Beyond Simple Rule Extraction: Acquiring Planning Knowledge from Neural Networks

Ron Sun
CECS Department, University of Missouri
Columbia, MO, USA

Todd Peterson
Computer Science Department, Brigham Young University
Provo, Utah, USA

Chad Sessions
Computer Science Department, University of Alabama
Tuscaloosa, AL, USA

Abstract

This paper discusses learning in hybrid models that goes beyond simple classification rule extraction from backpropagation networks. Although simple rule extraction has received a lot of research attention, we need to further develop hybrid learning models that learn autonomously and acquire both symbolic and subsymbolic knowledge. It is also necessary to study autonomous learning of both subsymbolic and symbolic knowledge in integrated architectures. This paper will describe planning knowledge extraction from neural reinforcement learning that goes beyond extracting simple rules. It includes two approaches towards extracting planning knowledge: the extraction of symbolic rules from neural reinforcement learning, and the extraction of complete plans. This work points to a general framework for achieving the subsymbolic to symbolic transition in an integrated autonomous learning framework.

1 Introduction

This paper will discuss learning in hybrid models that goes beyond simple classification rule extraction from backpropagation networks. Although simple rule extraction has received a lot of research attention (Fu 1991, Towell and Shavlik 1993, Tickle et al 2000), we need to further develop hybrid learning models that learn autonomously and acquire both symbolic and subsymbolic knowledge. It is necessary to study autonomous learning of both subsymbolic and symbolic knowledge from scratch, without requiring a priori domain knowledge to begin with, in integrated architectures. In this regard, this paper will describe planning knowledge extraction from neural reinforcement learning (see Sun 1997, Sun and Peterson 1998, Sun et al 2000, Sun and Sessions 1998). It includes two approaches towards extracting planning knowledge: the extraction of explicit,

symbolic, planning rules from neural reinforcement learners, and the extraction of complete explicit plans from such learners. The advantages of symbolic knowledge extraction include (1) the improvement of learning (especially with the rule extraction approach), and (2) the improvement of the usability of results of neural reinforcement learning. Theoretically speaking, this work points to a general framework for achieving the subsymbolic to symbolic transition in an integrated autonomous learning framework (Sun 2000).

2 Hybrid Learning through Knowledge Extraction

We will first review reinforcement learning. Then we will detail rule extraction and plan extraction in turn.

2.1 Sequential Decision Tasks and Reinforcement Learning

In sequential decision tasks, an agent needs to perform a sequence of actions to reach some goal states. It learns autonomously to perform such tasks, from scratch, without teacher input. The characteristics of such a setting, which is the most essential in the real world, include: autonomous and on-line learning, non-stationary environments (revision of knowledge), no external teacher input (no known correct output), and so on.

One possible algorithm for dealing with this type of setting is Q-learning (Watkins 1989). A Q-value is an evaluation of the "quality" of an action in a given state: $Q(x, a)$ indicates how desirable action a is in state x (which consists of sensory input). One easy way of choosing an action is to choose the one that maximizes the Q-value in the current state; that is, we choose a if $Q(x, a) = \max_i Q(x, i)$. To ensure adequate exploration, a stochastic decision process, for example, based on Boltzmann distribution, can be used, so that different actions can be tried in accordance with their respective probabilities to ensure various possibilities are all looked into. That is,

$$p(a|x) = \frac{e^{Q(x,a)/\alpha}}{\sum_i e^{Q(x,a_i)/\alpha}}$$

Here α controls the degree of randomness (temperature) of the decision-making process.

To acquire the Q-values, we use the *Q-learning* algorithm (Watkins 1989). In the algorithm, $Q(x, a)$ estimates the maximum discounted cumulative reinforcement that the agent will receive from the current state x on:

$$\max(\sum_{i=0}^{\infty} \gamma^i r_i),$$

where γ is a discount factor that favors reinforcement received sooner relative to that received later, and r_i is the reinforcement received at step i (which may be 0). The updating of $Q(x, a)$ is based on minimizing

$$r + \gamma e(y) - Q(x, a)$$

where γ is a discount factor, $e(y) = \max_a Q(y, a)$, and y is the new state resulting from action a. Thus, the updating is based on the *temporal difference* in evaluating the current state and the action chosen. Through successive updates of the Q function, the agent can learn to take into account future steps in longer and longer sequences notably without explicit planning (Watkins 1989). The agent may eventually converge to a stable function and find an optimal sequence that maximizes the reinforcement received. Hence, the agent learns to deal with sequential decision tasks, in an autonomous, on-line fashion, without requiring any a priori domain-specific knowledge. Note that Q-learning can be implemented in a backpropagation network.

2.2 Rule Extraction

A two-level model (the CLARION model) was developed for extracting rules and using them along with neural reinforcement learning (Sun 1997, Sun and Peterson 1997, 1998, Sun et al 2000). In the model, the bottom level is a backprop net implementing Q-learning (Watkins 1989, Lin 1992, Tesauro 1992) and the top level is a localist rule net (Sun 1992; see details later).

2.2.1 The Algorithm

A high-level pseudo-code algorithm that describes the operation of the model is as follows:

1. Observe the current state x.
2. Compute in the bottom level the Q-values of x associated with each of all the possible actions a_i's: $Q(x, a_1)$, $Q(x, a_2)$, ..., $Q(x, a_n)$.
3. Find out all the possible actions (b_1, b_2, ..., b_m) at the top level, based on the input x and the rules in place.
4. Compare the values of a_i's with those of b_j's, and choose an appropriate action b
5. Perform the action b, and observe the next state y and (possibly) the reinforcement r.
6. Update the bottom level in accordance with *Q-Learning*.
7. Update the top level with *Rule-Extraction-Revision*.
8. Go back to Step 1.

The Bottom Level. It implements Q-learning. To implement Q-learning, we chose to use a four-layered network, in which the first three layers form a backpropagation network for computing Q-values and the fourth layer (with only one node) performs stochastic decision making. The output of the third layer (i.e., the output layer of the backpropagation network) indicates the Q-value of each action (represented by an individual node), and the node in the

fourth layer determines probabilistically the action to be performed based on a Boltzmann distribution (Watkins 1989).

The learning of the network is based on minimizing the following:

$$err_i = \begin{cases} r + \gamma e(y) - Q(x,a) & \text{if } a_i = a \\ 0 & \text{otherwise} \end{cases}$$

where i is the index for an output node representing the action a_i.

Q-learning allows autonomous learning (without any a priori domain knowledge) of closed-loop action policies in sequential decision tasks, from which symbolic rules and plans can be extracted. The use of neural networks in implementing Q-learning is due to the need to deal with huge state spaces in many tasks (Lin 1992, Tesauro 1992, Sutton 1996).

The Top Level. We devised a novel rule learning algorithm based on neural reinforcement learning. The basic idea is as follows: the agent performs rule learning (extraction and subsequent revision) at each step, which is associated with the following information: (x, y, r, a), where x is the state before action a is performed, y is the new state entered after an action a is performed, and r is the reinforcement received after action a. If some action decided by the bottom level is successful then the agent extracts a rule that corresponds to the decision and adds the rule to the rule network. Then, in subsequent interactions with the world, the agent verifies the extracted rule by considering the outcome of applying the rule: if the outcome is not successful, then the rule should be made more specific and exclusive of the current case; if the outcome is successful, the agent may try to generalize the rule to make it more universal. Rules are in the following form: *conditions* $\longrightarrow$ *action*, where the left-hand side is a conjunction of individual conditions each of which refers to a primitive: a value range or a value in a dimension of the (sensory) input state.

At each step, the agent updates the following statistics for each rule condition and each of its minor variations (i.e., the rule condition plus/minus one value), with regard to the action a performed: that is, PM_a (i.e., Positive Match) and NM_a (i.e., Negative Match). Here, positivity/negativity is determined by the following inequality: $\max_b Q(y,b) - Q(x,a) + r > threshold$, which indicates whether or not the action is reasonably good. Based on these statistics, the information gain measure is calculated; that is,

$$IG(A,B) = log_2 \frac{PM_a(A)+1}{PM_a(A)+NM_a(A)+2} - log_2 \frac{PM_a(B)+1}{PM_a(B)+NM_a(B)+2}$$

where A and B are two different conditions that lead to the same action a. The measure compares essentially the percentage of positive matches under different conditions A and B (with the Laplace estimator; Lavrac and Dzeroski 1994). If A can improve the percentage to a certain degree over B, then A is considered better than B. In the algorithm, if a rule is better compared with the match-all rule (i.e, the rule with the condition that matches all inputs), then the rule is considered successful (for the purpose of deciding on expansion or shrinking operations).

The agent decides on whether or not to extract a rule based on a simple success criterion which is fully determined by the current step (x, y, r, a):

- *Extraction*: if $r + \gamma e(y) - Q(x, a) > threshold$, where a is the action performed in state x and y is the resulting new state (that is, if the current step is successful), and if there is no rule that covers this step in the top level, set up a rule $C \longrightarrow a$, where C specifies the values of all the input dimensions exactly as in x.

The criterion for applying the *expansion* and *shrinking* operators, on the other hand, is based on the afore-mentioned statistical test. Expansion amounts to adding an additional value to one input dimension in the condition of a rule, so that the rule will have more opportunities of matching inputs, and shrinking amounts to removing one value from one input dimension in the condition of a rule, so that it will have less opportunities of matching inputs. Here are the detailed descriptions of these operators:

- *Expansion*: if $IG(C, all) > threshold1$ and $\max_{C'} IG(C', C) \geq 0$, where C is the current condition of a matching rule, *all* refers to the match-all rule (with regard to the same action specified by the rule), and C' is a modified condition such that C' = C plus one value (i.e., C' has one more value in one of the input dimensions) (that is, if the current rule is successful and the expanded condition is potentially better), then set $C'' = argmax_{C'} IG(C', C)$ as the new (expanded) condition of the rule.

- *Shrinking*: if $IG(C, all) < threshold2$ and $\max_{C'} IG(C', C) > 0$, where C is the current condition of a matching rule, *all* refers to the match-all rule (with regard to the same action specified by the rule), and C' is a modified condition such that C' = C minus one value (i.e., C' has one less value in one of the input dimensions) (that is, if the current rule is unsuccessful, but the shrunk condition is better), then set $C'' = argmax_{C'} IG(C', C)$ as the new (shrunk) condition of the rule. If shrinking the condition makes it impossible for a rule to match any input state, delete the rule.

The Combination. In the overall algorithm, Step 4 is for making the final decision of which action to take by incorporating outcomes from both levels. The agent combines the corresponding values for an action from the two levels by a weighted sum; that is, if the top level indicates that action a has an activation value v (which should be 0 or 1 as rules are binary) and the bottom level indicates that a has an activation value q (the Q-value), then the final outcome is $w_1 * v + w_2 * q$. Stochastic decision making with Boltzmann distribution based on the weighted sums is then performed to select an action out of all the possible actions. Relative weights or percentages of the two levels are automatically set based on the relative performance of the two levels (Sun and Peterson 1998).

2.2.2 Advantages of Rule Extraction

Extracting rules has some advantages that make it indispensable:

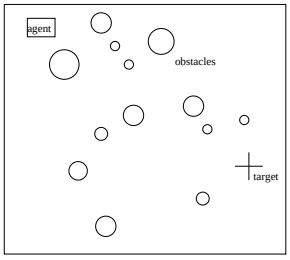

Figure 1: Navigation through a minefield

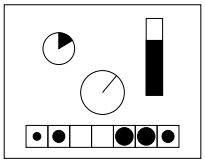

Figure 2: The Navigation Input
The display at the upper left corner is the fuel gauge; the vertical one at the upper right corner is the range gauge; the round one in the middle is the bearing gauge; the 7 sonar gauges are at the bottom.

- Extracted rules can help to speed up learning. If properly used, rules that are extracted on-line during learning can help to facilitate the learning process itself.
- Rules can help to improve the performance of learned systems.
- Rules help to guide the exploration of new situations, and reduce the time necessary to learn in new situations.
- Rules can also help in comprehending and communicating learned skills.

Such advantages have been demonstrated (Sun and Peterson 1998).

For example, we tested CLARION on the simulated navigation task as shown in Figure 1. The agent has to navigate an underwater vessel through a minefield to reach a target location. The agent receives information only from a number of instruments. As shown in Figure 2, the sonar gauge shows how close the mines are in seven equal areas that range from 45 degrees to the left of the agent to 45 degrees to the right. The fuel gauge shows how much time is left before fuel runs out. The bearing gauge shows the direction of the target from the present heading of the agent. The range gauge shows how far the target is from the current location. Based only on such information, the agent decides on (1) how to turn and (2) how fast to move. The time allotted to the agent for each episode is 200 steps. The agent, within an allotted time period, can either (1) reach the target (a success), (2) hit a mine (a failure), or (3) run out of time (a failure).

Mine Density During Training: 10

	Train	B+T	B	T	B+T	B	T	B+T	B	T
	10	10	10	10	30	30	30	60	60	60
CLARION	651.8	63.5	6.5	4.5	35.5	1.5	0.5	11.5	1.0	0.0
s.d.	31.3	34.4	4.5	6.9	21.0	2.3	1.5	9.5	2.0	0.0
Q	645.7		82.0			42.0			14.5	
s.d.	86.9		14.2			24.5			18.1	

Figure 3: Learning and transfer from 10-mine minefields

Q refers to the bottom level used alone with Q-learning as the sole learning method. *Train* indicates the total numbers of successful episodes during training. The next three blocks contain performance data (in percentage), in three different mine densities (10, 30, and 60) using the trained models with either the top level, the bottom level, or both together.

Learning speed. Figures 3, 4, and 5 show the data of CLARION. In terms of learning effectiveness, which is measured by the number of successful episodes out of a total of 1000 episodes of training (averaged over 10 runs), the "train" columns of these figures show the difference between CLARION and the bottom-level neural net alone (trained with only Q-learning). It appears that at higher mine densities (that is, in the more difficult settings), CLARION performed significantly better compared with the bottom-level neural net alone. In the 30-mine and 60-mine cases, the superiority of CLARION (over the neural net alone with Q-learning) is statistically significant (with t tests, $p < 0.05$).

Transfer. The right three blocks of Figures 3, 4, and 5 show the transfer data, where transfer is measured by the percentage of successful episodes in new settings by the trained models (each trained model is applied to minefields that contain a *different* number of mines for a total of 20 episodes; the data is averaged over 10 runs). As indicated by the tables, CLARION outperforms the bottom-level neural net alone (trained with Q-learning) in transfer at higher mine densities; the higher the mine density, the more pronounced the difference is. The differences are statistically significant in the 30-mine and 60-mine cases (using t tests, $p < 0.05$). Finally, comparing the transfer performance of the top level, the bottom level, and the whole system, after they are trained together, we notice that the whole system always performs much better than either level alone. Learning rules does help to improve transfer performance.

Trained performance. The right three blocks of Figures 3, 4, and 5 also contain the trained performance data. Trained performance is defined to be the percentage of successful episodes in the *same* settings as used in training by the trained models (each trained model is tested for a total of 20 episodes; the data is averaged over 10 runs). At higher mine densities, the trained performance of CLARION is better than the bottom-level neural net alone (trained with Q-learning). Comparing the performance of the whole system and the two levels separately after the two levels are trained together, we again notice that the whole system performs much better than the bottom level and the top level alone. The differences are statistically significant.

Rules Extracted. Here is a small subset of rules to give an idea:

```
Bearing: straight ahead
LeastDense: center and right
FurthestMine: center
```

Mine Density During Training: 30

	Train	B+T	B	T	B+T	B	T	B+T	B	T
	30	10	10	10	30	30	30	60	60	60
CLARION	663.8	89.0	5.0	1.0	75.0	7.0	0.0	47.5	2.5	0.0
s.d.	48.4	26.5	3.2	2.0	23.6	5.6	0.0	24.9	2.5	0.0
Q	539.1		77.0			68.0			35.5	
s.d.	105.6		17.5			20.4			20.7	

Figure 4: Learning and transfer from 30-mine minefields

Mine Density During Training: 60

	Train	B+T	B	T	B+T	B	T	B+T	B	T
	60	10	10	10	30	30	30	60	60	60
CLARION	581.4	99.5	9.5	2.0	96.0	8.5	0.0	76.0	6.0	0.0
s.d.	79.0	1.5	6.5	3.3	3.7	4.5	0.0	15.9	6.6	0.0
Q	495.8		71.5			67.5			47.5	
s.d.	137.9		11.6			16.8			24.3	

Figure 5: Learning and transfer from 60-mine minefields

```
LeftAvgMineDistance: close
CenterAvgMineDistance: very far
RightAvgMineDistance: very far
Direction: go straight, Speed: very fast

Bearing: far right
LeastDense: left
FurthestMine: left
LeftAvgMineDistance: far
CenterAvgMineDistance: close
RightAvgMineDistance: very close
Direction: turn left, Speed: very fast

Bearing: straight ahead, right, far right, right behind
LeastDense: right
FurthestMine: right
LeftAvgMineDistance: very close or close
CenterAvgMineDistance: close
RightAvgMineDistance: far
Direction: turn right, Speed: very fast
```

2.3 Plan Extraction

The above rule extraction method generates isolated rules, bearing in mind individual states, but not the chaining of these rules in accomplishing a sequential decision task. In contrast, the following plan extraction method generates a complete explicit plan that can by itself accomplish a sequential task. By an *explicit plan*, we mean an action policy consisting of an explicit sequence of action steps, that does not require (or requires little) environmental feedback during execution (compared with a completely closed-loop action policy as generated by Q-learning).

2.3.1 The Algorithm

The PlanExtraction algorithm (Sun and Sessions 1998) turns a set of Q-values (and the corresponding closed-loop policy resulting from these values) into a plan that is in the form of a sequence of steps (in accordance with the traditional AI formulation of planning). The basic idea is that we use beam search, to find the best action sequences (or conditional action sequences) that achieve the goal with a certain probability. The search is based on the insight that the optimal Q-value learned through Q-learning represents the total future probability of

reaching the goal (Sun and Sessions 1998). Thus Q-values can be used as a guide in searching for explicit plans.

The algorithm employs the following data structures for plan extraction. The *current state set, CSS*, consists of multiple pairs in the form of $(s, p(s))$, in which the first item indicates a state s and the second item $p(s)$ indicates the probability of that state. For each state in CSS, we find the corresponding best action. In so doing, we have to limit the number of branches at each step, for the sake of time efficiency of the algorithm as well as the representational efficiency of the resulting plan. The set thus contains up to (a fixed number) n pairs, where n is the branching factor in beam search. In order to calculate the best default action at each step, we include a second set of states CSS', which covers a certain number (m) of possible states not covered by CSS.

Set the current state set $CSS = \{(s_0, 1)\}$ and $CSS' = \{\}$
Repeat until the termination conditions are satisfied (e.g., $step > D$)
- For each action u, compute the probabilities of transitioning to each of all the possible next states (for all $s' \in S$) from each of the current states ($s \in CSS$): $p(s', s, u) = p(s) * p_{s,s'}(u)$
- For each action u, compute its estimated utility with respect to each state in CSS: $Ut(s, u) = \sum_{s'} p(s', s, u) * \max_v Q(s', v)$. That is, we calculate the *probabilities* of reaching the goal after performing action u from the current state s.
- For each action u, compute the estimated utility with respect to *all* the states in CSS': $Ut(CSS', u) = \sum_{s \in CSS'} \sum_{s'} p(s) * p_{s,s'}(u) \max_v Q(s', v)$
- For each state s in CSS, choose the action u_s with the highest utility $Ut(s, u)$: $u_s = argmax_u Ut(s, u)$
- Choose the best default action u with regard to all the states in CSS': $u = argmax_{u'} Ut(CSS', u')$
- Update CSS to contain n states that have the highest n probabilities, i.e., with the highest $p(s')$'s: $p(s') = \sum_{s \in CSS} p(s', s, u_s)$, where u_s is the action chosen for state s.
- Update CSS' to contain m states that have the highest m probabilities calculated as follows, among those states that are not in the new (updated) CSS: $p(s') = \sum_{s \in CSS \cup CSS'} p(s', s, u_s)$, where u_s is the action chosen for state s (either a conditional action in case $s \in CSS$ or a default action in case $s \in CSS'$), and the summations are over the old CSS and CSS' (before updating). [1]

In the measure Ut, the agent takes into account the probabilities of reaching the goal in the future from the current states (based on the Q-values; see Theorem 1 in Sun and Sessions 1998), as well as the probability of reaching the current states based on the history of the paths traversed (based on $p(s)$'s). This is because what we are aiming at is the estimate of the overall success probability of a path. The basic idea, combining measures of past history and future possibilities, is essentially the same as the A* algorithm. However, instead of an additive combination, we use a multiplicative combination, because

[1] For both CSS and CSS', if a goal state or a state of probability 0 is selected, we may remove it and, optionally, reduce the beam width of the corresponding set by 1.

Action	Description
0-	left
1-	slightly left
2-	straight
3-	slightly right
4-	right
-5	0 knots
-6	10 knots
-7	20 knots
-8	30 knots
-9	40 knots

Figure 6: The possible actions in the Navigation task.

probabilities require such a combination. In the algorithm, the agent selects the best actions (for $s \in CSS$ and CSS') that are most likely to succeed based on these measures.

2.3.2 Advantages of Plan Extraction

The advantages of plan extraction lie in improving the usability of results and in compressing action policies. Instead of closed-loop policies resulting from Q-learning that have to rely on moment-to-moment sensing, some extracted plans can be run with little (or even no) sensing of the environment, and thus they are useful in situations where there is little or no environmental feedback, or where there is only unreliable feedback. Even when reliable feedback is available, they are beneficial in terms of saving sensing costs. Furthermore, usually extracted plans are much smaller than the original action policies represented by Q-values; that is, they achieve policy compression. Thus, extracting plans also saves representational cost (Sun and Sessions 1998).

Minefield navigation. An an example, in this task, based on the results of Q-learning, plans were extracted. One example plan is shown in Figure 7. The plan indicates what to do in different circumstances, that is, a strategy for navigating through the minefield that avoids mines and goes in the general direction of the target. The plan is basically a sequence of zigzagging actions. The plan is much smaller than the corresponding action policy in the form Q-values (which has 10^{12} entries) from which the plan was extracted, and thus achieves policy compression.

Tower of Hanoi. Another example is the Tower of Hanoi task, which produces easily interpretable plans and is thus used here for illustration. It consists of three pegs, with peg 1 starting with the three discs. The goal of the task is to get the three discs from peg 1 to peg 3. Only the top disc on a peg can be moved. The three discs are of three different sizes and a larger disc may not rest on a smaller disc. There are six possible actions (as shown in Figure 8), although at most only three can be performed at each step (due to the relative size constraint). These moves generate 27 allowable states (see Figure 9).

Q-learning was applied to learn an action policy. Plan extraction was then applied. The plan steps are as follows:

> Step 1: moving the top disc from peg 1 to 3 (action 1). Step 2: from peg 1 to 2 (action 0). Step 3: from peg 3 to 2 (action 5). Step

```
Step 1: [(2, 9)]
Step 2: [(6, 49) (3, 49) (27, 9) (24, 9) (33, 49)]
Step 3: [(19, 9) (2, 9) (4 49)]
Step 4: [(6, 49) (5, 9) (3, 49) (24, 9)]
Step 5: [(19, 9) (6, 49) (4, 49) (2, 9)]
Step 6: [(19, 9) (6, 49) (5, 9) (3, 49)]
Step 7: [(6, 49) (19, 9) (4, 49)]
Step 8: [(19, 9) (6, 49) (5, 9)]
Step 9: [(6, 49) (19, 9)]
Step 10: [(19, 9) (6, 49)]
Step 11: [(6, 49) (19, 9)]
Step 12: [(19, 9) (6, 49)]
Step 13: [(6, 49) (19, 9)]
Step 14: [(19, 9)]
Step 15: [(6, 49)]
Step 16: [(19, 9)]
```

Figure 7: A conditional plan extracted for the Navigation task. $n = 5, m = 8$. In each pair of parentheses, the first number is an (aggregated) state, and the second is an action. Actions are numbered as indicated in Figure 6. No default action has a utility value above zero and thus none is included. The plan is a sequence of zigzagging actions for avoiding mines.

Action	Description
A0	Peg 1 to Peg 2
A1	Peg 1 to Peg 3
A2	Peg 2 to Peg 1
A3	Peg 2 to Peg 3
A4	Peg 3 to Peg 1
A5	Peg 3 to Peg 2

Figure 8: The possible actions in the Tower of Hanoi task

4: from peg 1 to 3 (action 1). Step 5: from peg 2 to 1 (action 2). Step 6: from peg 2 to 3 (action 3). Step 7: from peg 1 to 3 (action 1).

The goal can be reached by an extracted plan with a 100% probability. The plan is much smaller than the corresponding action policy in the form of Q-values (which requires at least 54 entries), and achieves policy compression. This plan does not require step-by-step sensing of the environment, and thus plan extraction enhances usability as well.

3 Concluding Remarks

Both subsymbolic and symbolic knowledge are needed for intelligent agents. They both can and should be learned on-line, autonomously, starting with no a priori domain knowledge, although different algorithms may be used to learn them. Hence, we need to explore hybrid learning frameworks for agents. In accordance with the characteristics of autonomous, sequential learning agents (Sun 2000, Sun et al 2000), we developed learning methods that perform neural reinforcement learning first and extraction of symbolic rules and plans later on its basis, in an integrated hybrid learning framework. The extraction of symbolic planning knowledge enhances the overall hybrid system in various ways.

State Number	Description
0	(123,0,0)
1	(23,1,0)
2	(23,0,1)
3	(13,2,0)
4	(13,0,2)
5	(12,3,0)
6	(12,0,3)
7	(3,12,0)
8	(3,0,12)
9	(3,1,2)
10	(3,2,1)
11	(2,13,0)
12	(2,0,13)
13	(2,1,3)
14	(2,3,1)
15	(1,23,0)
16	(1,0,23)
17	(1,2,3)
18	(1,3,2)
19	(0,123,0)
20	(0,23,1)
21	(0,13,2)
22	(0,12,3)
23	(0,3,12)
24	(0,2,13)
25	(0,1,23)
26	(0,0,123)

Figure 9: The possible states in the Tower of Hanoi task
State 0 is the starting state. State 26 is the goal state. In the description, pegs are separated by commas. "1" indicates the smallest disc, "2" the median-sized, and "3" the largest.

Acknowledgements: This work was supported in part by Office of Naval Research grant N00014-95-1-0440.

References

L.M. Fu, (1991). Rule learning by searching on adapted nets, *Proc.of AAAI'91*, pp.590-595.

N. Lavrac and S. Dzeroski, (1994). *Inductive Logic Programming.* Ellis Horword, New York.

L. Lin, (1992). Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning.* Vol.8, pp.293-321.

R. Maclin and J. Shavlik, (1994). Incorporating advice into agents that learn from reinforcements. *Proc.of AAAI-94.* Morgan Kaufmann, San Meteo, CA.

R. Sun, (1992). On variable binding in connectionist networks. *Connection Science*, Vol.4, No.2, pp.93-124. 1992.

R. Sun, (1997). Learning, action, and consciousness: a hybrid approach towards modeling consciousness. *Neural Networks*, 10 (7), pp.1317-1331

R. Sun, (2000). Symbol grounding: a new look at an old idea. *Philosophical Psychology*, Vol.13, No.2, pp.149-172.

R. Sun, E. Merrill, and T. Peterson, (2000). From implicit skills to explicit knowledge: a bottom-up model of skill learning. *Cognitive Science.* in press.

R. Sun and T. Peterson, (1997). A hybrid model for learning sequential navigation. *Proc. of IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA'97).* Monterey, CA. pp.234-239. IEEE Press.

R. Sun and T. Peterson, (1998). Autonomous learning of sequential tasks: experiments and analyses. *IEEE Transactions on Neural Networks*, Vol.9, No.6, pp.1217-1234.

R. Sun and C. Sessions, (1998). Extracting plans from reinforcement learners. *Proceedings of the 1998 International Symposium on Intelligent Data Engineering and Learning (IDEAL'98)*. pp.243-248. eds. L. Xu, L. Chan, I. King, and A. Fu. Springer-Verlag.

R. Sutton, (1996). Generalization in reinforcement learning. *Advances in Neural Information Processing Systems 8*. MIT Press. Cambridge, MA.

T. Tesauro, (1992). Practical issues in temporal difference learning. *Machine Learning*. Vol.8, 257-277.

A. Tickle, J. Diederich, et al, (2000). Lessons from past, current issues, and future research directions in extracting knowledge embedded in artificial neural networks. In: S. Wermter and R. Sun, (eds.) *Hybrid Neural Systems*, Springer-Verlag, Berlin.

G. Towell and J. Shavlik, (1993). Extracting refined rules from Knowledge-Based Neural Networks, *Machine Learning*. 13 (1), 71-101.

C. Watkins, (1989). *Learning with Delayed Rewards*. Ph.D Thesis, Cambridge University, Cambridge, UK.