Soar Workshop
RL Tutorial
May 14, 2018
Topics

• RL as a learning mechanism
• Architecture & agent design
• Eater integration
What is Reinforcement Learning (RL)?

- One of the core tasks in Machine Learning (ML)
  - In addition to supervised & unsupervised

- Goal: learn an optimal action **policy**; given an environment that provides states, affords actions, and provides feedback as numerical **reward**, maximize the expected future reward
  - Typically involves **learning** a **value function** that maps states (or state-action pairs) to a prediction of expected future reward
RL Cycle

Goal: learn an action-selection policy such as to maximize expected receipt of future reward

Agent

Environment

state \( s_t \)

reward \( r_{t+1} \)

action \( a_t \)

next state \( s_{t+1} \)
Soar 9

Symbolic Long-Term Memories
- Procedural
- Semantic
- Episodic

Symbolic Working Memory

Spatial Visual System
- Object-based
- continuous metric space

Decision Procedure

Perception

Action

Reinforcement Learning

Chunking

Semantic Learning

Episodic Learning
Methods for Learning Procedural Knowledge

**Chunking**
- Converts *deliberation* in substates into *reaction* via rule compilation
- Creates new rules

**Reinforcement Learning**
- *Tunes* operator numeric preferences to reflect expectation of reward
- Updates existing rules

*Can be used together*
Soar Basic Functions

1. **Input** from environment
2. Elaborate current situation: *parallel rules*
3. Propose operators via acceptable preferences
4. Evaluate operators via *preferences: Numeric indifferent preference*
5. **Select operator**
6. Apply operator: Modify internal data structures: *parallel rules*
7. **Output** to motor system [and access to long-term memories]
Left-Right Demo

1. Soar Java Debugger
2. Source *left-right.soar* file

![Diagram showing left-right transition with values -1 and +1]
Left-Right Demo

*Script*

1. srand 50412
2. step
3. run 1 -p
4. click: op_pref tab
   ➢ note numeric indifferents
5. print left-right*rl*left
6. print left-right*rl*right
7. run
   ➢ note movement direction
8. print left-right*rl*left
9. print left-right*rl*right
10. init-soar
11. Repeat from #2 (~5 times)
Left-Right: Takeaways

Reinforcement learning changes rules in procedural memory

• Changes are persistent
• Change affects numeric indifferent preferences, which in turn affects the selection of operators
• Change is in the direction of the underlying reward signal (will discuss this more shortly)
RL -> Architecture & Agent Design

Value function
via RL rules [agent]

Reward
via working-memory structures [architecture, agent]

Policy updates
via Temporal Difference (TD) Learning [architecture]
RL Rules

The RL mechanism maintains Q-values for state-operator pairs in specially formulated rules, identified by syntax

- RHS with a single action, asserting a single numeric indifferent preference with a constant value

```
sp {left-right*rl*left
   (state <s> ^name left-right
    ^operator <op> +)
   (<op> ^name move
    ^dir left)
   -->
   (<s> ^operator <op> = 0)}
```

```
sp {left-right*rl*right
   (state <s> ^name left-right
    ^operator <op> +)
   (<op> ^name move
    ^dir right)
   -->
   (<s> ^operator <op> = 0)}
```
Left-Right Demo

Focus: RL Rules

1. Soar Java Debugger
2. Source left-right.soar file
3. print --full --rl
4. run
5. print --full --rl
6. print --rl
Reward Representation

Each state in WM has a **reward-link structure**

Reward is recognized by syntax

\[
(<\text{reward-link}> \ ^\text{reward} \ <\text{r}>) \\
(<\text{r} \ ^\text{value} \ [\text{integer or float}])
\]

- The reward-link is **not** directly modified by the environment or architecture (i.e. requires agent interpretation/management)
- Reward is collected at the beginning of each *decide* phase
- Reward on a state’s reward-link pertains only to that state (more on this later)
- Reward can come from multiple sources: reward values are summed by default
Reward Rule Examples

sp \{left-right*reward*left

(state <s> ^name left-right
^location left
^reward-link <rl>)

-->

(<rl> ^reward <r>)
(<r> ^value -1)\}

sp \{left-right*reward*right

(state <s> ^name left-right
^location right
^reward-link <rl>)

-->

(<rl> ^reward <r>)
(<r> ^value 1)\}
RL Cycle

Agent

Environment

state $s_t$

reward $r_{t+1}$

action $a_t$

state $s_{t+1}$
## RL Cycle in Soar

<table>
<thead>
<tr>
<th>Input</th>
<th>Propose</th>
<th>Decide</th>
<th>Apply</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

May 14, 2018

Reinforcement Learning in Soar

17
## RL Cycle in Soar

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Propose</th>
<th>Decide</th>
<th>Apply</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td></td>
<td></td>
<td>state_d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## RL Cycle in Soar

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Propose</th>
<th>Decide</th>
<th>Apply</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>$\text{state}_d$</td>
<td>$\text{evaluate operators}_d$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# RL Cycle in Soar

<table>
<thead>
<tr>
<th>Input</th>
<th>Propose</th>
<th>Decide</th>
<th>Apply</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>state$_d$</td>
<td>evaluate operators$_d$</td>
<td>select operator$_d$</td>
<td></td>
</tr>
<tr>
<td>d+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# RL Cycle in Soar

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Propose</th>
<th>Decide</th>
<th>Apply</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>$\text{state}_d$</td>
<td>$\text{evaluate operators}_d$</td>
<td>$\text{select operator}_d$</td>
<td></td>
<td>initiate external action(s)</td>
</tr>
<tr>
<td>d+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

May 14, 2018

Reinforcement Learning in Soar
## RL Cycle in Soar

<table>
<thead>
<tr>
<th>Input</th>
<th>Propose</th>
<th>Decide</th>
<th>Apply</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>$\text{state}_d$</td>
<td>$\text{evaluate operators}_d$</td>
<td>$\text{select operator}_d$</td>
<td>initiate external action(s)</td>
</tr>
<tr>
<td>$d+1$</td>
<td>$\text{state}_{d+1}$</td>
<td>$\text{reward}_{d+1}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## RL Cycle in Soar

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Propose</th>
<th>Decide</th>
<th>Apply</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>$\text{state}_d$</td>
<td>$\text{evaluate operators}_d$</td>
<td>$\text{select operator}_d$</td>
<td></td>
<td>initiate external action(s)</td>
</tr>
<tr>
<td>$d+1$</td>
<td>$\text{state}_{d+1}$</td>
<td>$\text{evaluate operators}_{d+1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{reward}_{d+1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## RL Cycle in Soar

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Propose</th>
<th>Decide</th>
<th>Apply</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td></td>
<td>evaluate operators\textsubscript{d}</td>
<td>select operator\textsubscript{d}</td>
<td></td>
<td>initiate external action(s)</td>
</tr>
<tr>
<td></td>
<td>state\textsubscript{d}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d+1</td>
<td></td>
<td>evaluate operators\textsubscript{d+1}</td>
<td>select operator\textsubscript{d+1}</td>
<td>update policy\textsubscript{d}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>state\textsubscript{d+1}</td>
<td>reward\textsubscript{d+1}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RL Updates

• Takes place during *decide* phase, after operator selection
• For all RL rule instantiations \((n)\) that supported the *last* selected operator

\[
value_{d+1} = value_d + \left( \frac{\delta_d}{n} \right)
\]

Where, roughly...

\[
\delta_d = \alpha [ \text{reward}_{d+1} + \gamma(q_{d+1}) - value_d ]
\]

Where...

• \(\alpha\) is a parameter (learning rate)
• \(\gamma\) is a parameter (discount rate)
• \(q_{d+1}\) is dictated by learning policy
  • On-policy (SARSA): value of selected operator
  • Off-policy (Q-learning): value of operator with maximum selection probability
Value Function

Issues

Structure

• What features comprise RL-rule conditions (tradeoff: convergence speed vs. performance)

• High dimensionality -> computationally infeasible

Initialization

• Quality estimates may bootstrap agent performance and reduce time to convergence
Eaters RL

• General idea:
  • RL rules will learn to select between forward and rotate operators.
Eaters RL 1

Get your eater code
Add to top of file or
create a new file (eater-RL.soar)
– turn on RL
  • `rl -s learning on`
  • `indiff -g` # use greedy decision making
  • `indiff -e 0.001` # low epsilon
Eaters RL 2

Remove indifferent preference from proposals so RL rules will influence decision.

sp {random*propose*forward
  (state <s> ^name eater
   ^io.input-link.front)
-->
  (<s> ^operator <op> +)
  (<op> ^name forward)}

sp {random*propose*rotate
  (state <s> ^name eater
   ^io.input-link.front)
-->
  (<s> ^operator <op> +)
  (<op> ^name rotate)}

Just add these to a new file and they will load over your old versions.
Eaters RL 3

Generate RL rules for every color and operator combination:

\[
gp \{\text{eater*evaluate*forward} \\
\text{ (state } <s> \text{ ^name eater} \\
\quad ^\text{io.input-link.front} [ \text{ red wall blue empty green purple } ] \\
\quad ^\text{operator } <op1> +) \\
\quad (<op> ^\text{name forward}) \\
\quad --\rightarrow \\
\quad ( <s> ^\text{operator } <op1> = 0.0) \}
\]

\[
gp \{\text{eater*evaluate*rotate} \\
\text{ (state } <s> \text{ ^name eater} \\
\quad ^\text{io.input-link.front} [ \text{ red wall blue empty green purple } ] \\
\quad ^\text{operator } <op1> +) \\
\quad (<op1> ^\text{name rotate}) \\
\quad --\rightarrow \\
\quad ( <s> ^\text{operator } <op1> = 0.0) \}
\]

Each of these will generate 6 rules!

RL will change the value of \( = 0.0 \) in each of the rules as it learns
Eaters RL 4

Add rule that assigns reward – use the change in score:

\[
\text{sp \{eater*elaborate*state}
\]

\[
(\text{state <s> ^name eater}
\]

\[
^\text{reward-link <rl>}
\]

\[
^\text{io.input-link.score-diff <d>})
\]

---

\[
(\text{<rl> ^reward.value <d>})
\]

}
Run!

• Run eater
• Look at rl rules: \( p - r \)
• Reset eater (type “r”), run again
• See how rl rules change:
  • Number of updates
  • Value of indifferent preference

• Gets better, but is very limited by the operators available (forward and rotate).