

# Reinforcement Learning for Production-based Cognitive Models

Adrian Brasoveanu<sup>†</sup> & Jakub Dotlačil<sup>‡</sup>

<sup>†</sup>UC Santa Cruz, <sup>‡</sup>Utrecht University

Virtual MathPsych/ICCM · July 2020

## Overarching goal

**Goal:** explore how complex, production-based cognitive models of linguistic skills can be acquired.

- e.g., the parsers in Brasoveanu and Dotlačil (2020), Engelmann (2016), Hale (2011), and Lewis and Vasishth (2005)

## Why learnability for cognitive models of linguistic skills

- cognitive-model learnability largely understudied for linguistics (we just started using such models more widely)
- the learnability problem is more difficult for models of linguistic skills vs. other 'high-level' cognitive processes because . . .
- . . . committing to theoretically-grounded models requires us to hand-code complex systems of representations and rules

**So, can such complex rule systems be learned from data?**

[given the right kind of learning system and the right kind of data]

## Learnability: rule acquisition and rule ordering

The learnability problem can be divided into two parts:

- 1 rule acquisition: forming complex rules out of simpler ones
- 2 rule ordering: deciding which rule to fire when

ACT-R's (Anderson 2007; Anderson and Lebiere 1998) solutions:

- 1 production compilation – for rule acquisition
- 2 rule-utility estimation – for rule ordering

Neither systematically applied to models of linguistic skills, except for Taatgen and Anderson (2002) (production compilation in morphology acquisition)

**Our contribution:** focus on the easier problem 2 (rule ordering) & show how Reinforcement Learning (RL, Sutton and Barto 2018) methods can be leveraged to formalize and partially solve it.

RL and ACT-R have close connections (Fu and Anderson 2006, Sutton and Barto 2018, Ch. 14), but they are largely unexplored.

# Road map

- 1 Introduction
- 2 Learning goal-conditioned rules in lexical decision**
- 3 Production-rule ordering as an RL problem
- 4 Simulations and results
- 5 Conclusion

# Learning goal-conditioned rules in lexical decision

- Parsing is too complex
- The range of issues that emerge when systematically integrating ACT-R and RL are best showcased with a simpler example
- We choose a basic learning algorithm: tabular  $Q$ -learning  
(a model-free, off-policy learning algorithm Watkins and Dayan 1992; Watkins 1989)
- Focus on a very simple task: lexical decision (LD)
  - participants see a string of letters on a screen
  - if they think the string is a word, they press one key (J)
  - if they think the string is not a word, they press a different key (F)
  - after pressing the key, the next stimulus is presented

## Why a model for LD, and what kind of model

- **Our specific goal:** 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 model for LD tasks is simple, so good starting point
- But it is a component of more complex syntactic and semantic parsing models, so can be scaled up
- The model provides the basic scaffolding of production rules needed for LD tasks, which is all we need here; **not** our focus:
  - fleshing out the model to capture major exp. results about LD
  - comparing it to previously proposed cognitive models of LD

## Three LD tasks

We model three LD tasks of increasing length, hence difficulty:

- 1 stimulus: the word *elephant*
- 2 stimuli: the word *elephant* and a non-word
- 4 stimuli: the word *elephant*, a non-word, the word *dog* and another non-word

The model components are split between:

- declarative memory: stores the lexical knowledge of an English speaker
- procedural memory: stores production rules that enable the model to carry out the LD task

# Four production rules

## Rule 1: Retrieving

goal>		STATE: retrieving	
visual>		VALUE: =val	
		VALUE: ~FINISHED	
⇒			
goal>		STATE: retrieval_done	
+retrieval>		ISA: word	
		FORM: =val	

## Rule 3: No Lexeme Found

goal>		STATE: retrieval_done	
retrieval>		BUFFER: empty	
		STATE: error	
⇒			
goal>		STATE: retrieving	
+manual>		CMD: press-key	
		KEY: F	

## Rule 2: Retrieval done

goal>		STATE: retrieval_done	
retrieval>		BUFFER: full	
		STATE: free	
⇒			
goal>		STATE: retrieving	
+manual>		CMD: press-key	
		KEY: J	

## Rule 4: Finished

goal>		STATE: retrieving	
visual>		VALUE: FINISHED	
⇒			
goal>		STATE: done	

Goal-state preconditions stricken out b/c Q-learning agent is supposed to learn them.



## Rule ordering in LD tasks

Assume:

- fully specified, hand-coded rules
- the initial goal STATE of the ACT-R model is `retrieving`
- the word *elephant* appears on the virtual screen of the model
- it is automatically stored in the VALUE slot of the visual buffer

The LD task unfolds as follows:

- 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
  - the goal STATE is updated to `retrieval_done`
- when the word is successfully retrieved, **Rule 2** fires and the `J` key is pressed

Then ...

# Rule ordering in LD tasks (ctd.)

## 1-stimulus task

- the text FINISHED is displayed on the screen
- **Rule 4** fires and ends the task

## 2-stimuli task

- the non-word is displayed, then **Rule 1** fires again
- the retrieval attempt fails (cannot retrieve non-word from dec. mem.), so **Rule 3** fires and the F key is pressed
- the text FINISHED is displayed and **Rule 4** fires to end the task

## 4-stimuli task

- first non-word displayed, **Rule 1** fires, **Rule 3** fires and the F key is pressed
- *dog* displayed, **Rule 1** fires, **Rule 2** fires and the J key is pressed
- second non-word is displayed, **Rule 1** fires, **Rule 3** fires and the F key to be pressed
- the text FINISHED is displayed and **Rule 4** fires to end the task

## Preview of task and results

The sequences of rule firings for the three LD tasks:

- 1-stimulus task: Rules **1 – 2 – 4**
- 2-stimuli task: Rules **1 – 2 – 1 – 3 – 4**
- 4-stimuli task: Rules **1 – 2 – 1 – 3 – 1 – 2 – 1 – 3 – 4**

We let the *Q*-learning agent learn to successfully carry out these LD tasks. The agent gets a reward of:

- 1 if it reaches the final goal-state *done*
- 0.15 for any intermediate rule firing, to encourage it to finish the task asap
- 0 (no penalty) if it chooses to wait and fire no rule
  - the optimal course of action when waiting for retrieval requests from declarative memory to complete, for example

## Preview of task and results (ctd.)

- The agent learns by trial and error how to properly order the rules and complete the LD tasks as efficiently as possible
- No small feat: the actual number of steps, i.e., decision points, when the agent needs to select an action, is larger than the high-level sequences of rule firings discussed above
- 1-stimulus task: 12 steps where the agent needs to decide whether to wait or to fire a specific rule; when the agent does not complete the task perfectly, much more than 12 steps
- 2-stimuli task: 18 steps if the task is completed perfectly
- 4-stimuli task: 34 steps if the task is completed perfectly

## Preview of task and results (ctd.)

### Why so many steps per task?

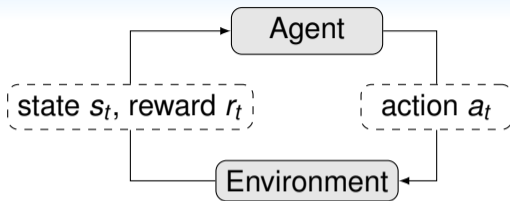
- LD simulations involve the visual and motor modules (to read strings of characters and to press keys) in addition to the declarative memory module
- Visual and motor actions, just as retrievals from declarative memory, take time, and the agent needs to make decisions while waiting for them to complete

The higher the number of steps, the harder the task is to learn:

- learning is faster and less noisy for shorter tasks (fewer stimuli)
- but the  $Q$ -learning agent learns even the most complex 4-stimuli task fairly well ...
- ... it just 'learns' a lot of noise in the process also
- parsing tasks much more challenging: many more rules, steps and states

- 1 Introduction
- 2 Learning goal-conditioned rules in lexical decision
- 3 Production-rule ordering as an RL problem**
- 4 Simulations and results
- 5 Conclusion

# Markov Decision Processes (MDPs)



- MDPs: deterministic or stochastic models of decision-making sequences, and the basis of RL approaches to learning
- An agent interacts with its environment and needs to make decisions at discrete time steps  $t = 1, 2, \dots, n$
- At every time step  $t$ , the current state  $s_t$  provides all info relevant for the current action selection (Markov property)
- The environment passes to the agent the state  $s_t$  and a reward signal  $r_t$
- The agent observes  $s_t$  and  $r_t$ , and takes an action  $a_t$ , which is passed from the agent to the environment; the cycle continues at time step  $t + 1$

## Markov Decision Processes (MDPs, ctd.)

- Agent's **policy**: complete specification of what action to take at any time step
- Given the Markov property, the policy  $\pi$  is a mapping from the state space  $S$  to the action space  $A$ ,  $\pi : S \rightarrow A$ .
  - a deterministic policy is a mapping from any given state  $s_t$  to an action  $a_t = \pi(s_t)$
  - a stochastic policy is a mapping from any given state  $s_t$  to a probability distribution over actions  $a_t \sim \pi(s_t)$
- **Agent's goal**: maximize some form of cumulative reward over an episode
- **Episode**: a complete, usually multi-step interaction between the agent and its environment
  - in our case, an episode would be a full simulation of a 1/2/4-stim LD task
- **Learning**: the agent learns, i.e., solves/optimizes the MDP, by updating its policy  $\pi$  to maximize the per-episode cumulative reward.



# Discounted return and state-action values

The discounted return  $G$  at a time step  $t < n$

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \cdots + \gamma^{n-t-1} r_n$$

- sum of current reward and discounted future rewards until the final step  $n$
- discount factor  $\gamma$  determines the present value of future rewards ( $0 \leq \gamma < 1$ )

The (state-)action value function  $Q_\pi(s, a)$

- the expected discounted return when starting in state  $s$ , performing action  $a$  and then following the policy  $\pi$  until the end of the episode

# Tabular Q-learning

Estimating the  $Q$  function is one way to find an optimal policy

- improve estimate based on interactions between agent and environment
- optimal policy (that maximizes return): choose a max-value action in any state
- we represent the  $Q$  function  $S \times A \rightarrow \mathbb{R}$  as a look-up table storing the estimated values of all possible state-action pairs
  - the  $Q$  table is initialized to an arbitrary fixed value, usually 0
- at each time step  $t$ , we update the entry for  $(s_t, a_t)$  in the  $Q$  table based on the info the agent gets from the environment at the next step, namely:
  - the reward  $r_{t+1}$
  - the new state  $s_{t+1}$ , for which we can compute a value estimate based on our current  $Q$  table

## Q-learning for production selection

In our ACT-R model of LD tasks:

- the agent: Q-value table guiding rule selection at every cognitive step
- the environment: the cognitive state of the ACT-R model/mind
- states that the environment passes to the agent consist of:
  - the current goal buffer
  - the current retrieval buffer
  - the value in the visual-what buffer, if any
  - the state of the manual buffer (busy or free)
- the action space:
  - the 4 rules *retrieving*, *lexeme retrieved*, *no lexeme found* and *finished*
  - a special action *None*: the agent selects it when it prefers to wait for a new state

## Reward structure

Rewards encourage the agent to finish the task asap & select fewest rules in the process:

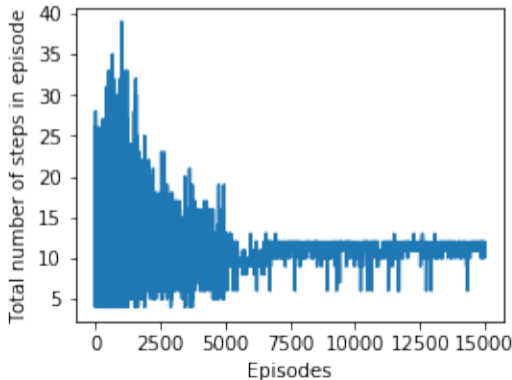
- positive reward 1 when the LD task is completed (the goal STATE is *done*)
- negative reward  $-0.15$  for every rule it selects other than *None*
- no penalty for selecting the special action *None*;
- at every step, a negative reward equal to the amount of time that has elapsed between the immediately preceding step and the current step (multiplied by  $-1$  to make it negative)
- the negative temporal reward discourages the agent from just repeatedly selecting the special action *None*

this quickly times out the LD task and fast-forwards the agent to the maximum waiting time the ACT-R environment allows for – set to 2 s per stim here

- 1 Introduction
- 2 Learning goal-conditioned rules in lexical decision
- 3 Production-rule ordering as an RL problem
- 4 Simulations and results**
- 5 Conclusion

## One-stimulus task

- $\epsilon$ -greedy policy for all simulations,  $\epsilon$  annealed from 1 to a min value of 0.01
- we simulate 15,000 episodes (LD decision tasks consisting of 1 stim only – the word ‘elephant’)
- agent learns to complete task perfectly in 12 steps after 5,000 episodes
- fewer steps when agent times out task, e.g., repeatedly selects action *None*



## One-stimulus task: final Q-table

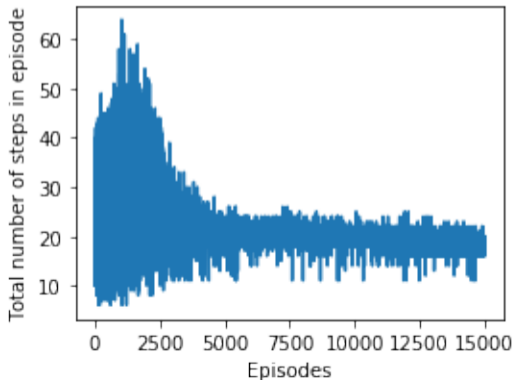
- Look only at states for which at least one action/rule has a non-0 value – a total of 8 states
- For each state, identify the action/rule with the highest value
- 3 states in which agent fires no rule (*None* has max-value)
  - waiting for text to be auto-detected and stored in vis. buffer  
goal: {state: retrieving}, manual: free/busy, retrieval: {}, vis\_val: NO\_VAL
  - waiting for retrieval process to complete  
goal: {state: retrieval\_done}, manual: free, retrieval: {}, vis\_val: elephant
- 3 states where max-value action is *finished*; in all of them, the visual value is the text FINISHED
- The last 2 state-action pairs:
  - goal: {state: retrieving}, manual: free, retrieval: {}, visual\_value: elephant  $\Rightarrow$  *retrieving*  
(correctly starts the retrieval process)
  - goal: {state: retrieval\_done}, manual: free, retrieval: {form: elephant}, visual\_value: elephant  $\Rightarrow$  *lexeme retrieved* (correctly presses  $\perp$  key when retrieval successful)

Thus: no need to hand-code goal states in rule preconditions.

Q-learning agent learns by trial-and-error when to fire which rule, and when to wait.

## Two-stimuli task: final Q-table

- 15,000 episodes, task completed perfectly (18 steps) after 9,000 episodes
- Final Q-table: 13 states with at least one non-0-value action
- 4 states where the agent does nothing: waiting for retrieval process to complete (for word 'elephant' or non-word), and waiting for visual value to be stored in visual buffer (whether manual buffer is busy or free)



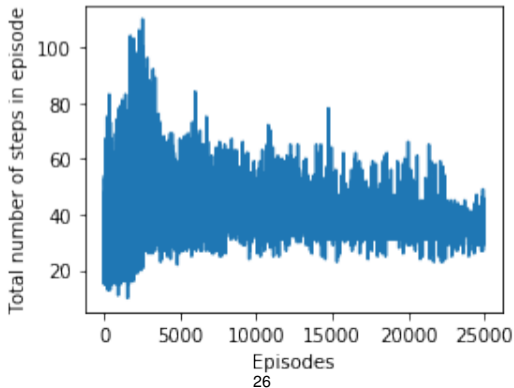


## Two-stimuli task: final Q-table (ctd.)

- 4 states where the agent fires the *finished* rule correctly because the visual value is FINISHED in all of them
- 4 more state-action pairs are exactly what we expect:
  - goal state *retrieving*, the retrieval buffer is empty and the manual buffer is free  $\Rightarrow$  *retrieving*  
whether the visual value is *elephant* or the non-word
  - goal state *retrieval\_done* and retrieval process successful (based on visual value *elephant*)  $\Rightarrow$  *lexeme retrieved*
  - goal state *retrieval\_done* and retrieval process unsuccessful (based on the non-word visual value)  $\Rightarrow$  *no lexeme found*
- 1 non-optimal state-action pair:
  - goal state *retrieval\_done*, retrieval buffer contains *elephant*, visual value is the non-word  $\Rightarrow$  *retrieving*
  - reflects the trial-and-error learning process, which is more error prone than for the simpler 1-stimulus task

## Four-stimuli task: final Q-table

- 25,000 episodes, task completed reliably (less than 40 steps) after 22,000 episodes
- Even after 25,000 episodes, agent still tries incorrect rules, waits for no good reason
- Final Q-table: 24 states with at least one non-0-value action
  - 18 are exactly what we expect
  - 6 reflect the noise in the trial-and-error learning process



# Conclusion

We've shown that:

- the learnability problem for production-based cognitive models can be systematically formulated and computationally addressed as an RL problem

But this is merely a first inroad into a rich nexus of issues:

- not cognitively realistic to require a high number of episodes (trial-and-error interactions) for learning to happen

what specifically in the human cognitive architecture enables us to learn from much fewer interactions?

- there are other RL learning algorithms, e.g., (Expected) Sarsa, policy based approaches, neural-network function approximation

how do these algorithms perform on LD decision tasks?

- how do all these different RL algorithms perform on a variety of production-based cognitive models?









whether the models are linguistic, e.g., syntactic or semantic parsing, or non-linguistic

## 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 I

- 
- Anderson, John R. (2007). *How can the human mind occur in the physical universe?* Oxford University Press.
- 
- Anderson, John R. and Christian Lebiere (1998). *The Atomic Components of Thought*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- 
- Brasoveanu, Adrian and Jakub Dotlačil (2020). *Computational Cognitive Modeling and Linguistic Theory*. Language, Cognition, and Mind (LCAM) Series. Springer (Open Access). DOI: <https://doi.org/10.1007/978-3-030-31846-8>.
- 
- Engelmann, Felix (2016). "Toward an integrated model of sentence processing in reading". PhD thesis. Potsdam: University of Potsdam.
- 
- Fu, Wai-Tat and John R. Anderson (2006). "From recurrent choice to skill learning: A reinforcement-learning model". In: *Journal of Experimental Psychology: General* 135.2, pp. 184–206. DOI: 10.1037/0096-3445.135.2.184.
- 
- Hale, John (2011). "What a rational parser would do". In: *Cognitive Science* 35, pp. 399–443.
- 
- Lewis, Richard and Shravan Vasishth (2005). "An activation-based model of sentence processing as skilled memory retrieval". In: *Cognitive Science* 29, pp. 1–45.
- 
- Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

## References II



Taatgen, Niels A. and John R. Anderson (2002). “Why do children learn to say “broke”? A model of learning the past tense without feedback”. In: *Cognition* 86.2, pp. 123–155.



Watkins, Christopher J. C. H. and Peter Dayan (1992). “Q-learning”. In: *Machine Learning* 8.3, pp. 279–292. DOI: 10.1007/BF00992698. URL: <https://doi.org/10.1007/BF00992698>.



Watkins, Christopher John Cornish Hellaby (1989). “Learning from Delayed Rewards”. PhD thesis. Cambridge, UK: King’s College. URL: [http://www.cs.rhul.ac.uk/~chrisw/new\\_thesis.pdf](http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf).