Reinforcement Learning for Production-based Cognitive Models

Adrian Brasoveanu & Jakub Dotlačil
UC Santa Cruz & Utrecht University

AMLaP · September 5, 2020
Decision making

• How do we make decisions?
• How do we learn to make decisions in situations with recurring choices?
Decision making

• How do we make decisions?
• How do we learn to make decisions in situations with recurring choices?
Decision making

• How do we make decisions?
• How do we learn to make decisions in situations with recurring choices?

Decision making and processing

• After visual encoding of a word, I can choose from:
  1. moving attention  2. lex. retrieval  3. syn. integration

  What should I choose first?
  What sequence of actions should I choose?
Decision making

- How do we make decisions?
- How do we learn to make decisions in situations with recurring choices?

Decision making and processing

- After visual encoding of a word, I can choose from:
  1. moving attention
  2. lex. retrieval
  3. syn. integration

  What should I choose first?
  What sequence of actions should I choose?

- In parsing, what interpretation should I pursue?
The horse raced past the barn fell.  \( \sim \) Main Clause Int.  
\( \sim \) Reduced Rel. Int.
Production-system frameworks and decision making

- Production systems in cognitive sciences and psychology (Newell, 1973)
- Productions: conditionalized actions (actions that fire only if particular conditions are met) (ACT-R, SOAR)
• Production systems in cognitive sciences and psychology (Newell, 1973)

• Productions: conditionalized actions 
  (actions that fire only if particular conditions are met) 
  (ACT-R, SOAR)

• Production systems in psycholinguistics: 
  cognitive models in ACT-R, SOAR 
  Brasoveanu and Dotlačil, 2020; Hale, 2014; Lewis and Vasishth, 2005
• Production systems in cognitive sciences and psychology (Newell, 1973)

• Productions: conditionalized actions (actions that fire only if particular conditions are met) (ACT-R, SOAR)

• Production systems in psycholinguistics: cognitive models in ACT-R, SOAR Brasoveanu and Dotlačil, 2020; Hale, 2014; Lewis and Vasisht, 2005

Can the agent acquire such production systems?
• Production systems in cognitive sciences and psychology (Newell, 1973)

• Productions: conditionalized actions (actions that fire only if particular conditions are met) (ACT-R, SOAR)

• Production systems in psycholinguistics: cognitive models in ACT-R, SOAR
  Brasoveanu and Dotlačil, 2020; Hale, 2014; Lewis and Vasishth, 2005

Can the agent acquire such production systems?

• Can the agent learn which decisions to make under which conditions?
  Fu and Anderson (2006), Sutton and Barto (2018, Ch. 14)
Can the agent acquire such production systems?

**Our contribution:** show how Reinforcement Learning (RL, Sutton and Barto 2018) methods can be combined with a production system (ACT-R) to learn sequential-choice behavior.

Explore two RL algorithms: tabular $Q$-learning and Deep $Q$-networks.
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

Focus on a 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 tabular $Q$-learning and the Deep $Q$-network agent can learn the order of productions in LD tasks
Why a model for LD, and what kind of model

- **Our specific goal**: investigate the extent to which tabular \(Q\)-learning and the Deep \(Q\)-network agent can learn the order of productions in 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 it is relevant when we scale up

- Not our focus: fleshing out the model to capture major exp. results about LD (cf. Brasoveanu and Dotlačil 2020) comparing it to previously proposed cognitive models of LD
Why a model for LD, and what kind of model

- **Our specific goal**: investigate the extent to which tabular $Q$-learning and the Deep $Q$-network agent can learn the order of productions in 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 it is relevant when we scale up

- **Not our focus**:
  - fleshing out the model to capture major exp. results about LD (cf. Brasoveanu and Dotlačil 2020)
  - comparing it to previously proposed cognitive models of LD
Three LD tasks

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

- 1 stimulus: a word (elephant)
- 2 stimuli: a word (elephant) and a non-word
- 4 stimuli: a word (elephant), a non-word, another word (dog) and another non-word
we model three LD tasks of increasing length, hence difficulty:

- 1 stimulus: a word (elephant)
- 2 stimuli: a word (elephant) and a non-word
- 4 stimuli: a word (elephant), a non-word, another 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

1. If the goal is to retrieve lex. information and the visual buffer has a string of letters
   
   Attempt to retrieve that string from declarative memory

Q-learning agent has to learn the conditions (what precedes $\Rightarrow$).
Four production rules

1. If the goal is to retrieve lex. information and the visual buffer has a string of letters
   \[\Rightarrow\]
   Attempt to retrieve that string from declarative memory

2. If the retrieval is done and it is successful
   \[\Rightarrow\]
   Press the success key (J)

The Q-learning agent has to learn the conditions (what \(\Rightarrow\)).
Four production rules

1. If the goal is to retrieve lex. information and the visual buffer has a string of letters
   \[\Rightarrow\]
   Attempt to retrieve that string from declarative memory

2. If the retrieval is done and it is successful
   \[\Rightarrow\]
   Press the success key (J)

3. If the retrieval is done and it is a failure
   \[\Rightarrow\]
   Press the failure key (F)

4. If there is a string 'FINISHED' in the visual buffer
   \[\Rightarrow\]
   Done
Four production rules

1. If the goal is to retrieve lex. information and the visual buffer has a string of letters
   
   Attempt to retrieve that string from declarative memory

2. If the retrieval is done and it is successful
   
   Press the success key (J)

3. If the retrieval is done and it is a failure
   
   Press the failure key (F)

4. If there is a string ‘FINISHED’ in the visual buffer
   
   Done

The Q-learning agent has to learn the conditions (what precedes ⇒).
Four production rules

1. If the goal is to retrieve lex. information and the visual buffer has a string of letters
   
   Attempt to retrieve that string from declarative memory

2. If the retrieval is done and it is successful
   
   Press the success key (J)

3. If the retrieval is done and it is a failure
   
   Press the failure key (F)

4. If there is a string ‘FINISHED’ in the visual buffer
   
   Done

The Q-learning agent has to learn the conditions (what precedes).
Four production rules

1. If the goal is to retrieve lex. information and the visual buffer has a string of letters

   Attempt to retrieve that string from declarative memory

2. If the retrieval is done and it is successful

   Press the success key (J)

3. If the retrieval is done and it is a failure

   Press the failure key (F)

4. If there is a string ‘FINISHED’ in the visual buffer

   Done

The $Q$-learning agent has to learn the conditions (what precedes $\Rightarrow$).
The sequences of rule firings for the learned LD tasks:

- 1-stim: Rules $[1 - 2] - 4$
The sequences of rule firings for the learned LD tasks:

- 1-stim: Rules $[1 - 2] - 4$

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 $\text{done}$
- $-0.15$ 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 agent learns by trial and error how to properly order the rules and complete the LD tasks as efficiently as possible.

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.
The agent learns by trial and error how to properly order the rules and complete the LD tasks as efficiently as possible.

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
- 2-stimuli task: 18 steps if task completed perfectly
- 4-stimuli task: 34 steps if task completed perfectly
Why so many steps per task?

- Several points where agent should wait (retrieval, waiting for key press to complete, while encoding visual information)

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 RL agents learn even the most complex 4-stimuli task fairly well (at least the tabular $Q$ agent) ...
- ...they just ‘learn’ a lot of noise (incorrect rules) in the process also, particularly the neural-network agents
Introduction

Learning goal-conditioned rules in lexical decision

Production-rule ordering as an RL problem

Simulations and results

Conclusion
• MDPs: stochastic models of sequential decision making, and the basis of RL approaches to learning
• Agent interacts with environment, needs to make decisions at discrete time steps $t = 1, 2, \ldots, n$
Markov Decision Processes (MDPs)

- MDPs: stochastic models of sequential decision making, and the basis of RL approaches to learning
- Agent interacts with environment, needs to make decisions at discrete time steps \( t = 1, 2, \ldots, n \)
- At every time step \( t \): current state \( s_t \) provides all info relevant for the current action selection (Markov property)
- Environment passes to agent state \( s_t \) and reward signal \( r_t \)
Markov Decision Processes (MDPs)

- MDPs: stochastic models of sequential decision making, and the basis of RL approaches to learning
- Agent interacts with environment, needs to make decisions at discrete time steps \( t = 1, 2, \ldots, n \)
- At every time step \( t \): current state \( s_t \) provides all info relevant for the current action selection (Markov property)
- Environment passes to agent state \( s_t \) and reward signal \( r_t \)
- Agent observes \( s_t \) and \( r_t \), and takes action \( a_t \), which is passed from agent to environment; 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
- A stochastic policy $\pi$ is a mapping from any given state $s_t$ to a probability distribution over actions $a_t \sim \pi(s_t)$
• Agent’s **policy**: complete specification of what action to take at any time step

• A stochastic policy $\pi$ 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 cumulative reward over LD task
Markov Decision Processes (MDPs, ctd.)

- Agent’s **policy**: complete specification of what action to take at any time step
- A stochastic policy $\pi$ 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 cumulative reward over LD task
- Learning: agent learns (solves/optimizes the MDP) by updating its policy $\pi$ to maximize 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 ($0 \leq \gamma \leq 1$)
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 ($0 \leq \gamma \leq 1$)

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

$Q$ function provides expected discounted return when:
- starting in state $s$,
- performing action $a$,
- following the policy $\pi$ until the end of the episode.
Estimating the $Q$ function is one way to find an optimal policy

- optimal policy: choose max-value action in any state
- $Q$ estimated based on experience (interactions between agent and environment)
Tabular $Q$-learning

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

- optimal policy: choose max-value action in any state
- $Q$ estimated based on experience (interactions between agent and environment)
- tabular $Q$ learning: $Q$ function $S \times A \rightarrow \mathbb{R}$ represented as look-up table storing estimated values of all state-action pairs
  - $Q$ table initialized to 0
Tabular $Q$-learning

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

- optimal policy: choose max-value action in any state
- $Q$ estimated based on experience (interactions between agent and environment)
- tabular $Q$ learning: $Q$ function $S \times A \rightarrow \mathbb{R}$ represented as look-up table storing estimated values of all state-action pairs
  - $Q$ table initialized to 0
- at each time step $t$: update entry for $(s_t, a_t)$ based on info agent gets from environment at next step
  - reward $r_{t+1}$
  - new state $s_{t+1}$ (its value estimated from current $Q$ table)
$Q$-learning for production selection

In our ACT-R model of LD tasks:

- **agent**: $Q$-value table guiding rule selection at every cognitive step
In our ACT-R model of LD tasks:

- **agent:** $Q$-value table guiding rule selection at every cognitive step

- **environment:** cognitive state of ACT-R model/mind
In our ACT-R model of LD tasks:

- **agent**: $Q$-value table guiding rule selection at every cognitive step

- **environment**: cognitive state of ACT-R model/mind

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

- positive reward 1 when LD task is completed
- negative reward $-0.15$ for every rule other than None
- no penalty for None
Rewards encourage the agent to finish the task asap & select fewest rules in the process:

- positive reward 1 when LD task is completed
- negative reward \(-0.15\) for every rule other than None
- no penalty for None

- at every step, negative temporal reward: time elapsed between immediately preceding and current step
  - negative temporal reward discourages agent from repeatedly selecting an action (None) and timing out task in small number of steps
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

• we simulate 15,000 episodes (LD tasks consisting of 1 stim only – the word ‘elephant’)
One-stimulus task

- we simulate 15,000 episodes (LD 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 rule has non-0 value – a total of 8 states
- For each state, identify rule with highest value

Thus: no need to hand-code goal states in rule preconditions. (at least for tabular $Q$ in this very simple task)
One-stimulus task: final $Q$-table

- Look only at states for which at least one rule has non-0 value – a total of 8 states
- For each state, identify rule with highest value
- 3 states in which agent fires no rule (None has max-value)
  - waiting for text to be read off the virtual screen
  - waiting for retrieval process to complete
- Last 2 state-action pairs:
  - correctly start the retrieval process as soon as the text is read off the virtual screen
  - correctly press the J key when retrieval is successful

Thus: no need to hand-code goal states in rule preconditions. (at least for tabular $Q$-table in this very simple task)
One-stimulus task: final $Q$-table

- Look only at states for which at least one rule has non-0 value – a total of 8 states
- For each state, identify rule with highest value
- 3 states in which agent fires no rule (None has max-value)
  - waiting for text to be read off the virtual screen
  - waiting for retrieval process to complete
- 3 states where max-value rule is finished; in all of them, text on the virtual screen is FINISHED
One-stimulus task: final $Q$-table

- Look only at states for which at least one rule has non-0 value – a total of 8 states
- For each state, identify rule with highest value
- 3 states in which agent fires no rule (None has max-value)
  - waiting for text to be read off the virtual screen
  - waiting for retrieval process to complete
- 3 states where max-value rule is finished; in all of them, text on the virtual screen is FINISHED
- Last 2 state-action pairs:
  - correctly start the retrieval process as soon as the text is read off the virtual screen
  - correctly press the J key when retrieval is successful

Thus: no need to hand-code goal states in rule preconditions. (at least for tabular $Q$ in this very simple task)
One-stimulus task: final $Q$-table

- Look only at states for which at least one rule has non-0 value – a total of 8 states
- For each state, identify rule with highest value
- 3 states in which agent fires no rule (None has max-value)
  - waiting for text to be read off the virtual screen
  - waiting for retrieval process to complete
- 3 states where max-value rule is finished; in all of them, text on the virtual screen is FINISHED
- Last 2 state-action pairs:
  - correctly start the retrieval process as soon as the text is read off the virtual screen
  - correctly press the J key when retrieval is successful

Thus: no need to hand-code goal states in rule preconditions. (at least for tabular $Q$ in this very simple task)
Two-stimuli task: final $Q$-table

- 15,000 episodes, task completed perfectly (18 steps) after 9,000 episodes
Two-stimuli task: final $Q$-table

- 15,000 episodes, task completed perfectly (18 steps) after 9,000 episodes

- Final $Q$-table: 13 states with non-0-value rules
Two-stimuli task: final $Q$-table

- 15,000 episodes, task completed perfectly (18 steps) after 9,000 episodes

- Final $Q$-table: 13 states with non-0-value rules
- 4 states where the agent does nothing
  - waiting for retrieval process to complete
  - waiting for text to be read off the virtual screen
• 4 states where agent fires finished correctly because text on the virtual screen is FINISHED
Two-stimuli task: final $Q$-table (ctd.)

• 4 states where agent fires finished correctly because text on the virtual screen is FINISHED

• 4 state-action pairs are exactly what we expect:
  • trigger retrieving as soon as text is read off virtual screen
  • trigger lexeme retrieved when retrieval process successful
  • trigger no lexeme found when retrieval process unsuccessful
Two-stimuli task: final $Q$-table (ctd.)

- 4 states where agent fires finished correctly because text on the virtual screen is FINISHED
- 4 state-action pairs are exactly what we expect:
  - trigger retrieving as soon as text is read off virtual screen
  - trigger lexeme retrieved when retrieval process successful
  - trigger no lexeme found when retrieval process unsuccessful
- 1 state-action pair that reflects trial-and-error learning process
  - state: previous stimulus not fully processed, but new stimulus already read off virtual screen
  - action: agent attempts to retrieve
Four-stimuli task: final $Q$-table

- 25,000 episodes, task completed fairly well after 22,000 episodes
Four-stimuli task: final $Q$-table

- 25,000 episodes, task completed fairly well after 22,000 episodes

- Even after 25,000 episodes, agent still tries incorrect rules, waits for no good reason
Four-stimuli task: final $Q$-table

- 25,000 episodes, task completed fairly well 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
Deep $Q$-Networks (DQN) in LD tasks

- we use a neural network (multilayer perceptron, one hidden layer) to approximate the $Q$-function
Deep $Q$-Networks (DQN) in LD tasks

- we use a neural network (multilayer perceptron, one hidden layer) to approximate the $Q$-function
- DQN takes longer to learn the 1-stim task ($\approx 8,000$ episodes), does not reliably learn 2-stim task, but learns the 4-stim task much faster than tabular $Q$-learning ($\approx 2,000$ episodes)

DQN: 1-stim

DQN: 2-stim

DQN: 4-stim
Deep $Q$-Networks (DQN) in LD tasks

- we use a neural network (multilayer perceptron, one hidden layer) to approximate the $Q$-function
- DQN takes longer to learn the 1-stim task ($\approx 8,000$ episodes), does not reliably learn 2-stim task, but learns the 4-stim task much faster than tabular $Q$-learning ($\approx 2,000$ episodes)

DQN: 1-stim

DQN: 2-stim

DQN: 4-stim

- DQN generalizes much more aggressively, which might be why it is good at the more difficult 4-stim task
Deep $Q$-Networks (DQN) in LD tasks

- we use a neural network (multilayer perceptron, one hidden layer) to approximate the $Q$-function
- DQN takes longer to learn the 1-stim task ($\approx 8,000$ episodes), does not reliably learn 2-stim task, but learns the 4-stim task much faster than tabular $Q$-learning ($\approx 2,000$ episodes)
  
  **DQN: 1-stim**

  ![](image1.png)

  **DQN: 2-stim**

  ![](image2.png)

  **DQN: 4-stim**

  ![](image3.png)

- DQN generalizes much more aggressively, which might be why it is good at the more difficult 4-stim task
- but even for 4-stim, it only partially learns when to trigger rules, and learns a lot more noise (incorrect rules) than tabular $Q$
Conclusion

We’ve shown that:

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

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

- what specifically in the human cognitive architecture enables us to learn from much fewer interactions?
- there are other RL algorithms, e.g., (Expected) Sarsa, policy-based approaches etc.; how do these algorithms perform on LD tasks?
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:

- what specifically in the human cognitive architecture enables us to learn from much fewer interactions?
- there are other RL algorithms, e.g., (Expected) Sarsa, policy-based approaches etc.; how do these algorithms perform on LD tasks?
- how do all these different RL algorithms perform on a variety of production-based cognitive models?
We are grateful to four anonymous AMLaP 2020 reviewers and the audience of the UCSC Linguistics Department S-circle (May 2020) for their questions and feedback.

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.


