smallest angle from the teammate to the agent to an opponent is large enough (i.e. there is an open passing lane).

There is a substantial body of work on advice taking in RL. Examples are Clouse and Utgoff [2], who allow a human observer to step in and advise the learner to take a specific action; Driessens and Dzeroski [4], who use human guidance to create a partial initial Q-function; and Kuhlmann et al. [5], who propose giving advice that increases Q-values by a fixed amount. As most advice-taking approaches do, these studies assume that advice comes directly from a

human interacting with the learner. Here, however, we will focus on advice that is automatically extracted from a source task.

3 SVMs in Reinforcement Learning

In reinforcement learning [14], an agent navigates through an environment trying to earn rewards. The environment's state is typically described by a set of features. After each action the agent takes, it receives a reward and observes the next state.

In Q-learning [19], one common form of RL, the agent builds a Q-function to estimate the long-term value of taking an action from a state. An agent's *policy* is typically to take the action with the highest Q-value in the current state, except for occasional exploratory actions that are needed to discover better policies. After taking the action and receiving a reward, the agent updates its Q-value estimates for the current state.

The Q-function can be approximated with SVM regression models [3], so that each action's Q-value is estimated by a weighted linear sum of the state features. In this case, after taking a sequence of actions and receiving a corresponding sequence of rewards, the RL agent learns a linear SVM with weights that minimize:

$$\text{ModelSize} + C \times \text{DataMisfit}$$

Here *ModelSize* is the sum of the absolute values of the feature weights, and *DataMisfit* is the disagreement between the learned function's outputs and the correct outputs (estimated based on the rewards received). The numeric parameter C specifies the relative importance of minimizing disagreement with the data versus finding a simple model.

The agent learns many intermediate SVM models as its performance improves. Eventually it reaches an asymptote, so there is a final model that represents the learned task. This is the model from which rules are extracted to perform transfer learning.

Advice can be included in this RL algorithm with *Knowledge-Based Support Vector Regression*, abbreviated KBKR [6–8]. This algorithm adds another term to the optimization problem, so that it minimizes:

$$\text{ModelSize} + C \times \text{DataMisfit} + \mu \times \text{AdviceMisfit}$$

Here *AdviceMisfit* is the disagreement between the learned function's outputs and the advice constraints. The numeric parameter *mu* specifies the relative importance of minimizing disagreement with the advice versus minimizing the original quantity. Over time *mu* decays and C increases so that the advice has less impact as the learner gains experience and no longer requires guidance.

Advice therefore becomes a soft constraint on the task solution. Depending on whether the advice agrees with the training examples, the learner can fully

follow the rule, only follow it approximately (which is like refining it), or ignore it altogether.

The details behind this intuitive idea are as follows. Let A be a matrix in which each row contains the feature values for a training example. Let y be the vector of Q-value estimates (for a single action) for this set of examples. We wish to model the Q-function as a weighted sum:

$$Aw + b\overrightarrow{e} = y \tag{1}$$

where w is a vector of weights, b is a scalar offset, and $\overrightarrow{e}$ denotes a vector of ones (we omit this for simplicity from now on).

To learn a good Q-function, we want to find w and b to satisfy this equation. However, an exact solution may not exist. Also, it is preferable to have non-zero weights for only a few important features in order to keep the model simple and avoid overfitting the training examples. Therefore we include a vector of *slack* variables s to allow inaccuracies on some examples, and a penalty parameter C for trading off these inaccuracies with the complexity of the solution. The linear equation then becomes a linear minimization problem:

$$\begin{array}{l} \min\limits_{(w,b,s)} \; ||w||_1 + \nu|b| + C||s||_1 \\ s.t. \quad |Aw + b - y| \leq s. \end{array} \tag{2}$$

where $|\cdot|$ denotes an absolute value, $||\cdot||_1$ denotes a sum of absolute values from a vector, and ν is a penalty on the offset term that discourages constant models. Solving this problem means finding w, b, and s such that the penalties are minimized but the model's value for $A_i w + b$ is within s_i of y_i. The RL agent therefore finds a compromise between accuracy and simplicity.

Advice generated from the source task can be expressed in the form:

$$Bx \leq d \Longrightarrow Q_p(x) - Q_n(x) \geq \beta, \tag{3}$$

where B is a matrix and d is a vector [7]. This can be read as: if the current state satisfies the set of linear inequalities $Bx \leq d$, the Q-value of the preferred action p should exceed that of the non-preferred action n by at least β. For example, consider the advice rule from Section 2:

IF distance(nearestOpponent, self) ≤ 5 AND
angle(teammate, self, minAngleOpponent(teammate)) ≥ 40
THEN prefer pass(teammate) over other actions

If we assume that the two features mentioned in this rule make up the entire feature vector x, then we would express this advice with:

$$B = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$d = \begin{pmatrix} 5 \\ -40 \end{pmatrix} \tag{4}$$

where Q_p represents the Q-value of *pass*, β is a constant small fraction of the Q-value range, and there is one equation for each other action n. Note that the $\geq$ inequality in the second constraint is converted to a $\leq$ inequality by multiplying both sides of the equation by -1.

Just as our method allows some inaccuracy on the training examples, it allows advice to be satisfied only partially. To do so, following Mangasarian et al. [8], we introduce slack variables z and ζ and penalty parameters μ_1 and μ_2 for trading off the impact of the advice on the solution with the impact of the training examples. The resulting linear program, which finds Q-functions for all of the actions simultaneously, is:

$$\begin{aligned}
&\min_{(w_a, b_a, s_a, z_i, \zeta_i \geq 0, u_i \geq 0)} \\
&\sum_{a=1}^{m} (||w_a||_1 + \nu |b_a| + C||s_a||_1) + \sum_{i=1}^{k} (\mu_1 ||z_i||_1 + \mu_2 \zeta_i) \\
&\quad \text{s.t.} \quad \text{for each action } a \in \{1, \ldots, m\} : \\
&\qquad\qquad |A_a w_a + b_a - y_a| \leq s_a \\
&\qquad\qquad \text{for each advice item } i \in \{1, \ldots, k\} : \\
&\qquad\qquad |w_p - w_n + B_i^T u_i| \leq z_i \\
&\qquad\qquad - d^T u_i + \zeta_i \geq \beta_i - b_p + b_n.
\end{aligned} \tag{5}$$

By solving this optimization problem, the RL agents learn a policy that satisfies the advice as long as it does not disagree too much with the training examples.

4 Extracting Rules from an RL Source Task

The final SVM model representing the learned RL task can be separated into several models (one per action). Each SVM calculates the Q-value of one action as a weighted sum of the state features. We discuss two methods for extracting transfer advice from these models. In both approaches, the resulting rules are of the form:

IF *condition*
THEN prefer one action over the others

where the *condition* is a conjunction of linear constraints on the state features.

As is common in transfer learning, our methods assume the existence of a *mapping* showing how features and actions correspond between the source and target tasks. Transfer is a reasonable endeavor only if there is substantial correspondence. However, there may be some features and actions in one task that do not have parallels in the other, and the transfer methods we discuss handle this problem in different ways.

4.1 Acquiring Rules from the Q-function

One approach for generating rules from RL source-task models first appeared in Torrey et al. [18]. It is called *policy transfer*, because it builds a set of rules to describe the entire policy represented by the source-task models. To reference the neural-network rule-extraction literature, this is best described as a *decompositional* strategy [1], in which the internal mechanics of the model affect the extracted rules.

Recall that the *policy* of an RL agent determines which action it will choose– generally the action with the highest Q-value. The policy can therefore be expressed as a set of rules, one for each action, saying to prefer that action when its SVM regression model assigns it the highest Q-value. Alternatively, it can be expressed as a set of rules, two for each pair of actions, saying to prefer one action over the other when its SVM regression model assigns it the higher Q-value of the pair. Table 1 gives a simple example of the construction of this pairwise ruleset.

The policy-transfer rules effectively tell the target-task learner to pretend it is performing the source task (via the mapping) and choose actions accordingly. They do not attempt to constrain the actual Q-values of target-task actions, but only the relative ordering of the Q-values; this can be important if the Q-value ranges of the tasks differ. We set the parameter Δ to approximately 1% of the target task's Q-value range.

One complication that might arise for this method is if a source-task feature has no corresponding feature in the target task– for example, if the f_1 in Table 1 has no logical f'_1 mapping. In this case, the algorithm gives f'_1 a constant value. By default it uses the average value that the f_1 feature takes in the source-task data, although the user may also tell it to use the minimum or maximum value if that seems more appropriate.

Table 1. An example of constructing policy-transfer rules. The actions in the old task are a, b, and c, and the corresponding actions in the new task are a', b', and c'. The learned models for the old task are linear Q-value expressions with weights w and features f, and these are translated into rules that use the corresponding new task features f'.

SOURCE TASK MODEL:	ADVICE FORMAT:
$Q_a = w_{a1} * f_1 + w_{a2} * f_2$	IF $Q'_a - Q'_b \geq \Delta$
$Q_b = w_{b1} * f_1$	THEN prefer a' to b'
$Q_c = w_{c2} * f_2$	(and so on for each pair of actions)
USER-PROVIDED MAPPING:	FULL ADVICE EXPRESSION:
$(a, b, c) \longrightarrow (a', b', c')$	IF $(w_{a1} - w_{b1}) * f'_1 + w_{a2} * f'_2 \geq \Delta$
$(f_1, f_2) \longrightarrow (f'_1, f'_2)$	THEN prefer a' to b'
TRANSLATED MODEL:	(and so on for each pair of actions)
$Q'_a = w_{a1} * f'_1 + w_{a2} * f'_2$	
$Q'_b = w_{b1} * f'_1$	
$Q'_c = w_{c2} * f'_2$	

Table 2. Our policy-transfer algorithm.

GIVEN
  A learned source-task model AND
  A mapping of features and actions from source to target
DO
  for each $a \in Actions(source)$:
    for each $b \neq a \in Actions(source)$ generate advice:
      IF $Q'_a - Q'_b \geq \Delta$
      THEN prefer a' TO b' in target task

Table 2 summarizes our policy-transfer algorithm in pseudocode.

We present experiments with policy transfer as part of a case study in Section 5. The rules constructed by this method are long, complex and not well suited to human interpretation. They capture very fine details of the source-task models, which may not actually be desirable for transfer learning because general principles are more likely to transfer to new tasks than specific details. The approach we discuss next was designed to address these shortcomings.

4.2 Acquiring Rules from Observed Behavior

A second approach for generating rules from RL source-task models is presented in Torrey et al. [17]. It is called *skill transfer*, because it learns rules that represent important source-task skills. In the neural-network rule-extraction literature, this falls under the category of *pedagogical* strategies [1], in which the model is treated as a black box and the rules mimic its outputs.

Skill transfer is intended to capture general knowledge rather than fine detail. Instead of transferring an entire policy, this method transfers only the skills that the source and target tasks have in common. A *skill* is associated with one action, and describes the circumstances under which that action should be taken. Our skill transfer algorithm learns skills by observing behavior in the source task and applying inductive logic programming [9].

Inductive Logic Programming (ILP) is a method for learning first-order rules to explain examples. For example, recall the rule from Section 2:

IF distance(nearestOpponent, self) < 5 AND
   angle(teammate, self, minAngleOpponent(teammate)) > 40
THEN prefer pass(teammate) over other actions

This rule might describe the *pass* skill in soccer: pass to a teammate if an opponent is too close and the passing lane is open. If the symbol *teammate* refers to a specific teammate object, then the rule is called *propositional*; if it is a variable that can refer to any teammate object, then the rule can be called *first-order*. The first-order version is more powerful and more general than the propositional version, since it is more likely to capture the essential elements of the passing skill.

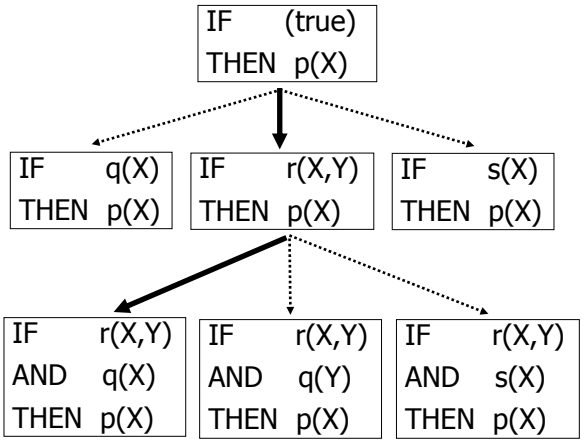

Fig. 2. A top-down heuristic ILP search that adds one constraint at a time. Here the lower-case letters are predicates and the upper-case letters are variables. The rule begins with no constraints, and at each step, the search adds the new constraint that the heuristic function scores highest, indicated by the solid arrow above. New variables that were not in the head $p(X)$ can be introduced into the body of the rule.

A common approach for ILP is to start by selecting a *seed* example, such as one state in which the soccer player performed the *pass* action. It then calculates a set of facts that are true for this example, which might include *distance(nearestOpponent, self)* < 5 and *angle(teammate, self, minAngleOpponent(teammate))* > 40 along with many other less relevant facts. The clause that contains all these constraints is called the *bottom clause* [12]. A rule is then learned by searching for some subset of the bottom clause that is more general, covering many positive examples but few negative examples. The search algorithm could be any of those familiar to students of general artificial intelligence, such as heuristic search and randomized search. Figure 2 gives an example of a top-down heuristic search.

We use the Prolog-based Aleph software package [12] to perform ILP. The metric we use to score rules and select the best is F(β), a generalization of the more familiar F(1) metric, with $\beta^2 = 0.1$ to put more weight on rule precision than rule recall. The search therefore concentrates on finding rules that cover a reasonable amount of data with high accuracy.

Skill transfer with ILP is accomplished by applying the standard ILP procedure using states from source-task games as the examples. One challenge for using ILP on reinforcement learning data is to decide which states are positive examples and which are negative examples. This is an important process, and it is not immediately obvious how the choices should be made. Figure 3 illustrates the procedure that we use in Torrey et al. [17] with an example from the simulated soccer domain.

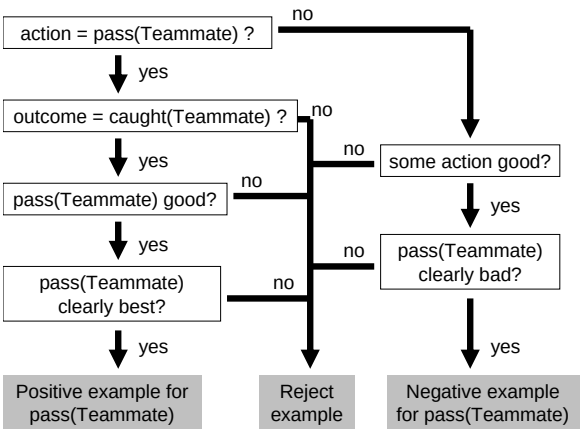

Fig. 3. Example showing how we select training examples for the skill pass(Teammate).

In a positive example, several conditions must be met: the skill is performed, the desired outcome occurs, and the expected Q-value is above the 10th percentile in the training set and is at least 1.05 times the predicted Q-values of all other actions. (The desired outcome of *pass(teammate)*, for example, is that the ball is caught by the *teammate*.) The purpose of these conditions is to remove ambiguous states in which several actions may be good or no actions seem good.

There are two potential types of negative examples. These conditions describe one type: some other action is performed, that action's Q-value is above the 10th percentile in the training set, and the Q-value of the skill being learned is at most 0.95 times that Q-value and below the 50th percentile in the training set. This again rules out ambiguous states. The second type of negative example includes states in which the skill being learned was taken but the desired outcome did not occur (for example, *pass(teammate)* was taken but the ball was caught by an opponent).

Note that the source and target-task features remain propositional in skill transfer, since we use a linear SVM model that requires a fixed-length feature vector. After learning first-order rules, our skill-transfer algorithm propositionalizes them for use in the target task.

As in policy transfer, there may be features that exist in the source but not the target. Here there is a simple solution, however: we restrict the search space of ILP so that learned rules may only contain features that both tasks share. This forces the algorithm to consider only skill definitions that are relevant in the target task.

As in policy transfer, a rule learned by the skill-transfer method describes the conditions under which one action should be preferred over the other actions shared between the source and target task. However, these rules are much simpler and easier to interpret. These are desirable qualities for extracted rules in general, and because they imply more general rules, for transfer learning as well.

Table 3. Our skill-transfer algorithm.

GIVEN	DO
Games from source task	For each skill to transfer:
List of skills to be transferred	Collect training examples
User advice (optional)	Learn rules with Aleph
	Select rule with highest $F(\beta)$ score
	Propositionalize rules for target task

Table 3 summarizes our skill transfer algorithm in pseudocode. We present experiments with skill transfer as part of the case study in Section 5.

Because the skill-transfer rules are more accessible to human understanding, they open up more possibilities for further human contribution. The user could add constraints to the learned rules reflecting known differences in the source and target tasks, or even provide simple new rules for skills required in the target task that were not in the source. We call this *user advice*, and include an example in the case study. User advice provides a natural and powerful way for a human to guide transfer.

5 Case Study

To illustrate the transfer learning techniques in this chapter, we present a case study in the simulated soccer domain, which is a motivating domain for transfer. The results are reproduced from Torrey et al. [16].

The RoboCup project [10] has the overall goal of producing robotic soccer teams that compete on the human level, but it also has a software simulator for research purposes. Stone and Sutton [13] introduced RoboCup as an RL domain that is challenging because of its large, continuous state space and non-deterministic action effects.

Since the full game of soccer is quite complex, researchers have developed several smaller games in the RoboCup domain (see Figure 4). These are inherently multi-agent games, but a standard simplification is to have only one agent (the one in possession of the soccer ball) learning at a time using a model built with combined data from all the agents.

One RoboCup task is M-on-N KeepAway [13], in which the objective of the M reinforcement learners called *keepers* is to keep the ball away from N hand-coded players called *takers*. The keeper with the ball may choose either to hold it or to pass it to a teammate. Keepers without the ball follow a hand-coded strategy to receive passes. The game ends when an opponent takes the ball or when the ball goes out of bounds. The learners receive a +1 reward for each time step their team keeps the ball.

The KeepAway state representation was designed by Stone and Sutton [13] and consists of distances and angles between players. The keepers are ordered by their distance to the learner *k0*, as are the takers.

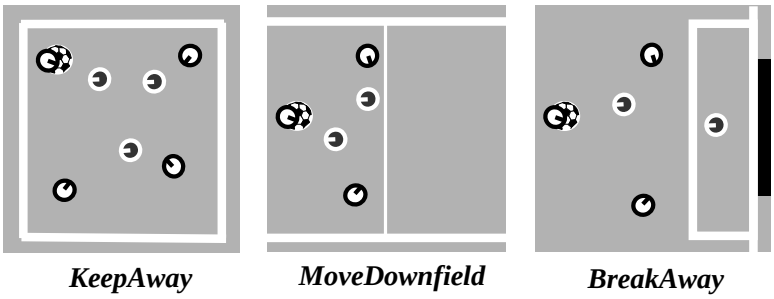

Fig. 4. Snapshots of RoboCup soccer tasks.

A second RoboCup task is M-on-N MoveDownfield, where the objective of the M reinforcement learners called *attackers* is to move across a line on the opposing team's side of the field while maintaining possession of the ball. The attacker with the ball may choose to pass to a teammate or to move ahead, away, left, or right with respect to the opponent's goal. Attackers without the ball follow a hand-coded strategy to receive passes. The game ends when they cross the line, when an opponent takes the ball, when the ball goes out of bounds, or after a time limit of 25 seconds. The learners receive symmetrical positive and negative rewards for horizontal movement forward and backward.

The MoveDownfield state representation was introduced in Torrey et al. [17] and consists of distances and angles between players and the goal. The attackers are ordered by their distance to the learner *a0*, as are the defenders.

A third RoboCup task is M-on-N BreakAway, where the objective of the M attackers is to score a goal against $N-1$ hand-coded *defenders* and a hand-coded *goalie*. The attacker with the ball may choose to pass to a teammate, to move ahead, away, left, or right with respect to the opponent's goal, or to shoot at the left, right, or center part of the goal. Attackers without the ball follow a hand-coded strategy to receive passes. The game ends when they score a goal, when an opponent takes the ball, when the ball goes out of bounds, or after a time limit of 10 seconds. The learners receive a $+1$ reward if they score a goal, and zero reward otherwise.

The BreakAway state representation was introduced in Torrey et al. [18] and consists of distances and angles between players and the goal. The attackers are ordered by their distance to the learner *a0*, as are the non-goalie defenders.

These three RoboCup games have substantial differences in features, actions, and rewards. The goal, goalie, and shoot actions exist in BreakAway but not in the other two tasks. The move actions do not exist in KeepAway but do in the other two tasks. Rewards in KeepAway and MoveDownfield occur for incremental progress, but in BreakAway the reward is more sparse. These differences mean the solutions to the tasks may be quite different. However, some knowledge should clearly be transferable between them, since they share many features and some actions, such as the *pass* action. Furthermore, since these are difficult RL tasks, speeding up learning through transfer is desirable.

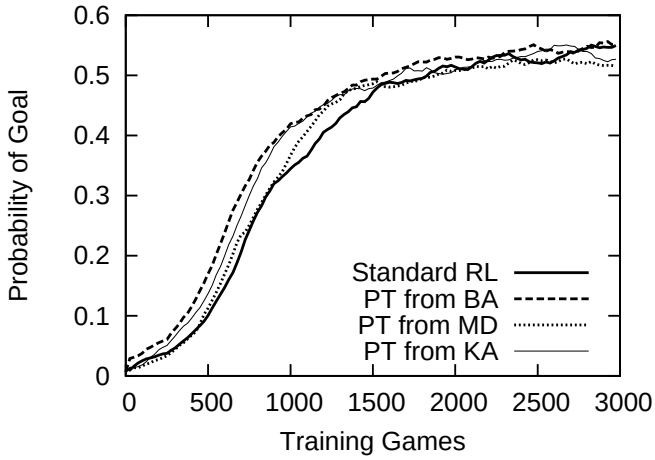

Fig. 5. Probability of scoring a goal while training in 3-on-2 BreakAway with standard RL and policy transfer (PT) from 2-on-1 BreakAway (BA), 3-on-2 MoveDownfield (MD) and 3-on-2 KeepAway (KA).

5.1 Policy-Transfer Results

Figure 5 displays results from several policy-transfer experiments. The target task in each experiment is 3-on-2 BreakAway, and the three source tasks are 2-on-1 BreakAway, 3-on-2 MoveDownfield, and 3-on-2 KeepAway. One curve is the average of 25 runs of standard reinforcement learning. The other curves are RL with transfer via model reuse from various source tasks. Each transfer curve is an average of 5 transfer runs from 5 different source runs, for a total of 25 runs (this way, the results include both source and target variance). Because the variance is high, the y-value at each data point is smoothed by averaging over the y-values of the last 10 data points.

The extracted rules are too large and complex to include an example here. However, the results in Figure 5 show that policy transfer has a small overall positive impact, particularly when the source and target tasks are most similar.

5.2 Skill-Transfer Results

Figure 6 displays results from several skill-transfer experiments. The source and target tasks are the same as in the policy-transfer experiments. Again each transfer curve is an average of 25 runs, consisting of 5 runs from 5 different source runs, with the y-value at each point smoothed over the last 10 points.

For the experiments in which the target was a different task than the source, we provided some simple handwritten user advice to help with the new required skills. In transfer from KeepAway to BreakAway the important new skills are moving ahead and shooting, and from MoveDownfield to BreakAway only shooting. From one size of BreakAway to another no user advice is needed.

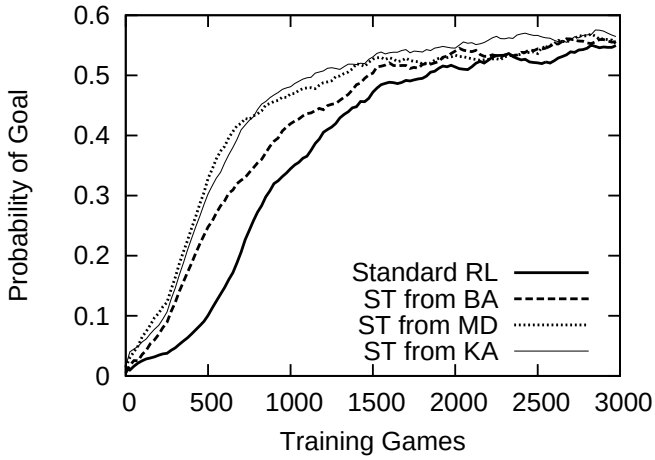

Fig. 6. Probability of scoring a goal while training in 3-on-2 BreakAway with standard RL and skill transfer (ST) from 2-on-1 BreakAway (BA), 3-on-2 MoveDownfield (MD) and 3-on-2 KeepAway (KA).

We took the appropriate subset of user advice from this set:

IF distBetween(a0, GoalRight) $<$ 10 AND
angleDefinedBy(GoalRight, a0, goalie) $>$ 40
THEN prefer shoot(GoalRight) over other actions

IF distBetween(a0, GoalLeft) $<$ 10 AND
angleDefinedBy(GoalLeft, a0, goalie) $>$ 40
THEN prefer shoot(GoalLeft) over other actions

IF distBetween(a0, goalCenter) $>$ 10
THEN prefer moveAhead over other actions

Note that this user advice is not tuned. By adjusting the numerical values in the rules above, it is possible to improve the performance further. However, we assume that users will only give approximate advice.

To give an example of the skill concepts learned, the following is an example rule for *pass* that our skill transfer algorithm extracted from 3-on-2 MoveDownfield:

IF distBetween(Teammate, fieldCenter) $\geq$ 6,
distBetween(Teammate, minDistTaker(Teammate)) $\geq$ 8,
angleDefinedBy(Teammate, a0, minAngleTaker(Teammate)) $\geq$ 41 AND
angleDefinedBy(OtherTeammate, a0, minAngleTaker(OtherTeammate)) $\leq$ 23
THEN prefer pass(Teammate) over other actions

This rule specifies a minimum pass angle and an open distance around the receiving teammate. It also requires that the teammate not be too close to the center of the field and gives a maximum pass angle for the alternate teammate.

The following is an example rule for *shoot* extracted from 2-on-1 BreakAway:

IF distBetween(a0, goalCenter) $\geq$ 6,
angleDefinedBy(GoalPart, a0, goalie) $\geq$ 52,
distBetween(a0, oppositePart(GoalPart)) $\geq$ 6,
angleDefinedBy(oppositePart(GoalPart), a0, goalie) $\leq$ 33 AND
angleDefinedBy(goalCenter, a0, goalie) $\geq$ 28
THEN prefer shoot(GoalPart) over other actions

This rule requires a large open shot angle, a minimum distance to the goal, and angle constraints that restrict the goalie's position to a small area.

The results in Figure 6 show that skill transfer can have a large overall positive impact in most transfer scenarios. The skill-transfer method of rule extraction not only produces more understandable rules than policy transfer, but also leads to better performance in the target task.

6 Summary and Open Problems

Rule extraction can have practical applications beyond explaining a machine-generated solution to humans. Transfer learning, in which the solution to one task is used while learning a related task, is one such application. The transfer learning framework could be also viewed as a way to evaluate rulesets based on their ability to improve learning in a related task.

This chapter discussed ways to transfer rules between SVM-based reinforcement learning tasks. In this context, it is more important to obtain general rules that express the basic skill concepts than to describe the specifics of the source-task model. Therefore, learning by observing behavior is more effective than building rules from complex Q-functions. The use of inductive logic programming is also beneficial because it allows rules to employ first-order logic, which makes them more general.

We use extracted rules as advice for a target task. With an SVM-based reinforcement learner, there is a straightforward method of incorporating advice as a soft constraint. This allows the rules to improve learning in the target task to the extent that they are relevant, but also provides protection against negative transfer effects.

There are several open problems in this area, in both the rule-extraction and advice-taking steps. One is to develop advice-taking methods for relational reinforcement learning (RRL), so that first-order advice can be applied directly without propositionalizing it first. Another is to extract rules that describe multiple-step plans rather than single-step action choices, which might capture more information from the source task than the current approaches do.

The research referenced in this chapter was supported by DARPA grant HR0011-04-1-0007.

References

1. R. Andrews, J. Diederich, and A. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. In *Knowledge Based Systems*, 1995.
2. J. Clouse and P. Utgoff. A teaching method for reinforcement learning. In *Proceedings of the 9th International Conference on Machine Learning*, 1992.
3. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.
4. K. Driessens and S. Dzeroski. Integrating experimentation and guidance in relational reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, 2002.
5. G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In *AAAI Workshop on Supervisory Control of Learning and Adaptive Systems*, 2004.
6. R. Maclin, J. Shavlik, L. Torrey, and T. Walker. Knowledge-based support vector regression for reinforcement learning. In *IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, 2005.
7. R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proceedings of the 20th National Conference on Artificial Intelligence*, 2005.
8. O. Mangasarian, J. Shavlik, and E. Wild. Knowledge-based kernel approximation. *Journal of Machine Learning Research 5*, pages 1127–1141, 2004.
9. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming 19,20*, pages 629–679, 1994.
10. I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.
11. V. Soni and S. Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 2006.
12. A. Srinivasan. *The Aleph Manual*, 2001.
13. P. Stone and R. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *Proceedings of the 18th International Conference on Machine Learning*, 2001.
14. R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
15. M. Taylor, P. Stone, and Y. Liu. Value functions for RL-based behavior transfer: A comparative study. In *Proceedings of the 20th National Conference on Artificial Intelligence*, 2005.
16. L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Advice-based transfer in reinforcement learning. Technical Report TR06-2, Machine Learning Research Group, U. Wisconsin-Madison, 2006.
17. L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Relational skill transfer via advice taking. In *Proceedings of the 17th European Conference on Machine Learning*, 2006.
18. L. Torrey, T. Walker, J. Shavlik, and R. Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Proceedings of the 16th European Conference on Machine Learning*, 2005.
19. C. Watkins. Learning from delayed rewards. Technical Report PhD Thesis, University of Cambridge, Psychology Dept., 1989.