Reinforcement Learning for Production-based Cognitive Models

Adrian Brasoveanu (abrsvn@gmail.com)
Department of Linguistics, 1156 High Street
Santa Cruz, CA 95064, USA

Jakub Dotlačil (j.dotlacil@gmail.com)
Utrecht University
Utrecht, The Netherlands

Abstract

We introduce a framework in which we can start exploring in a computationally explicit way how complex, mechanistically specified, and production-based cognitive models of linguistic skills, e.g., ACT-R based parsers, can be acquired. Cognitive models for linguistic skills pose this learnability problem much more starkly than models for other 'high-level' cognitive processes, as they call for richly structured representations and complex rules that require a significant amount of hand-coding. In this paper, we focus on how Reinforcement Learning (RL) methods can be used as a way to solve the production selection problem in ACT-R. Production rules are treated as the actions of an RL agent, and the ACT-R model/mind as the environment in the RL sense. We focus on a basic learning algorithm (tabular Q-learning) and a very simple task, namely lexical decision (LD), framed as a sequential decision problem, with the goal of learning when a specific rule should be fired. Learning is faster and less noisy for shorter LD tasks (fewer stimuli), but the Q-learning agent manages to learn longer tasks fairly well. Realistically long LD tasks and more complex models, e.g., parsers, are left for future research.

Keywords: ACT-R, Reinforcement Learning, production-based models, linguistic skills, lexical decision

Learnability of production-based models

We introduce a framework in which we can start exploring in a computationally explicit way how complex, mechanistically specified cognitive models of linguistic skills, e.g., the parsers in Engelmann (2016); Hale (2011); Lewis and Vasishth (2005), can be acquired. Linguistic cognitive model learnability is an understudied issue, primarily because computationally explicit cognitive models are only starting to be more widely used in psycholinguistics. Cognitive models for linguistic skills pose this learnability problem much more starkly than models for other 'high-level' cognitive processes, since cognitive models that use theoretically-grounded linguistic representations and processes call for richly structured representations and complex rules that require a significant amount of hand-coding.

The learnability problem for production-rule based models can be divided into two parts:

i. rule acquisition – forming complex rules out of simpler ones, and
ii. rule ordering – deciding which rule to fire when.

ACT-R's (Anderson & Lebiere, 1998) partial answers to these problems are production compilation for (i), and rule-utility estimation for (ii). Apart from Taatgen and Anderson (2002), which investigates the role of production compilation in morphology acquisition, neither solution has been systematically applied to complex models for linguistic skills.

We focus here on the easier problem (ii). Our main contribution is to show how advances in the machine learning subfield of Reinforcement Learning (RL, Sutton & Barto, 2018) can be leveraged to solve it. RL and ACT-R have very close connections (Fu & Anderson, 2006), but they have remained largely unexplored.

Learning goal-conditioned rules: An example

The framework and the range of issues that emerge when we try to systematically integrate ACT-R and RL are best showcased with a simple example. We choose a basic learning algorithm, specifically, a tabular Q-learning agent, which is a model-free off-policy learning algorithm (Watkins, 1989; Watkins & Dayan, 1992). Also, we focus on a very simple task, namely lexical decision (LD). In an LD task, participants see a string of letters on a screen. If the participants think the string of letters is a word, they press one key (`J` in our setup); if they think the string is not a word, they press a different key (`F` in our setup). After pressing the key, the next stimulus is presented.

We investigate the extent to which the Q-learning agent can be used to learn goal-conditioned rules in an ACT-R based cognitive model of LD tasks. The main point of proposing and examining an ACT-R model of LD tasks is to construct a simple example of a production-rule based model that enables us to study learnability issues, and that can be scaled up in future work to more complex and cognitively realistic syntactic and semantic parsing models. In particular, the model provides the basic scaffolding of production rules needed for LD tasks, which is all that we need for our purposes. Fleshing the model out to capture major experimental results about LD, or comparing it to previously proposed cognitive models of LD is not our focus here.

LD tasks can be modeled in ACT-R with a small number of rules (see Brasoveanu and Dotlačil 2019 and Chapter 7 in Brasoveanu and Dotlačil 2020 for recent attempts), so they are a good starting example. We model three LD tasks of increasing length, hence difficulty:

i. 1 stimulus: the word *elephant*,
ii. 2 stimuli: the word *elephant* and a non-word, and

iii. 4 stimuli: the word *elephant*, a non-word, the word *dog*, and another non-word.

The model components are split between declarative memory, which stores the lexical knowledge of an English speaker, and procedural memory, which stores rules that enable the model to carry out the LD task. The rules are conditionalized actions: they fire/execute actions when their conditions are satisfied by the cognitive state of the ACT-R mind (the buffers). We assume 4 rules, provided in standard ACT-R format in (1)-(4) below (see Chapter 2 in Brasoveanu and Dotlačil 2020, for example, for more discussion of the format). These rules were originally hand-coded to fire serially by conditioning all the actions on specific goal states. The goal conditions are stricken out below because we remove them and let the Q-learning agent learn them.

(1) **Rule 1: Retrieving**

~~goal>~~ | ~~STATE:~~ ~~retrieving~~ |

visual> | VALUE: =val
VALUE: ∼FINISHED |

$\Longrightarrow$

goal> | STATE: retrieval_done |

+retrieval> | ISA: word
FORM: =val |

(2) **Rule 2: Lexeme Retrieved**

~~goal>~~ | ~~STATE:~~ ~~retrieval_done~~ |

retrieval> | BUFFER: full
STATE: free |

$\Longrightarrow$

goal> | STATE: retrieving |

+manual> | CMD: press-key
KEY: J |

(3) **Rule 3: No Lexeme Found**

~~goal>~~ | ~~STATE:~~ ~~retrieval_done~~ |

retrieval> | BUFFER: empty
STATE: error |

$\Longrightarrow$

goal> | STATE: retrieving |

+manual> | CMD: press-key
KEY: F |

(4) **Rule 4: Finished**

~~goal>~~ | ~~STATE:~~ ~~retrieving~~ |

visual> | VALUE: FINISHED |

$\Longrightarrow$

goal> | STATE: done |

With fully specified, hand-coded rules, the LD task unfolds as follows. Assume the initial goal STATE of the ACT-R model is `retrieving`, and the word *elephant* appears on the virtual screen of the model, which is automatically stored in the VALUE slot of the visual buffer.

At this initial stage, the preconditions of **Rule 1** are satisfied, so the rule fires. As a consequence, we attempt 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. At that point:

i. in the 1-stimulus task, the text FINISHED is displayed on the screen, then **Rule 4** fires and ends the task;

ii. in the 2-stimuli task, the non-word is displayed, then **Rule 1** fires again; the retrieval attempt fails since we cannot retrieve a non-word from declarative memory, so **Rule 3** fires and the `F` key is pressed; at that point, the text FINISHED is displayed on the screen, then **Rule 4** fires and ends the task;

iii. in the 4-stimuli task, the first non-word is displayed, **Rule 1** fires again, then, just as in the 2-stimuli task, **Rule 3** fires and the `F` key is pressed, after which the word *dog* is displayed, **Rule 1** fires for the third time followed by **Rule 2**, which means that the `J` key is pressed and the second non-word is displayed; now, **Rule 1** fires for the final time, followed by **Rule 3**, which triggers the `F` key to be pressed; at this point, the 4-stimuli task is over, so the text FINISHED is displayed on the screen, then **Rule 4** fires and ends the task.

Thus, assuming fully specified, hand-coded rules, the sequences of rule firings for the three LD tasks are as follows:

i. 1-stimulus task: Rules **1 – 2 – 4**

ii. 2-stimuli task: Rules **1 – 2 – 1 – 3 – 4**

iii. 4-stimuli task: Rules **1 – 2 – 1 – 3 – 1 – 2 – 1 – 3 – 4**

Instead of hand-coding the goal-state preconditions, we only specify the actions (and preconditions associated with buffers other than the goal buffer): that's the reason for striking out the goal specifications in (1)-(4). We then let the Q-learning agent learn to successfully carry out the LD tasks. 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 small negative reward of -0.15 to encourage it to finish the task as soon as possible. However, the agent does not get the small penalty if it chooses to wait and fire no rule: this is the optimal course of action when waiting for retrieval requests from declarative memory to complete, for example.

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 tasks as efficiently as possible. This is no small feat given that the actual number of steps, i.e., decision points, when the agents needs to select an action, is larger than the high-level sequences of rule firings discussed above. For example, for a 1-stimulus task, there are actually 12 steps where the agent needs to decide whether to wait or to fire a specific rule (when the agent does not complete the

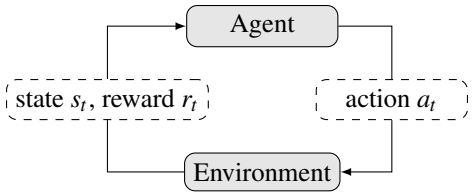

Figure 1: The agent-environment interaction in an MDP

task perfectly, it might take much more than 12 steps). The 2-stimuli task requires 18 such steps (if the task is completed perfectly), and the 4-stimuli task requires 34 steps (again, if the task is completed perfectly). The reason for this is that our LD simulations involve the visual and motor modules (to read strings of characters and press keys) in addition to the declarative memory module. Visual and motor actions, just as retrievals from declarative memory, take time, and the agent needs to make decisions while waiting for them to complete.

The higher the number of steps, i.e., the higher number of decision points for the agent, the harder the task is to learn. As the next sections show, learning is faster and less noisy for shorter tasks (fewer stimuli), but the Q-learning agent manages to learn even the most complex 4-stimuli task fairly well.

Production-rule ordering as an RL problem

Markov decision processes

Markov Decision Processes (MDPs) are the deterministic or stochastic models of decision-making sequences that form the basis of RL approaches to learning. In an MDP, an agent interacts with its environment and needs to make decisions at discrete time steps $t = 1, 2, \ldots, n$. Defining what counts as the agent and what counts as its environment is part of the modeling process: the agent could be a whole cognitive agent (animal, human or robot) acting in the world or in an experimental environment, or it could be a component of a cognitive agent interacting with an 'environment' consisting of other cognitive components.

At every time step t, all the information from the past that is relevant for the current action selection is captured in the current state of the process s_t. This is the Markov property: the future is independent of the past given the current state. The environment passes to the agent the state s_t and, at the same time, a reward signal r_t. The agent observes the current state s_t and reward r_t, and takes an action a_t, which is passed from the agent to the environment. Then, the cycle continues, as shown in Figure 1. At time step $t+1$, the environment responds to the agent's action with a new state s_{t+1} and a new reward signal r_{t+1}. Based on these, the agent selects a new action a_{t+1} etc.

The definitions of 'state' and 'action' depend on the problem, and are also part of the modeling process. The agent's *policy* is a complete specification of what action to take at any time step. Given the Markovian nature of the MPD, the policy π is effectively a mapping from the state space S to the action space A, $\pi : S \to A$. A deterministic policy is a mapping from any given state s_t to an action $a_t = \pi(s_t)$, while a stochastic policy is a mapping from any given state s_t to a probability distribution over actions $a_t \sim \pi(s_t)$.

The agent's goal is to maximize some form of cumulative reward – e.g., total reward, average reward, future-discounted sum of rewards – over an *episode*, which is a complete, usually multi-step interaction between the agent and its environment. In our case, an episode would be a full simulation of an LD task (be it a 1-stimulus or a 2/4-stimuli task).

The agent learns (solves/optimizes the MDP) by updating its policy π to maximize the per-episode cumulative reward.

The standard cumulative reward for an episodic task is the discounted return G, which is the sum of the current reward and the discounted future rewards until the final step n of the episode. In finite/episodic tasks, future rewards are discounted because we assume the agent has a preference for more immediate rewards rather than rewards in the far future. The discount factor γ is a real number between 0 and 1 that determines the present value of future rewards. The discounted return at a time step $t < n$, where n is the final step in the episode, is defined as:

(5) **Discounted return at time** t (γ is the discount factor):
$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \cdots + \gamma^{n-t-1} r_n$

The agent selects actions with the goal of maximizing the expected discounted return.

We define the (state-)action value function $Q_\pi(s,a)$ to be the expected (discounted) return when starting in state s, performing action a, and then following the policy π until the end of the episode. With Q_π in hand, we can solve the optimization problem framed by an MDP: if we know this function, we can always select an optimal action in any state (optimal actions are actions with maximal expected return).

Q-learning, discussed in the next subsection, estimates Q_π by directly sampling experience from the environment. This is in contrast to Dynamic Programming methods, for example, which compute optimal policies under the (unrealistic) assumption that we have a perfect model of the environment.

Q-learning

The agent's goal is to maximize its return. One way of doing this is to bypass the policy and directly estimate the value of all state-action pairs, i.e., estimate the Q function, and improve this estimate based on the interactions between the agent and its environment. With a good estimate of the Q function in hand, we can devise an optimal policy by selecting a maximum-value action in each state.

Tabular Q-learning is an algorithm that enables us to estimate the Q function, and use this estimate as the basis for an optimal policy. The Q function $S \times A \to \mathbb{R}$ is represented as a look-up table that stores the estimated values of all possible state-action pairs. Before learning begins, the Q table is initialized to an arbitrary fixed value (0). The agent then updates the Q table incrementally at each time step t: the value of the pair (s_t, a_t), which consists of the state of the environment s_t

relative to which the agent took the action a_t, is updated based on the reward signal r_{t+1} and the new state s_{t+1} that the agent receives from the environment after taking action a_t.

Q-learning is a form of temporal difference (TD) learning, as the update in (6) shows. The Q^{new} value estimate for the state-action pair (s_t, a_t) is based on the Q^{old} value, updated by some proportion α of the TD error. The parameter α is the learning rate, $0 < \alpha \leq 1$. This update $\alpha \cdot$ TD error is provided on the second line of (6).

(6) **Q-learning update**:

$$Q^{new}(s_t, a_t) \leftarrow Q^{old}(s_t, a_t) + \overbrace{\alpha \cdot \Big(\underbrace{r_{t+1} + \gamma \cdot \underbrace{\max_{a_{t+1}} Q^{old}(s_{t+1}, a_{t+1})}_{\text{next-state value estimate}}}_{\text{TD target (updated value)}} - Q^{old}(s_t, a_t) \Big)}^{\text{TD (temporal difference) error}}$$

The TD error is the difference between the TD target – which is an updated estimate of the value of the (s_t, a_t) pair – and the Q^{old} value estimate. The TD target consists of:

- the reward r_{t+1} the agent receives after action a_t, which is part of the new data the agent gets back from the environment after action a_t, plus
- the estimate of the value of the next state s_{t+1}, where the next state s_{t+1} is the other part of the new data the agent gets back from the environment after action a_t.

The Q-learning estimate for the value of the next state s_{t+1} is discounted by the discount factor γ, since this state is in the future relative to the state-action pair (s_t, a_t) we're currently updating. The Q-learning value estimate for s_{t+1} is aggressively confident/optimistic:[1] the agent looks at all the possible actions a_{t+1} that can be taken in state s_{t+1} and confidently assumes that the action a_{t+1} with the highest Q^{old}-value provides an accurate estimate of the s_{t+1} value.

Q-learning for production-rule ordering

Returning to our ACT-R model of LD tasks, *the agent* (in the RL sense) is a Q-value table that assigns values to all possible state-action pairs, and that guides the rule selection process at every cognitive step. *The environment* is the cognitive state of the ACT-R model/mind, and could conceivably consist of:

i. all the modules – procedural memory, declarative memory and visual and motor modules, together with
ii. their associated buffers – goal, retrieval, visual-what, visual-where and the manual buffer.

This, however, would lead to a very large state space S, which in turn would lead to a large Q-value table. Function-approximation approaches – e.g., using neural networks – would mitigate this problem, but we will continue using a tabular approach here and take a state s to only consist of:

i. the current goal buffer,
ii. the current retrieval buffer,
iii. the value in the visual-what buffer, if any (otherwise, we explicitly mark the visual-what buffer as having no value), and finally,
iv. the state of the manual buffer (busy or free).

Four example states are provided below:

```
- {goal: {state: retrieving}, manual: free,
   retrieval: {}, visual_value: NO_VALUE}
- {goal: {state: retrieval_done}, manual: free,
   retrieval: {}, visual_value: elephant}
- {goal: {state: retrieval_done}, manual: free,
   retrieval: {form: elephant},
   visual_value: elephant}
- {goal: {state: done}, manual: busy,
   retrieval: {}, visual_value: NO_VALUE}
```

The action space consists of the 4 rules in (1)-(4) above, namely `retrieving`, `lexeme retrieved`, `no lexeme found` and `finished`, together with a special action `None` that the agent selects when it wants to not fire any rule because it prefers to wait for a new cognitive state.

The full details of the reward structure are as follows:

- the agent receives a positive reward of 1 at the end of an episode (when the LD task is completed), specifically, when the goal STATE is `done`;
- the agent receives a negative reward of -0.15 for every rule it selects (other than `None`);
- there is no penalty for waiting and selecting no rule, i.e., for selecting the special action `None`;
- at every step, the agent receives a negative reward equal to the amount of time that has elapsed between the immediately preceding step and the current step (multiplied by -1 to make it negative).

This reward structure is designed to encourage the agent to finish the task as soon as possible, and in the process select the smallest number of rules. The negative temporal reward in particular discourages the agent from just repeatedly selecting the special action `None`. This would end up timing out the LD task in a small number of steps, and it would fast-forward the agent to the maximum waiting time the ACT-R environment allows for, which we set to 2 seconds per word for LD tasks.

Thus, we work with two time 'counters' here. On one hand, we have the continuous cognitive-process time that the ACT-R model/mind keeps track of, and which models the reaction time of human participants in experimental tasks. On the other hand, we have the discrete RL time that is the counter for agent-environment interactions: the discrete time steps $t = 1, 2, \ldots, n$ in our MDP. From the perspective of the discrete RL/MDP time, the continuous ACT-R time is a feature of the environment, reflected in the reward signal that the (Q-learning) agent gets when it interacts with, i.e., samples experience from, the environment.

[1] In contrast to the Expected Sarsa estimate, for example (van Seijen, van Hasselt, Whiteson, & Wiering, 2009).

Simulations and results

We assume the usual ACT-R defaults, e.g., rule firing time is set to 50 ms. The learning rate α is set to 10^{-3}, and γ to 0.95. We use an ε-greedy policy for all simulations, with ε multiplicatively annealed from a starting value of 1 to a minimum value of 0.01. Specifically, at every RL step (after every action/rule selection), if $\varepsilon > 0.01$, its value is updated as follows: $\varepsilon \leftarrow \varepsilon \cdot (1 - 10^{-5})$.

One-stimulus task

We simulate 15,000 episodes, i.e., 15,000 LD tasks consisting of 1 stimulus only (the word 'elephant'), from which the Q-learning agent learns. The plot in Figure 2 shows that, after about 5,000 episodes, the task is completed in ≈ 12 steps, which is the length of the task when the agent completes it perfectly. For some episodes, the number of steps is smaller than 12. In these cases, the agent times out the task, e.g., by selecting the `None` action several times, and receives steeply negative temporal rewards leading to very low returns.

A close examination of the agent's final Q-value table, which stores the agent's rule-firing preferences for any given goal state, indicates that the agent has learned goal-conditioned rules perfectly. We only look at states for which at least one action/rule has a non-0 value (recall that all Q-values are initialized to 0). For each such state, we identify the action/rule with the highest value. There are 8 states with at least one non-0 value action. Let's examine them.

There are 3 states in which the agent fires no rule, that is, the maximum-value action is `None`:

```
- {goal: {state: retrieving}, manual: free,
   retrieval: {}, visual_value: NO_VALUE}
- {goal: {state: retrieving}, manual: busy,
   retrieval: {}, visual_value: NO_VALUE}
- {goal: {state: retrieval_done}, manual: free,
   retrieval: {}, visual_value: elephant}
```

We see that when the goal state is `retrieving`, the retrieval buffer is empty, and the visual buffer stores no value, the agent does nothing (whether the manual buffer is busy or free): it simply waits for some text to be automatically detected and stored in the visual buffer. Similarly, when the goal state is `retrieval_done`, the visual buffer stores the value `elephant`, but the retrieval buffer is empty, the agent once again does nothing: it waits for the retrieval process that was just started to complete.

There are 3 states where the max-value action is `finished`:

```
- {goal: {state: retrieving}, manual: busy,
   retrieval: {}, visual_value: FINISHED}
- {goal: {state: retrieval_done}, manual: free,
   retrieval: {form: elephant},
   visual_value: FINISHED}
- {goal: {state: retrieving}, manual: free,
   retrieval: {}, visual_value: FINISHED}
```

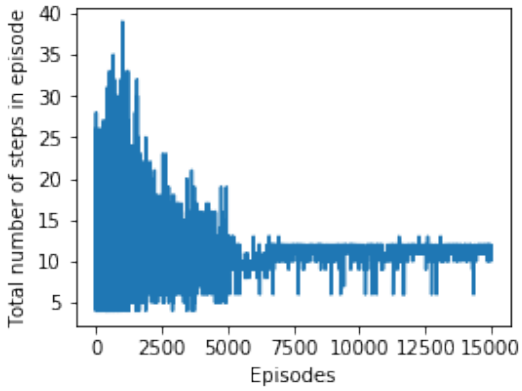

Figure 2: Steps per simulation to complete the 1-stimulus task

We see that whenever the value stored in the visual buffer is FINISHED, the agent correctly chooses the `finished` action irrespective of: (i) the goal state, (ii) the state of the manual buffer, or (iii) the contents of the retrieval buffer.

This leaves us with the 2 state-action pairs below:

```
- {goal: {state: retrieving}, manual: free,
   retrieval: {}, visual_value: elephant}
   ==> retrieving
- {goal: {state: retrieval_done}, manual: free,
   retrieval: {form: elephant},
   visual_value: elephant}
   ==> lexeme retrieved
```

We see that when the goal state is `retrieving` and the visual value is `elephant`, the agent correctly chooses the `retrieving` rule. Finally, when the goal state is `retrieval_done` and the retrieval buffer contains the word `elephant` (i.e., the retrieval process has been successful), the agent correctly chooses the `lexeme retrieved` rule.

Thus, there is no need to hand-code goal states in the conditions of a rule to deterministically guide the cognitive process – at least for this simple 1-stimulus LD task. The Q-learning agent learns by trial-and-error interaction with the environment when to fire which rule, and when to choose to wait and not fire any rule. The agent learns all this from a minimal, but fairly carefully designed, reward structure.

Two-stimuli task

We simulate 15,000 episodes, i.e., 15,000 LD tasks consisting of 2 stimuli only (the word 'elephant' and the non-word 'not_a_word'), from which the Q-learning agent learns. The plot in Figure 3 shows that, after about 9,000 episodes, the task is completed in ≈ 18 steps, which is the length of this task when the agent completes it perfectly.

A close examination of the agent's final Q-value table indicates that the agent has learned goal-conditioned rules almost perfectly. Once again, we only look at states for which at least one action has a non-0 value – a total of 13 states. For each state, we identify the maximum-value action.

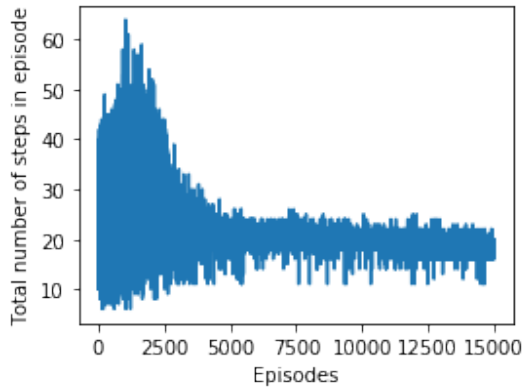

Figure 3: Steps per simulation to complete the 2-stimuli task

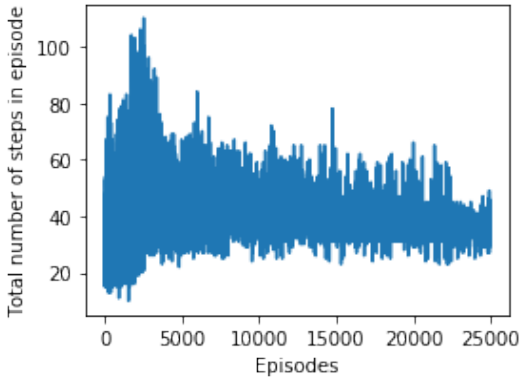

Figure 4: Steps per simulation to complete the 4-stimuli task

There are 4 states where the agent does nothing (selects the `None` action): while waiting for the word 'elephant' to be retrieved, while waiting for a visual value to be automatically detected and stored in the visual buffer (whether the manual buffer is busy or free), and while waiting for the retrieval of the text 'not_a_word' to fail.

There are 4 states where the agent correctly fires the `finished` rule: the visual value is FINISHED in all of them.

There are 5 state-action pairs in which the action is one of the other 3 rules. Out of these, 4 state-action pairs are exactly what we would expect:

- when the goal state is `retrieving`, the retrieval buffer is empty and the manual buffer is free, whether the visual value is `elephant` or `not_a_word`, the agent correctly fires the `retrieving` rule;
- when the goal state is `retrieval_done` and the retrieval process has completed successfully based on the visual value `elephant`, the agent correctly fires the `lexeme retrieved` rule;
- finally, when the goal state is `retrieval_done` and the retrieval process has completed unsuccessfully based on the visual value `not_a_word`, the agent correctly fires the `no lexeme found` rule.

However, unlike in the 1-stimulus task, there is 1 state-action pair that is not optimal, and is simply a reflection of the trial-and-error learning process that takes longer is and more error prone than for the simpler, 1-stimulus task. This state-action pair is the following:

- the goal state is `retrieval_done`, the retrieval buffer contains the word `elephant`, but the visual value is `not_a_word`; in such a state, the agent fires the `retrieving` rule, which is the maximum-value action.

Four-stimuli task

We simulate 25,000 episodes, i.e., 25,000 LD tasks consisting of 4 stimuli (the word 'elephant', the non-word 'not_a_word', the word 'dog' and the non-word 'not_a_word_again'), from which the Q-learning agent learns. We need more episodes for this task because it is longer, hence more complex, than the 1-stimulus or the 2-stimuli tasks. The plot in Figure 4 shows that it takes about 22,000 episodes for the task to be reliably completed in less than 40 steps. The task takes 34 steps when the agent completes it perfectly.

A close examination of the agent's final Q-value table indicates that the agent has learned goal-conditioned rules fairly well, but there still is a pretty large amount of noise associated with multiple goal states. Specifically, there are 24 states total with at least one action with a non-0 value. When we examine the maximum-value action for each of these states, 18 state-action pairs make sense and are as expected.

However, the remaining 6 state-action pairs do not make much sense, similar to the final state-action pair we discussed in the 2-stimuli task subsection. This noise is a reflection of the trial-and-error learning process that becomes increasingly difficult for tasks requiring large numbers of steps. With 4-stimuli, we see that even after 25,000 episodes, the agent still wastes time every now and then trying incorrect rules, or just waiting (selecting the action `None`) for no good reason.

Conclusion and future directions

We argued that the learnability problem of production-based cognitive models can be systematically formulated and computationally addressed as a reinforcement learning problem. But this is merely a first inroad into what promises to be a very rich nexus of learnability questions.

For example, an immediate follow-up would be to explore how RL algorithms perform on a variety of production-based cognitive models, whether linguistic, e.g., syntactic or semantic parsing, or non-linguistic. We have conducted pilot experiments with simple parsing models and tasks, and they are much more difficult than the LD tasks explored in this paper.

One reason is that the cardinalities of the state and action/rule spaces are much larger than for LD tasks, which makes tabular learning less effective and prompts us to explore function-approximation approaches, e.g., neural-

network based approaches (Deep Reinforcement Learning, see Mnih and al 2015 among many others).

The second reason is that parsing tasks are much longer – many more RL/MDP time steps, i.e., action-selection decision points – and standard RL methods are increasingly ineffective in this kind of sparse-reward, long time horizon tasks because the temporal credit assignment problem becomes very difficult. The difficulty is further increased when function-approximation approaches are used – because of vanishing-gradient issues among others. It would be worth exploring function-approximation approaches in realistically long LD tasks first (hundreds of LD stimuli, hence many steps per episode), and only subsequently explore parsing tasks.

This leads us to a second cluster of learnability issues that could be explored. There are other value-based tabular learning algorithms (Sarsa, Expected Sarsa), as well as non-tabular approaches to reinforcement learning (both value and policy based), e.g., linear or non-linear (neural network) function-approximation approaches. We already indicated that exploring these algorithms will most probably be necessary for more complex tasks like parsing, but it is worth exploring their performance and comparing them to Q-learning on LD decision tasks first (both on the tasks we used here, and on realistically long tasks with hundreds of stimuli), so that we establish a broad set of baselines before moving on to other linguistic and non-linguistic production-based cognitive models.

Similarly, we might want to investigate curriculum learning (see Elman 1993 for an early reference, and Rusu and al 2016 among many others for a more recent discussion) for increasingly complex tasks. For example, we could design a curriculum that starts with LD tasks with fewer stimuli and scales up to realistically long LD tasks. Similarly, we could start with parsing short sentences and scale up to sentences with a variety of embedded clauses, or even multi-sentential discourses, after which we could scale up to realistically long self-paced reading or eye-tracking tasks with tens or hundreds of such multi-clausal sentences or discourses.

Curriculum or transfer learning should also enable us to address the fact that it is not cognitively realistic to require such a high number of episodes/trial-and-error interactions for learning. The human cognitive architecture enables us to learn from much fewer interactions, and/or from explicit instructions. For tabular Q-learning, for example, this would mean that the agent starts with a pretrained Q-table.

Finally, a separate line of future work would go beyond our current focus on the easier problem of rule ordering, and investigate the extent to which Hierarchical RL methods (Sutton, Precup, and Singh 1998 among others) could be brought to bear on the harder problem of rule acquisition.

Acknowledgments

We are very grateful to two anonymous ICCM 2020 reviewers for their detailed feedback on an earlier version of this paper, and to the audience of the UCSC Linguistics Department S-circle (May 2020) for their questions and comments about this research project. We gratefully acknowledge the support of the NVIDIA Corporation with the donation of two Titan V GPUs used for this research, as well as the UCSC Office of Research and The Humanities Institute for a matching grant to purchase additional hardware.

References

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Brasoveanu, A., & Dotlačil, J. (2019). Quantitative comparison for generative theories. In *Proceedings of the 2018 berkeley linguistic society 44*.

Brasoveanu, A., & Dotlačil, J. (2020). *Computational Cognitive Modeling and Linguistic Theory*. Springer (Open Access). doi: https://doi.org/10.1007/978-3-030-31846-8

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, *48*, 71-99. doi: 10.1016/0010-0277(93)90058-4

Engelmann, F. (2016). *Toward an integrated model of sentence processing in reading*. Unpublished doctoral dissertation, University of Potsdam, Potsdam.

Fu, W.-T., & Anderson, J. R. (2006). From recurrent choice to skill learning: A reinforcement-learning model. *Journal of Experimental Psychology: General*, *135*(2), 184-206. doi: 10.1037/0096-3445.135.2.184

Hale, J. (2011). What a rational parser would do. *Cognitive Science*, *35*, 399–443.

Lewis, R., & Vasishth, S. (2005). An activation-based model of sentence processing as skilled memory retrieval. *Cognitive Science*, *29*, 1-45.

Mnih, V., & al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Rusu, A. A., & al. (2016). *Progressive neural networks.*

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Precup, D., & Singh, S. P. (1998). Intra-option learning about temporally abstract actions. In *ICML 1998* (pp. 556–564). Morgan Kaufmann.

Taatgen, N. A., & Anderson, J. R. (2002). Why do children learn to say "broke"? a model of learning the past tense without feedback. *Cognition*, *86*(2), 123–155.

van Seijen, H., van Hasselt, H., Whiteson, S., & Wiering, M. (2009). A theoretical and empirical analysis of Expected Sarsa. In *Ieee symposium on adaptive dp and rl* (p. 177-184).

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Unpublished doctoral dissertation, King's College, Cambridge, UK. Retrieved from http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3), 279–292. Retrieved from https://doi.org/10.1007/BF00992698 doi: 10.1007/BF00992698