
Transfer of Knowledge Structures with Relational Temporal Difference Learning

David Stracuzzi
Nima Asgharbeygi

STRACUDJ@CSLI.STANFORD.EDU
NIMAA@STANFORD.EDU

Center for the Study of Language and Information, Computational Learning Laboratory,
Stanford University, Stanford, CA 94305 USA

Abstract

The ability to transfer knowledge from one domain to another is an important aspect of learning. Knowledge transfer increases learning efficiency by freeing the learner from duplicating past efforts. In this paper, we demonstrate how reinforcement learning agents can use relational representations to transfer knowledge across related domains.

1. Introduction

Traditionally, research in the machine learning community has focused on learning in the context of individual domains. For a given domain, an appropriate learning method is selected and executed. The result is used within the context of the initial (training) domain, but not typically applied toward learning in any other domain, related or otherwise.

Recent research into transfer learning takes a different view. Learners are asked to perform on a sequence of two or more related tasks. The goal is to find ways to transfer knowledge acquired in one domain (the source) into subsequent domains (the target). The desired outcome is that effort expended on learning in the source domain(s) will reduce the amount of effort required to learn in the target domain(s).

In this paper, we consider how to apply reinforcement learning to the transfer task. Toward this end, we introduce a relational version of the popular TD(λ) algorithm (Sutton, 1988). Our approach is similar to that of Dzeroski et al. (2001), who discuss the use relational regression to learn Q-functions. This work takes a simpler approach to computing the value function, and focuses more on the underlying concept hierarchy.

2. Relational Temporal Difference Learning

Most work in reinforcement learning relies on propositional representations. This includes simple tabular approaches, the use of hand-crafted features to summarize important properties of states, and even the use of function approximators, such as backpropagation. None of these methods lend themselves to the problem of transferring knowledge from one domain to another. This is due in large part to the inflexibility of the underlying propositional representation.

We consider here a different view on approximate value function representation. Suppose that the states in a domain are factored, such that each state s is a set of relational ground literals from a finite set of possible ground literals. More formally, let $\mathcal{L} = \{L_1, \dots, L_n\}$ be a set of first-order predicates.

Each valid binding of constants to parameters gives a ground literal of predicate L_i . Denote the set of all groundings of L_i by \mathcal{I}_{L_i} . Then each state s_t is described as $s_t = I_{L_1}(s_t) \cup I_{L_2}(s_t) \cup \dots \cup I_{L_n}(s_t)$, in which $I_{L_i}(s_t) \subseteq \mathcal{I}_{L_i}$ denotes the set of groundings of predicate L_i that are true in state s_t . Thus, the predicates in \mathcal{L} must provide a complete description of the current state.

Given a factored state encoding, we can define a relational structure on top of this representation. Consider a hierarchical set of predicates $\mathcal{C} = \{C_1, \dots, C_m\}$, which we call *conceptual* predicates, each defined in terms of lower level predicates (including those from \mathcal{L}). The set I_{C_i} then represents the groundings of C_i given a set of ground literals.

Table 1 shows sample predicates from the game of TicTacToe. The predicate *Line* belongs to the set \mathcal{L} , which describes the game state. The predicates *CanMakeLine* and *HasFork* belong to the set \mathcal{C} and provide a more abstract view of the states. These con-

Table 1. A subset of predicates included in the sets \mathcal{L} and \mathcal{C} for the TicTacToe domain.

| |
|--|
| $\text{Line}(a_1, a_2, a_3)$ $\Leftarrow \text{Column}(a_1, a_2, a_3) \vee \text{Row}(a_1, a_2, a_3) \vee$ $\text{Diagonal}(a_1, a_2, a_3)$ |
| $\text{CanMakeLine}(p, b_1, b_2, b_3)$ $\Leftarrow (p = \text{X} \vee p = \text{O}) \wedge \text{Line}(b_1, b_2, b_3) \wedge$ $\text{Cell}(p, b_1) \wedge \text{Cell}(p, b_2) \wedge \text{Cell}(\text{EmptyCell}, b_3)$ |
| $\text{HasFork}(q, c_1, c_2, c_3, c_4, c_5)$ $\Leftarrow (q = \text{X} \vee q = \text{O}) \wedge (c_2 \neq c_4) \wedge$ $\text{CanMakeLine}(q, c_1, c_2, c_3) \wedge$ $\text{CanMakeLine}(q, c_1, c_4, c_5)$ |

cepts are used for evaluating states efficiently. Figure 2 shows a sample result of this inference process for a particular state.

Our approximate value function is distributed over this network of relational predicates \mathcal{C} . More precisely, we define a real-valued utility $U(C)$ for every concept $C \in \mathcal{L} \cup \mathcal{C}$ and approximate the value of any state s as

$$V_1(s) = \sum_{C \in W_{inf}(s)} |I_C(s)| \cdot U(C), \quad (1)$$

in which $W_{inf}(s) \subseteq \mathcal{L} \cup \mathcal{C}$ determines a subset of predicates that should influence the value of state s .

With respect to our example, each predicate in \mathcal{C} in Table 1 (*CanMakeLine* and *HasFork*) has an associated utility value. Following inference, the value of each concept’s utility is added to the value of the state. In Table 2 (c), each member of $I_C(s)$ adds utility to the value of a given state. Notice that, in this example, *CanMakeRow* has two distinct groundings. Each grounding carries the utility of the general predicate, so the value of the state shown in Table 2 (b) includes $2 \cdot U(\text{CanMakeRow}) + U(\text{HasFork})$.

Combining equation (1) with the standard TD(λ) update rule

$$\Delta V(s_k) = \alpha \lambda^{t-k} [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

yields an update for predicate values U :

$$U(C) := U(C) + |I_C(s_k)| \cdot \Delta V(s_k), \quad (2)$$

for all $C \in W_{inf}(s_k)$, $k = 1, \dots, t$. We call this class of learning algorithms rTD(λ), where “r” refers to the relational nature of the predicate hierarchy. Notice that rTD(λ) reduces to TD(λ) when there is a one-to-one mapping from states to predicates.

Table 2. Sample inference results (c) for TicTacToe assuming the cell constants shown in (a) and the game state shown in (b).

| | | | | | | | | | | | | | | | | | | | |
|---|---|-------|-------|-------|-------|-------|-------|-------|-------|--|---|--|---|---|---|--|---|--|--|
| <table style="border-collapse: collapse; margin: auto;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">C_1</td><td style="border-right: 1px solid black; padding: 2px 5px;">C_2</td><td style="padding: 2px 5px;">C_3</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">C_4</td><td style="border-right: 1px solid black; padding: 2px 5px;">C_5</td><td style="padding: 2px 5px;">C_6</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">C_7</td><td style="border-right: 1px solid black; padding: 2px 5px;">C_8</td><td style="padding: 2px 5px;">C_9</td></tr> </table> <p>(a)</p> | C_1 | C_2 | C_3 | C_4 | C_5 | C_6 | C_7 | C_8 | C_9 | <table style="border-collapse: collapse; margin: auto;"> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">X</td><td style="border-right: 1px solid black; padding: 2px 5px;"></td><td style="padding: 2px 5px;">X</td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">O</td><td style="border-right: 1px solid black; padding: 2px 5px;">X</td><td style="padding: 2px 5px;"></td></tr> <tr><td style="border-right: 1px solid black; padding: 2px 5px;">O</td><td style="border-right: 1px solid black; padding: 2px 5px;"></td><td style="padding: 2px 5px;"></td></tr> </table> <p>(b)</p> | X | | X | O | X | | O | | |
| C_1 | C_2 | C_3 | | | | | | | | | | | | | | | | | |
| C_4 | C_5 | C_6 | | | | | | | | | | | | | | | | | |
| C_7 | C_8 | C_9 | | | | | | | | | | | | | | | | | |
| X | | X | | | | | | | | | | | | | | | | | |
| O | X | | | | | | | | | | | | | | | | | | |
| O | | | | | | | | | | | | | | | | | | | |
| $I_L(s)$ | $I_C(s)$ | | | | | | | | | | | | | | | | | | |
| $\text{Cell}(X, C_1)$ $\text{Column}(C_1, C_4, C_7)$... $\text{Row}(C_1, C_2, C_3)$... $\text{Diagonal}(C_1, C_5, C_9)$... $\text{Line}(C_1, C_2, C_3)$... $\text{Line}(C_1, C_4, C_7)$ | $\text{CanMakeLine}(X, C_1, C_2, C_3)$ $\text{CanMakeLine}(X, C_1, C_5, C_9)$ $\text{HasFork}(X, C_1, C_2, C_3, C_5, C_9)$... | | | | | | | | | | | | | | | | | | |
| (c) | | | | | | | | | | | | | | | | | | | |

3. Experimental Demonstration

The purpose of our experimental evaluation is not to demonstrate learning in complex domains that are intractable for other learning methods. Instead, the goal is to study the behavior of relational temporal differencing in the context of transfer learning tasks. More specifically, we seek to demonstrate that rTD(λ) supports the representational abstractions required to transfer knowledge among multiple related domains.

3.1. The Learning Agent

The experiments reported here make use of two-player, deterministic Markov games. We therefore require generalized (minimax) versions of the single-agent temporal differencing update rules discussed above. Littman (1994) shows how to generalize single-agent Q-learning to minimax Q-learning. We used a similar generalization to derive the minimax temporal difference update rules. The minimax TD(λ) update is identical to the single agent update rule, except that each player has a separate value function computed independently from those of other players. In alternating-turn games, such as those considered here, the minimax policy reduces to a greedy policy. The agent therefore only needs to consider its legal moves at the current state.

Our learning agent is composed of an inference engine, a performance module, and a learning element. The performance module agent examines each candidate move from the current state, inferring the resulting state and predicate groundings. It then computes the approximate value of each next state using (1), and selects the action with the highest according to the

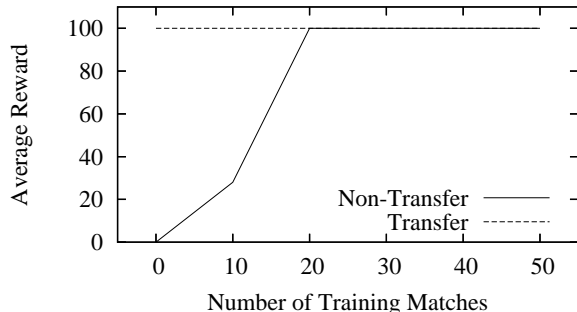


Figure 1. Learning curves for the KQKQP target.

current policy. The learning element then updates the predicate utilities according to (2).

3.2. Methodology

We begin discussion of our evaluation methodology by defining three terms. First, a *game* defines the environment in which the learner must operate. This includes a unique initial state, a unique action model, a set of terminal states, and a set of goal conditions. Second, a *match* is a competition between two agents (players) in a given game (environment). Each match begins with the initial state and ends when a terminal state is reached. Finally, a *scenario* is a source/target pairing of games such that the agent should be able to train in the source, and then transfer a portion of its acquired knowledge to improve performance (learning or otherwise) in the target.

To demonstrate the presence and effects of knowledge transfer, we first constructed two transfer learning *scenarios* derived from chess. Each game description consists of a set of predicates (the set \mathcal{L}) and includes a set of concepts that the agent could use to evaluate board positions (the set \mathcal{C}). Concepts for the source and target games in a scenario were identical, but the game descriptions were not. In both scenarios, the reward for a win (satisfying the goal) was 100, the reward for a loss (opponent satisfies goal) was -100 and the reward for a draw (neither player satisfies goal) was 0.

Using these scenarios, we repeated the following procedure, which constitutes a single trial. First, we establish a performance benchmark by training a learner for 80 matches in the target domain. We call this the non-transfer learner. Next, we train a separate (transfer) learner for 80 matches in the source game, and 80 matches in the target. After each training match in the target game, both the non-transfer and transfer learners play a test match against a fixed-policy opponent to establish a performance level. We repeat this procedure 25 times, and average the results to produce

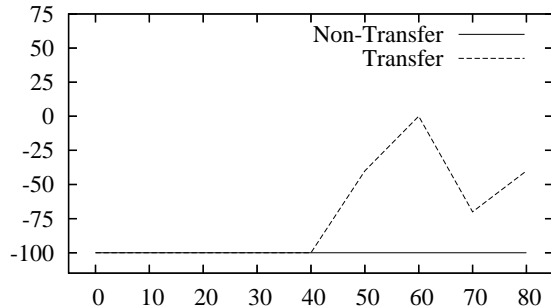


Figure 2. Learning curves for the KRK target.

a learning curve for the scenario.

3.3. Results

The first scenario (KQKQP) requires white, playing with a king and queen, to checkmate black, playing with a king, queen and pawn. Since black is ahead in the game (up by a pawn), white must find a sequence of moves which forces black’s hand. Otherwise, black will win. The target game in this scenario has the same goals as the source, but expands the powers of both queens, and adds an extra pawn for black. Note that according to the source game description, the solution used in the source cannot be used in the target as the added pawn blocks the necessary moves.

Figure 1 shows the learning curves for both the transfer and non-transfer learning agents in the target game. The non-transfer learner requires approximately 20 self-play training matches to find the solution to the problem. The transfer learner plays correctly immediately, suggesting that no changes to its state evaluation function were required. This demonstrates that the relational representation used by rTD(λ) is sufficiently abstract to assimilate the changes in rules between the source and target games.

The second scenario is a new spin on an well known chess end-game. The KRK endgame requires white to checkmate black, who has a king only, using a king and rook. In this case, the source is played on a small (5×5) board, while the target is played on a standard (8×8) chess board. Although the techniques are fundamentally the same, the difference in board size changes the identity of cells critical to executing the strategy.

Figure 2 shows the learning curves for both agents. Unlike the first scenario, in which initial games were drawn and both the transfer and non-transfer learners acquired the knowledge necessary to win, neither agent learned to win regularly. However, the agent without transfer did not manage to win a single match in the

target game, while the agent with transfer did ultimately improve performance and win several matches.

This is an example of a problem in which transfer is *required* for any useful learning to occur in the target. We believe that such cases represent the strongest argument in favor of research into knowledge transfer. Transfer of knowledge from one problem to another not only simplifies the learning process, as in the first scenario presented above, but also enables learning in other unfathomable domains, as in the second scenario.

4. Future Work

The work presented here demonstrates that reinforcement learning combined with a relational representation is capable of transferring knowledge from one domain to another. However, the assumption that the necessary concepts are always available to the learner is very strong. In the long term, methods that automatically construct the high-level predicates necessary to encode relational value functions are required. Our relational language provides a suitable formalization for this problem.

Several approaches hold promise for this problem. For example, relational predicates may be generated from the symbolic description of the domain, as demonstrated by Fawcett and Utgoff (1992). Statistical approaches to discovering emerging patterns have also been considered (Dong & Li, 1999). Another possibility is to use methods similar to explanation-based learning to find useful predicates from observed expert traces (Nejati et al., 2006). Confidence measures may also be useful in selecting states that require additional predicates to give more accurate value estimates.

A second line of research takes a longer view on the transfer learning problem. Stracuzzi (2005) considers the problem of memory organization. Here the learner is tasked with acquiring many concepts, and has many previous experiences to draw upon. The result is that explicitly considering each previous experience as a possible source for transfer becomes intractable. The learner is forced to organize its knowledge to maintain efficiency throughout the learning process.

5. Conclusion

The rTD(λ) framework presents a flexible method for transferring knowledge among related domains. Of particular importance is the use of a relational representation for both the state and the value function. The relational concept hierarchy provides the layers

of abstraction necessary to make transfer of knowledge structures feasible. The next step toward creating a robust transfer learning system is to incorporate methods for the automatic construction of the concept hierarchies. This will improve further the flexibility of the transfer learning system.

Acknowledgments

This material is based on research sponsored by DARPA under agreement FA8750-05-2-0283. The U. S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of DARPA or the U. S. Government.

References

- Dong, G., & Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining* (pp. 43–52). San Diego, CA.
- Dzeroski, S., Raedt, L. D., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, 43, 7–52.
- Fawcett, T., & Utgoff, P. E. (1992). Automatic feature generation for problem solving systems. *Proceedings of the Ninth International Workshop on Machine Learning* (pp. 144–153). Aberdeen, Scotland.
- Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 157–163). New Brunswick, NJ: Morgan Kaufmann.
- Nejati, N., Langley, P. W., & Konik, T. (2006). Learning hierarchical task networks by observation. *Proceedings of the 23rd International Conference on Machine Learning*. Pittsburgh, PA.
- Stracuzzi, D. J. (2005). Scalable knowledge acquisition through memory organization. *International and Interdisciplinary Conference on Knowledge Representation and Reasoning* (pp. 57–64). Espoo, Finland: Helsinki University of Technology.
- Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, 3, 9–44.