Reinforcement Learning for Production-based Cognitive Models

Adrian Brasoveanu Jakub Dotlačil

1. Learnability of production-based models

Main goal: framework to explore in a computationally explicit way the learnability of mechanistically-specified cognitive models of linguistic skills, e.g., the parsers in Lewis and Vasishth (2005); Hale (2011); Engelmann (2016).

Learnability is a major issue for cognitive models that use theoretically-grounded linguistic representations and processes, as they call for:

- richly structured representations
- complex rules that require a significant amount of hand-coding

Learnability of production-based models can be divided into **two questions**:

(*i*) **rule acquisition**: how do we form complex rules out of simpler ones?

(*ii*) **rule ordering**: how do we decide which rule to fire when?

ACT-R's (Anderson and Lebiere, 1998) partial answers: (*i*) prod. compilation, (*ii*) utility estimation. Neither systematically applied to complex models of linguistic skills.

Main contribution here: focus initially on easier question (*ii*), show how to leverage Reinforcement Learning (RL, Sutton and Barto 2018) to answer it.

RL & ACT-R have close connections (Fu and Anderson, 2006), but largely unexplored.

2. Rule-based model of lexical decision (LD)

LD tasks modeled in ACT-R with small number of rules (Brasoveanu and Dotlačil, 2019), so good starting example. **Three LD tasks** of increasing length (hence difficulty): 1 stimulus (the word *elephant*), 2 stimuli (the word *elephant* and a non-word), and 4 stimuli (*elephant*, non-word, *dog*, another non-word).
Declarative memory: stores lexical knowledge (words) of an English speaker.
Procedural memory: stores production rules to carry out LD tasks.
Rules: conditionalized actions; they execute actions when conditions are met.

3. Rules for LD

Rule 1: Retrieving

~~goal>~~ | ~~STATE:~~ ~~retrieving~~
[stricken out b/c the agent learns goal conditions]
visual> | VALUE: =val
VALUE: ~FINISHED
⟹
goal> | STATE: retrieval-done |
+retrieval> | ISA: word
FORM: =val |

Rule 2: Lexeme Retrieved

~~goal>~~ | ~~STATE:~~ ~~retrieval-done~~ |
retrieval> | BUFFER: full
STATE: free |
⟹
goal> | STATE: retrieving |
+manual> | CMD: press-key
KEY: J |

Rule 3: No Lexeme Found

~~goal>~~ | ~~STATE:~~ ~~retrieval-done~~ |
retrieval> | BUFFER: empty
STATE: error |
⟹
goal> | STATE: retrieving |
+manual> | CMD: press-key
KEY: F |

Rule 4: Finished

~~goal>~~ | ~~STATE:~~ ~~retrieving~~ |
visual> | VALUE: FINISHED |
⟹ goal> | STATE: done |

4. Q-learning for goal-conditioned rules in LD tasks

The 4 rules were initially hand-coded to fire serially. Assume initial goal STATE of ACT-R model is `retrieving`, and *elephant* appears on the virtual screen of the model, which is automatically stored in the VALUE slot of the visual buffer. **Rule 1** fires, attempting to retrieve a word with the form *elephant* from declarative memory, and the goal STATE is updated to `retrieval-done`. When the word is successfully retrieved, **Rule 2** fires and the `J` key is pressed, then:

- 1-stimulus task: the text *FINISHED* is displayed on the screen, then **Rule 4** fires and ends the task.
- 2-/4-stimuli tasks: a non-word is displayed, then **Rule 1** fires; the retrieval attempt fails (cannot retrieve a non-word), so **Rule 3** fires and the `F` key is pressed, after which the next text (*FINISHED* or *dog*) is displayed, etc.

Q: Can we learn how to order the rules if we do not hand-code goal states? (indicated above by striking out goals)
A: Q-learning agents (Watkins, 1989; Watkins and Dayan, 1992; Mnih et al., 2015) **can learn goal-conditioned rules.**
We give the agent a reward of 1 if it reaches the final goal-state `done`; for any intermediate rule firing, we give it a smaller negative reward -0.15 to encourage it to finish the task asap. The agent learns by trial and error to successfully carry out the LD tasks: it learns how to properly order the rules and complete the LD task as efficiently as possible.
Learning is faster and better for shorter tasks (fewer stimuli): given a goal state, the Q-learner selects the same rule as the original hand-coded version by assigning it the highest value relative to the other rules.

- 1-stim: task takes $\approx$ 12 steps; perfect learning; see `https://people.ucsc.edu/~abrsvn/1_stim.html`
- 2-stim: task takes $\approx$ 18 steps; almost perfect learning; see `https://people.ucsc.edu/~abrsvn/2_stim.html`
- 4-stim: task takes $\approx$ 34 steps; learning with some noise; see `https://people.ucsc.edu/~abrsvn/4_stim.html`