Reinforcement Learning (RL)
IJCAI 2016

Nate Derbinsky
Wentworth Institute of Technology

John Laird
University of Michigan
Topics

• RL as a learning mechanism
• Simple example
• Architecture & agent design
• Eater integration
Symbolic Long-Term Memories

- Procedural
- Semantic
- Episodic

Symbolic Working Memory

Spatial Visual System
- Object-based continuous metric space

Perception

Action

Decision Procedure

Reinforcement Learning

Chunking

Semantic Learning

Episodic Learning
RL Cycle

Goal: learn an action-selection policy such as to maximize expected receipt of future reward
Soar: **learn numeric preferences in rules** to maximize expected receipt of future reward [in rules]
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
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

\[ \text{via RL rules [agent]} \]

Reward

\[ \text{via working-memory structures [architecture, agent]} \]

Policy updates

\[ \text{via Temporal Difference (TD) Learning [architecture]} \]
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)}
```
Reward Representation

Each state in WM has a reward-link structure

Reward is recognized by syntax

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

• 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
    ^reward-link <rl>)

-->

(<rl> ^reward <r>)

(<r> ^value
}

sp {left-right*reward*right

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

-->

(<rl> ^reward <r>)

(<r> ^value
}
RL Updates

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

\[
\text{value}_{d+1} = \text{value}_d + (\delta_d / n)
\]

Where, roughly...

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

Where...
- *α* is a parameter (learning rate)
- *γ* 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
Eaters RL: General Idea

• Reward comes from:
  • eating food
  • -1 for movement (push toward efficiency)

• RL rules will learn to select between forward and rotate operators based on reward
Eaters RL 1: Enable RL

Get your eater code

Add to top of file – turn on RL

- `rl -s learning on`
- `indiff -g`  # use greedy decision making
- `indiff -e 0.1`  # low epsilon
Eaters RL 2: Modify Proposals

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

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

sp {propose*rotate
    (state <s> ^name eater
     ^io.input-link.time)
  -->
    (<s> ^operator <op> +)
    (<op> ^name rotate})
```
Eaters RL 3: General RL-Rules: GP

Generate RL rules for every color and operator combination:

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

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

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: Reward

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

\[
\text{sp} \{\text{eater}\text{elaborate*state}
  
  \text{(state <s> ^name eater}
  
    ^reward-link <rl>
    
    ^io.input-link.score-diff <d>)

  -->

  (<rl> ^reward.value <d>)
\]
Eaters RL 5: Run!

- Run eater
- Look at rl rules: $p \rightarrow r$
- 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).