RL Tutorial

Soar Workshop 32 – Nate Derbinsky

While waiting…

1. Make sure you have internet access
2. Download Soar Tutorial package v9.3.2
   code.google.com/p/soar/wiki/SoarTutorial
3. Download Graphviz
   www.graphviz.org
4. Download Eclipse (with at least Java)
   www.eclipse.org
5. Download tutorial support files
   web.eecs.umich.edu/~nlderbin/workshop32

Setting Expectations

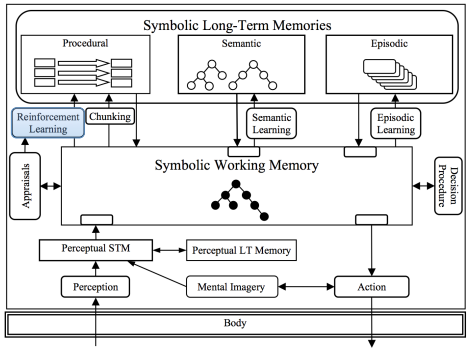
Not a tutorial on Reinforcement Learning (RL)

Reinforcement Learning: An Introduction
Richard S. Sutton, Andrew G. Barto

Topics

- RL as a learning mechanism in Soar
- Agent design
- Advanced issues
- Additional resources

Soar 9



Procedural Learning Mechanisms

Chunking

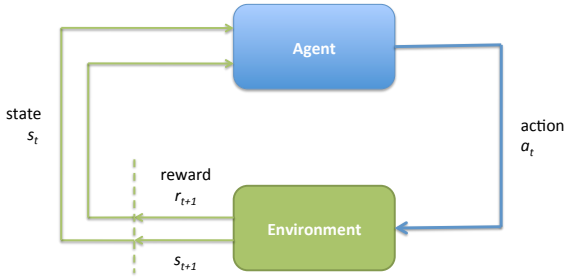
- Converts *deliberation* in substates into *reaction* via rule compilation
- Creates new rules

Reinforcement Learning

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

Reinforcement Learning Cycle

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



Left-Right Demo

1. Soar Java Debugger
2. Source left-right agent
   `Agents/left-right/left-right.soar`

Left-Right Demo

Script

1. `srand 5041231`
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)

Components of RL

- RL rules
- Reward representation
- Learning

RL Rules

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

- RHS must be 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)
}
```


Pop Quiz!

Which of the following are valid RL rules?

✖
```
sp {rule1
   (state <s> ^name quiz
              ^operator <op> +)
   (<op> ^name op-name)
-->
   (<s> ^operator <op> = 1, >)}
```

✖
```
sp {rule2
   (state <s> ^name quiz
              ^operator <op> +)
   (<op> ^name op-name)
-->
   (<s> ^operator <op> = 0.8)
   (write |debugging|)}
```

☺
```
sp {rule3
   (state <s> ^name left-right
              ^operator <op> +)
   (<op> ^name move
         ^dir right)
-->
   (<s> ^operator <op> = -1.4)}
```

✖
```
sp {rule4
   (state <s> ^name left-right
              ^operator <op> +)
   (<op> ^name move
         ^dir right)
-->
   (<s> ^operator <op> = <one>)}
```


Left-Right Demo

Focus: RL Rules

1. Soar Java Debugger
2. Source left-right agent
   `Agents/left-right/left-right.soar`
3. `print --full --rl`
4. `run`
5. `print --full --rl`
6. `print --rl`
7. Attempt sourcing rules from previous slide
   - Check with `print --rl`

Reward Representation

Soar creates a `reward-link` structure on each state in WM
- Soar Java Debugger
  1. `step 5`
  2. `print --exact (* ^reward-link *)`

Reward is recognized by syntax
```
(<reward-link> ^reward <r>)
(<r> ^value [val])
```

- [val] must be a numeric constant (integer or float)
- The reward-link is **not** directly modified by IO
- The reward-link is **not** automatically cleaned
- Reward is collected at the beginning of each *decide* phase
- Reward on a state's reward-link pertains to that state (more on this later)
- Multiple reward values are summed by default

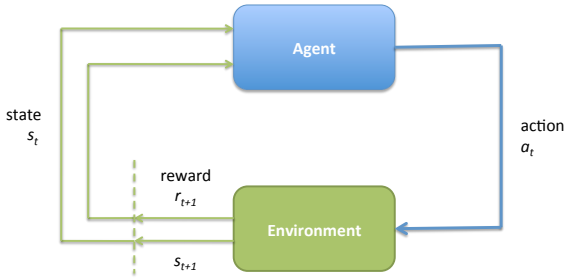
Reward Rule Examples

```
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)
}
```

Reinforcement Learning Cycle

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



RL Cycle in Soar

Decision	Input	Propose	Decide	Apply	Output
d					
d+1					

RL Cycle in Soar

Decision	Input	Propose	Decide	Apply	Output
d	$state_d$				
d+1					

RL Cycle in Soar

Decision	Input	Propose	Decide	Apply	Output
d	$state_d$	evaluate $operators_d$			
d+1					

RL Cycle in Soar

Decision	Input	Propose	Decide	Apply	Output
d	$state_d$	evaluate $operators_d$	select $operator_d$		
d+1					

RL Cycle in Soar

Decision	Input	Propose	Decide	Apply	Output
d	$state_d$	evaluate $operators_d$	select $operator_d$		initiate external action(s)
d+1					

RL Cycle in Soar

Decision	Input	Propose	Decide	Apply	Output
d	$state_d$	evaluate $operators_d$	select $operator_d$		initiate external action(s)
d+1	$state_{d+1}$ $reward_{d+1}$				

RL Cycle in Soar

Decision	Input	Propose	Decide	Apply	Output
d	$state_d$	evaluate $operators_d$	select $operator_d$		initiate external action(s)
d+1	$state_{d+1}$ $reward_{d+1}$	evaluate $operators_{d+1}$			

RL Cycle in Soar

Decision	Input	Propose	Decide	Apply	Output
d	$state_d$	evaluate $operators_d$	select $operator_d$		initiate external action(s)
d+1	$state_{d+1}$ $reward_{d+1}$	evaluate $operators_{d+1}$	select $operator_{d+1}$ update $policy_d$		

RL Updates

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

$$value_{d+1} = value_d + (\delta_d/\mathbf{n})$$

Where, roughly…

$$\delta_d = \alpha[reward_{d+1} + \Upsilon(q_{d+1}) - value_d]$$

Where…
- α is a parameter (learning rate)
- Υ 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

Left-Right Demo

Focus: RL Updates

1. Soar Java Debugger
2. Source left-right agent
   `Agents/left-right/left-right.soar`
3. `watch --rl`
4. `run`
5. `print s1`
6. `print --depth 2 r1`
7. `rl --stats`
8. `rl`

$$\delta = 0.3\left[1 + 0.9(0) - 0\right] = 0.3$$
$$\delta = 0.3\left[1 + 0.9(0) - 0.3\right] = 0.21$$

Controlling RL in Soar

Get/Set a parameter:

- `rl [-g|--get] <name>`
- `rl [-s|--set] <name> <value>`

Try changing learning rate to 1 (deterministic)

1. `rl`
2. `rl --set learning-rate 1`
3. `rl`

RL Agent Design

1. RL rules
2. Reward rules
3. Enable RL

Baseline: `left-right-start.soar`
web.eecs.umich.edu/~nlderbin/workshop32/rl

1. RL Rules

left-right-1.soar

a) Add RL rules for moving left/right (slide 10)
   - Check via `print --rl`

b) Remove indifferent preference from operator proposal rules
   - Note: this will cause a tie impasse if (a) is not done correctly

2. Reward Rules

left-right-2.soar

Add rules to add reward to reward-link, dependent upon the location to which the agent decided to move (slide 14)

3. Enable RL

left-right-3.soar

By default, RL is **disabled**. To enable, set the parameter (slide 26):

```
rl --set learning on
```

Note: the agent will function without this step, but rules will not update, and hence will act identically to the baseline agent.

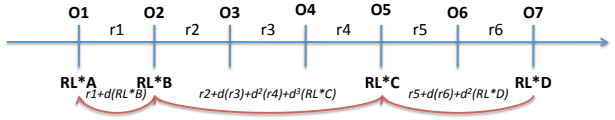
Advanced Issues

- Gaps in rule coverage
- RL in substates
- Exploration policies
- Rule generation

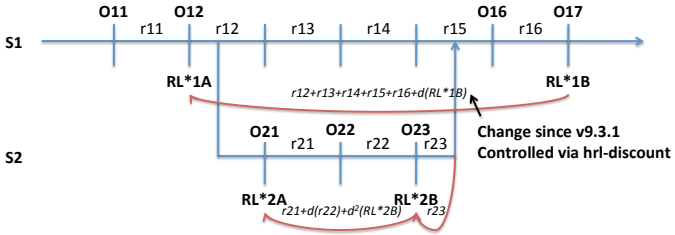
Gaps in Rule Coverage

Gap: one or more contiguous decision cycles during which no RL rules fire

By default, Soar will automatically propagate RL updates over gaps, where rewards are discounted with respect to the length of the gap



RL in Substates



- Rewards are collected independently on each state
- Rewards at a superstate are attributed to the last RL-supported, selected operator

Exploration Policies

There are numerous policies for selecting operators probabilistically (see Manual)

- Deterministic (first, last)
- Softmax (default)
- Epsilon Greedy (change since v9.3.1 – doesn't switch!)
- Boltzmann

Pertinent commands (see Manual)

- `indifferent-selection`
- `predict`
- `select`

Rule Generation

The number of RL rules required for an agent is usually unfeasibly large to write by hand

Options

- `gp` command
- Rule templates
- Chunking

gp Command

- Similar syntax to `sp` command
- Generates rules, at source time, according to the cross-product of a fixed number of dimensions, each with a fixed domain
  - Very fast, potentially creates a lot more rules than will be encountered by the agent

```
gp {example*gp
   (state <s> ^name left-right
              ^operator <op> +
              ^d1 [ a b c ]
              ^d2 [ 1 2 3 ])
   (<op> ^name move
         ^dir [ left right up down ])
-->
   (<s> ^operator <op> = 0)
}
```

```
> Total: 36 productions sourced.
> print
example*gp*36
example*gp*35
…
```

Left-Right Demo

Focus: gp (left-right-3-gp.soar)

Replace RL rules (slide 28) with a single `gp` command

- Source agent
- Verify rules: `print --full --rl`

Rule Templates

- Allows Soar to dynamically generate new RL rules based on a predefined pattern as the agent encounters novel states
- Similar to syntax of RL rules
  - Requires `:template` flag
    - Note: without will be an RL rule that matches multiple states
  - Numeric indifferent preference value can be a variable
- Behavior
  - Rather than firing, creates new RL rules from instantiations
  - Significantly slower than `gp`

Left-Right Demo

Focus: template (left-right-3-template.soar)

Replace RL rules (slide 28) with a single rule template

1. Source agent
2. Verify rules
   - `print --full --rl`
   - `print --full --template`
3. `run`
4. Verify rules
   - `print --full --rl`
   - `print --full --template`

Chunking

"Since RL rules are regular productions, they can be learned by chunking just like any other production. This method is more general than using the gp command or rule templates, and is useful if the environment state consists of arbitrarily complex relational structures that cannot be enumerated."

- Soar Manual

Additional Resources

- Documentation
- Demo agents
- Readings

Documentation

Manual and Tutorial

`Documentation/`

Additional Topics

- RL update details
- Eligibility traces
- Additional learning agent walkthrough (water-jug)
- Usage: commands, parameters, statistics, etc.

…

Demo Agents

`Agents/`

- Left-Right
- Water Jug RL
  - Discussed in Soar-RL Tutorial

Select Readings

code.google.com/p/soar/wiki/Publications

2005
- Soar-RL: Integrating Reinforcement Learning with Soar
  - Shelley Nason, John E. Laird (Cognitive Systems Research)

2007
- The Importance of Action History in Decision Making and Reinforcement Learning
  - Yongjia Wang, John E. Laird (ICCM)

2008
- A Computational Unification of Cognitive Control, Emotion, and Learning
  - Robert P. Marinier III (Dissertation)

2009
- Learning to Play Mario
  - Shiwali Mohan (Technical Report)
- Hierarchical Reinforcement Learning in the Taxicab Domain
  - Mitchell K. Bloch (Technical Report)

2010
- Instance-Based Online Learning of Deterministic Relational Action Models
  - Joseph Xu, John E. Laird (AAAI)
- Using Imagery to Simplify Perceptual Abstraction in Reinforcement Learning Agents
  - Samuel Wintermute (AAAI)

2011
- Learning to Use Episodic memory
  - Nicholas Gorski, John E. Laird (Cognitive Systems Research)

2012
- The Soar Cognitive Architecture
  - Laird (MIT Press)

Extra Time

- Extend Water-Jug base agent to use RL
  - See Soar-RL Tutorial for guidance