A Hybrid Architecture for Learning Reactive Sequential Decision Making

Ron Sun, Todd Peterson, Edward Merrill
The University of Alabama
Tuscaloosa, AL 35487
rsun@cs.ua.edu

Abstract
This paper introduces a hybrid model that combines connectionist, symbolic, and reinforcement learning for tackling reactive sequential decision tasks by a situated agent. Both procedural skills and high-level symbolic representations are acquired through an agent’s experience interacting with the world, in a bottom-up direction. It deals with on-line learning, that is, learning continuously from on-going experience in the world, without the use of preconstructed data sets or preconceived concepts. The model is a connectionist one based on a two-level approach proposed earlier.

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

Introduction
Reactive sequential decision tasks involve selecting and performing a sequence of actions, in order to accomplish an objective, mostly on the basis of moment-to-moment perceptual information. One example is learning to navigate in a maze like the one shown in Figure 1. Another example involves learning to navigate through mines to reach a target (see Figure 2).

A general specification is as follows: there is an agent that can select, from a finite set of actions, a particular action to perform at each time step (in either a discrete or a continuous sense). The selection decision is (mainly) based on the current state of the world. The world is presented to the agent, through sensory input, as a state vector which contains various pieces of information (with some correlated and some uncorrelated). The world changes either autonomously or as a result of some action by an agent. Thus, over time, the world is presented to an agent as a sequence of states. At certain points in a sequence, the agent may receive payoffs or reinforcements for their actions performed prior to the current state. Thus, the agent may need to perform temporal and structural credit assignment, to attribute the payoffs/reinforcements to various actions at various points in time (that is, the temporal credit assignment problem) in accordance to various aspects of a state (that is, the structural credit assignment problem).

While performing this kind of task, the agent is often under severe time pressure. Often a decision has to be made in a fraction of a second; therefore it cannot do much “information processing”, and falls outside of Allen Newell’s “rational band”. The decision making and learning in the agent thus cannot be too
time-consuming. As in humans, the agent may also be severely limited in other resources, such as memory, so that memorizing all the previous episodes is considered impossible. The perceptual ability of an agent may also be extremely limited so that only very local information is available. Learning in such a domain is an experiential, trial-and-error process; the agent develops knowledge tentatively on an on-going basis, since it cannot wait until the end of an episode. Learning is concurrent or on-line (Nosofsky et al. 1994).

The characteristics of a world need not be stationary from the view point of the learning agent. The world can be nonstationary in several ways: (1) the world and its characteristics can change over time; thus, the revision of knowledge learned by an agent may be necessary (if changes are substantial enough). (2) Even when the world per se is stationary, to an agent it may still seem uncertain and evolving; for, in such an experience-driven learning, there is no preselected set of (positive and negative) instances that provide a clear view of the world; different regions of the world may exhibit different characteristics and thus require the revision of concepts over time. (3) Once a knowledge structure is revised, the agent has to view whatever it experienced before in a new way (since knowledge provides a “filter” through which the agent sees the world), and thus the experience may seem different and the world nonstationary. (4) In an experience-driven learning situation, there is a lack of a clear and steady criterion for learning. Payoffs may be received sporadically, and it is thus up to the agent to decide what to make of them. The agent has to assign credits/blames on the basis of what is already known, which is changing itself. So the learning criterion is a moving target, and the learning process becomes nonstationary.

There are some existing methods for dealing with this type of task to a certain extent. Chief among them is the temporal difference method (Sutton 1988). Another approach, genetic algorithm (Holland et al. 1986), can also be used to tackle this kind of task (Grefenstette 1992, Schultz 1991). But they do not handle different types of knowledge as will be discussed next.

Hybrid Models

How can an agent develop a set of coping skills that is highly specific (geared towards particular situations) and thus highly efficient but, at the same time, acquire sufficiently general knowledge that can be readily applied to a variety of different situations? Although humans seem to possess such abilities and be able to achieve an appropriate balance between the two, existing systems fall far short. What appears to be missing is the duality and coexistence of both procedural and declarative knowledge, or both subconceptual and conceptual knowledge. There has been a great deal of work demonstrating the difference between procedural (subconceptual) knowledge and declarative (conceptual) knowledge: e.g., Anderson (1982, 1990), Keil (1989), Damasio et al. (1990), and Sun (1994). It is believed by some researchers that a balance of the two is essential to the development of complex agents. This assumption is based on two lines of argument. First, there are ample psychological data that point to the distinction between the two types and the need for both. Anderson (1982) put forward the distinction between declarative and procedural knowledge based on such data; Fitts and Posner (1967), Keil (1989), and Sun (1994) made similar points. Second, there are also many philosophical arguments for making the distinction and achieving an appropriate balance of the two. Dreyfus and Dreyfus (1987) proposed the distinction of analytical and intuitive thinking; Smolensky (1988) proposed the distinction between conceptual (publicly accessible) and subconceptual processing; in addition, the distinction between conscious and subconscious processes, although controversial, is well known (cf. James 1890, Shiffrin and Schneider 1977). In the current context, one way to learn a sequential navigation task is through trial-and-error: repeated practice gradually gives rise to a set of procedural skills that deal specifically with the practiced situations and their minor variations. However, such skills may not be transferable to truly novel situations, since they are so embedded in specific contexts and tangled together. Generic declarative knowledge has the following three advantages: (1) It helps to guide the exploration of novel situations and reduces the time (i.e., the number of trials) necessary to develop specific skills in new situations. In other words, it helps the transfer of learned skill (as shown in psychological data from Willingham et al 1989). (2) Generic knowledge can help to speed up learning. If properly used, generic knowledge that is extracted on-line during learning can help to facilitate the very learning process itself. (3) Generic knowledge can also help in communicating learned knowledge and skills to other agents.

A two-level hybrid models seem to provide the needed framework for representing both types of knowledge. There have been various two-level architectures proposed, such as Hendler (1987), Gelfand et al. (1989), Schneider and Oliver (1991), and Sun (1994). Based on the ideas proposed in Sun (1994), we developed CLARION (see Figure 3). The bottom level contains reactive routines or specific procedural knowledge (Anderson 1983). The top level contains rules, or generic declarative knowledge. An overall pseudo-code algorithm is as follows:

1. Observe the current state $x$ (in a proper representation).
2. Compute in the bottom level the Q-values of $x$ assci-
ated with each of all the possible actions: $Q(x, a_1)$, $Q(x, a_2)$, ..., $Q(x, a_n)$.

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 $Q$-values in accordance with the Q-learning algorithm.

7. Update the rule network in the top level using the Rule-Extract-Action-Generalization-Revision.

8. Go back to Step 1.

First of all, in terms of representation in the bottom level,\(^1\) we prefer a subsymbolic distributed representation, such as that provided by a backpropagation network. This is because of the automatized nature of procedural skills: there is generally a lack of conceptual thinking in performing procedural skills; as a consequence, details of such skills are in general inaccessible to self-introspection (consciousness) (Anderson 1982, Ackerman 1991). A distributed representation naturally captures this property of procedural skills (Sun 1994), with representational units that are capable of accomplishing tasks but are in general uninterpretable and subsymbolic. (Otherwise, a symbolic representation may be used, but then we will have to artificially assume that these representations are not accessible, while some other similar representations are accessible — the distinction is arbitrary and not intrinsic to the media of representations; cf. Anderson 1990, Rosenbloom et al. 1991).

In terms of learning, we use reinforcement learning (the temporal difference method). 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 some sketchy sensory input). To acquire the $Q$-values, we use the $Q$-learning algorithm (Watkins 1989), a temporal difference reinforcement learning algorithm. In the algorithm, $Q(x, a)$ estimates the maximum discounted cumulative reinforcement that the agent will receive from the current state $x$ on: $\max \left( \sum_{i=0}^{\infty} \gamma^i r_i \right)$, where $\gamma$ 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 $r + \gamma \epsilon(y) - Q(x, a)$, where $\gamma$ is a discount factor and $\epsilon(y) = \max_a Q(y, a)$. Thus, the updating is based on the temporal difference in evaluating the current state and the action chosen. Using Q-learning allows sequential behavior to emerge in an agent. Through successive updates of the Q function, the agent can learn to take into account future steps in longer and longer sequences.

To combine Q-learning with connectionist representation, we 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): $p(a|x) = \sum_i e^{\frac{1}{\alpha}Q(x, a_i)}$, where $\alpha$ controls the degree of randomness (temperature) of the decision-making process. The training of the network is based on minimizing the temporal difference as specified before.

Second, in terms of representing (generic) declarative knowledge, we prefer a symbolic or localist representation, in which each unit has a clear conceptual meaning or interpretation, because declarative knowledge is highly accessible and inferences are performed explicitly at a conceptual level (Smolensky 1988, Sun 1994). Because of the reactive nature of the task domain, symbolic representations that we need are relatively simple. We will thus focus on propositional rules. We use a localist connectionist model for representing these rules to facilitate correspondence with the bottom level and to encourage uniformity and integration. Basically, we connect the nodes representing conditions of a rule to the node representing the conclusion. However, we need to wire up rules involving conjunctive conditions. For details, see Sun (1992).

There are a number of ways symbolic representation can come into being. In this work, because of the dynamic nature of reactive sequential decision tasks, we need to be able to dynamically acquire our representation and to modify our representation in subsequent encounters if necessary. We thus need a simple and efficient way of acquiring rules and other structures, suitable for our tasks. We can make use of the bottom level which is trained with reinforcement learning to per-

\(^1\)Existing evidence shows that the difference between the two levels lies mainly in their representations (Reber 1989).
form specific procedural skills by extracting information from the network (Towell and Shavlik 1993) and thereby forming and modifying explicit rules. The basic idea for rule learning is as follows: if some action decided by the reactive level is successful then there might be general knowledge that can be extracted; the agent extracts a rule that corresponds to the action selected by the reactive level and adds the rule to the rule network. In subsequent interactions with the world, the agent tries to verify 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.

Specifically, three different criteria can be used for extracting rules from the bottom level: (1) direct reinforcement received at a step, (2) temporal difference (as used in updating Q-values), and (3) maximum Q-values in a state. We adopt a three-phase approach here, with three criteria being successively applied in different phases. After a rule is extracted, generalization and revision operations are used to tune the rule:

- **Expansion**: the value range of a condition is expanded by one interval, when a rule is successfully applied according to the criterion in the current phase.
- **Shrinking**: when a rule leads to unsuccessful results as judged by the criterion in the current phase, we reduce the value ranges of some or all conditions (cf. Michalski et al 1986).
- **Deletion**: remove a rule from the rule network when a counter example to the original case from which the rule was extracted is encountered, according to the current-phase criterion.
- **Merge**: when the conditions of two rules are close enough, the two rules may be combined so that a more general rule can be produced.

A complete model is shown in Figure 4. The necessity of having a two-level architecture can be summed up as follows: (1) Without the bottom level, the agent will not be able to represent procedural skills sufficiently. Such skills may involve graded, uncertain, and inconsistent knowledge and autonomous stochastic exploration (with numeric calculation and probabilistic firing). Thus they may not be captured by simpler mechanisms (such as those in Anderson 1990). (2) Without learning in the bottom level, the agent will not be able to learn from experience, and therefore will not be able to dynamically acquire either procedural skill in the bottom level (cf. Brooks 1991), or rules in the top level (as in the current model). The bottom level also captures the gradual learning of skills, different from one-shot rule learning. (3) Without the top level, the agent will not be able to represent generic, easily accessible, and crisp knowledge and explicitly access and communicate that knowledge to other agents (i.e., the explanation capability, which is absent in e.g. Sutton 1990, Brooks 1991). When novel situations are encountered (Gordon and Grefenstette 1992) and/or when precision, crispness, consistency, and certainty are needed, declarative knowledge is preferred. Explicit access and explanation is also important in facilitating cooperation among agents. (4) Without rule learning, the agent will not be able to acquire dynamically conceptual knowledge for the top level, and therefore has to resort to mostly pre-wired and/or externally given knowledge in the top level, and the agent will not be able to quickly acquire crisp knowledge.

We try two different methods of combining outcomes from the two levels. In the percentage method, in $p$ percent of the steps, if there is at least one rule indicating a proper action in the current state, we use the outcome from the rule level; otherwise, we use the outcome of the bottom level (which is always available). In the stochastic method, we combine 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 $v$ and the bottom level indicates that $a$ has a value $q$ (the Q-value), then the final outcome is $w_1 * v + w_2 * q$. Stochastic decision making with Boltzmann distribution is then performed.

An analysis of the algorithm is in order. In phase 1, the immediate reward determines rule induction. This means that those steps (state/action pairs) that lead straight to a positive reinforcement will be strongly favored. The downside of such rule induction is obvious: the agent, if relying completely on these rules, may ignore bigger reinforcements and dwell on insignificantly small immediate reinforcements. Thus, phase 1 can only be used for a short period at the beginning of the training to get the agent started.

In phase 2, the temporal difference in Q-values determines rule induction. It encourages the exploration of those steps (state/action pairs) that are worth exploring, i.e., leading to an improvement of a situation (a new state with a higher Q-value). It also encourages the in-
crease of Q-values, since the temporal difference entails also the increase of the Q-value for the current state and action. When learning is on-going, the exploration of promising actions (i.e., actions with a large temporal difference) is indeed needed, but the agent should not ignore those actions that have already been proven to be effective (i.e., those that already have large Q-values). This is achieved through combining the outcomes from the rules and the Q-learning network stochastically. In addition, the generalization process encourages the exploration of those steps that are similar in some ways to the promising steps (i.e., the generalization of those promising steps). However, generalization is not truth-preserving; it could lead to bad as well as good results.

In phase 3, a rule is induced if it represents the best action in a given state. This phase is typically used when learning is close to completion. Induced rules help to provide stability where it is needed. That is, for those states that are covered by rules (those states that have been experienced before), randomness is reduced through using rules (which always have values 1) as part of stochastic decision making; for those states where there is no rule (that is, those states which the agent does not know much about), randomness remains at the same level as before. Although we can simply reduce the randomness (temperature) parameter $\alpha$ to reduce the randomness, this change would be uniform, which is less desirable. As a result of selective reduction of randomness, phase 3 should improve the performance of the agent at the end of the training. These induced rules can also help transfer to new settings if the underlying regularity of a new setting is close enough to the old setting. Generalization may or may not improve performance, because although generalization to new states could provide useful guidance, it could also be wrong.

Experiments

Experiments with Mazes

We use a simple maze (see Figure 5) as the primary setting. In the maze, the agent has rudimentary sensory inputs regarding its immediate left, front and right side, indicating whether there is a wall, an opening, or the goal; the agent can move forward, turn to the left, or turn to the right. It has no information regarding its location except the simple sensory input described above. It has no episodic memory capability. It has no information about the goal, which has to be inferred from reinforcements. The reward for an agent reaching the goal is 1, and the punishment for hitting a wall is -0.1.

We first choose (optimize) the structures and parameters of backpropagation and Q-learning through trial-and-error: 8 hidden units are used, the learning rate is 0.1, the momentum parameter is 0.7, network weights are randomly initialized between -0.01 and 0.01; the Q-value discount rate is 0.9, the randomness parameter for stochastic decision making is set at 0.1. (Note that performance is not sensitive to these parameters.) The lengths of phase 1, 2 and 3 are 3, 20, and 37 episodes, respectively.

Learning Speed  Figure 6 shows the learning speeds, where learning speeds are measured by the total number of moves in the first 60 episodes. Perc. $x$ refers to the versions using the percentage combination with rules being applied $p = x\%$ of the times. Stoc. $y$ refers to the versions using the stochastic combination with rules being weighted at $w_1 = y\%$ and $w_2 = 1 - w_1$. The symbol gen indicates that generalization is performed. We recorded the results averaged over 50 trials with different random seeds.

It is clear from the figure that, when rules are used frequently (e.g., with Perc. 80 or Stoc. 20), CLARION learns faster than pure Q-learning by large margins. A $t$ test showed the differences were significant with over 99% confidence ($p < 0.01$). The data also indicate that generalization per se does not lead to faster learning. In addition, the number of rules learned at the end of the 60 episodes is relatively stable (with a mean around 7 and a small standard error). A comparison of the learning curves is shown in Figure 7.

Trained Performance  In Figure 8, we show the average number (averaged over 50 trials) of steps needed to reach the target in one episode, after 60 episodes of training, for the different models. The numbers are shown in the Moves column. The different versions of CLARION again outperform pure Q-learning by large

<table>
<thead>
<tr>
<th>Moves</th>
<th>Q-learning</th>
<th>Per c.60</th>
<th>Per c.80</th>
<th>Per c.60.gen</th>
<th>Per c.80.gen</th>
<th>Stoc.15</th>
<th>Stoc.20</th>
<th>Stoc.15.gen</th>
<th>Stoc.20.gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>15,348.48</td>
<td>15,348.48</td>
<td>9,449.42</td>
<td>7,726.20</td>
<td>5,666.26</td>
<td>5,666.26</td>
<td>5,040.86</td>
<td>9,726.20</td>
<td>6,022.68</td>
<td>6,022.68</td>
</tr>
<tr>
<td>3,726.20</td>
<td>3,726.20</td>
<td>2,726.20</td>
<td>2,726.20</td>
<td>2,726.20</td>
<td>2,726.20</td>
<td>2,726.20</td>
<td>2,726.20</td>
<td>2,726.20</td>
<td>2,726.20</td>
</tr>
</tbody>
</table>

Figure 5: The initial maze

The starting position is marked by 'S' in which the agent faces upward to the upper wall. The goal is marked by 'G'.

Figure 6: A Comparison of Learning Speeds

Moves indicate the total numbers of moves during training (averaged over 50 trials). Rules indicate the average numbers of rules at the end of training.

Figure 7:}

![Figure 7](image-url)
The reinforcements for CLARION are produced from two sources. One is the gradien t rew ard, which is pro/\-portional to the c hange in the "range" readings (i.e. the change in the distance to the target). The other is the end rew ard, which is determined b y how successful the agent is at the end of an episo de. The end rew ard is 1 if the agent reaches the target within the allotted time, and is in v ersely prop ortional to the distance (from the target) if the agent runs out of time or gets blown up.

In this experiment, each time the minefield is gener/\-ated anew in a random la yout, but it always contains the same number of mines, which in this case is 10. The time allotted to the agent for each episo de is 200 steps. Figure 12 shows learning differences between CLARION and Q-learning, where learning is measured by the total number of successful episo des out of a total 500 training episo des. CLARION again outperforms Q-learning alone.
Figure 11: 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.

Figure 12: Learning
The number of successful episodes during training is included for each case.

Discussions
The type of task CLARION tackles differs from the tasks dealt with by traditional learning algorithms. Most of the supervised concept/rule learning algorithms (such as AQ and ID3) require consistent data and pre-classification (and are expected to extract only perfect classification rules). As “batch” algorithms, they require the agent to obtain all data before learning starts, which means high space complexity, more than what is typically available to a situated agent in reactive sequential decision tasks, and slow start. They cannot be applied directly to sequential decision tasks, because they do not perform temporal credit assignment and thus cannot handle sequences well. There is also the incremental nature of supervised learning (e.g., the version space algorithm of Mitchell 1982). Although incremental, they require labeled, complete, and consistent descriptions of instances, which is not available to CLARION. They cannot handle sequences in reactive sequential decision tasks either.

Compared with unsupervised rule/concept learning algorithms, such as Fisher (1987) and Stepp and Michalski (1983), CLARION differs from these systems in that (1) it does have feedback available (reinforcements or payoffs) although there is no direct supervision; (2) it usually does not have available to it a complete description of an instance on which it can base its decision; (3) temporal credit assignment is necessary.

Most of the existing models that contain both declarative and procedural knowledge explore mainly top-down learning (advice taking), such as Anderson (1982, 1990), Gelfand et al (1989), Maclin and Shavlik (1994) and Schneider and Oliver (1991). CLARION explores bottom-up learning, to demonstrate how conceptual/symbolic knowledge can emerge through extracting information from lower-level representation and through interacting with the world in the same way as subconceptual/procedural knowledge does.

Grefenstette (1982) and Schultz (1991) developed a model SAMUEL for learning minefield navigation using GA-based search. GA uses a delayed “reinforcement” calculated based on the summed past experience at the end of each “generation” and thus makes learning slow. GA may also require memorizing long sequences in order to assign credits properly for success/failure.

Conclusions
The analysis of the nature of reactive sequential decision tasks reveals a number of unique characteristics. Thus, a hybrid architecture CLARION is developed. Its learning is reactive, experience-driven, and on-line, developing both subsymbolic and symbolic representations in a bottom-up direction through exploring the world. CLARION is able to learn faster and transfer better than one-level learning alone. Experiments in two different tasks demonstrate the advantage of the model, and show that the combination of the two types of knowledge yields synergistic performance.

References
J. Gelfand, D. Handelman and S. Lane, (1989). Integrating Knowledge-based Systems and Neural Networks


