Exploring Continuous RL Algorithms for Soar-RL

Explore CS Undergraduate Research Experience

Researchers: Xinyi Zhu, Jason Lin

Mentor: Robert Wray

45th Soar Workshop

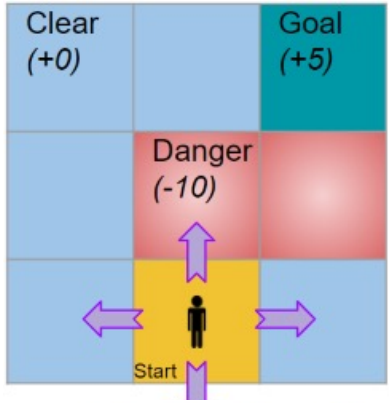
5 May 2025

Problem

Identify alternative choices of RL algorithm for use in Soar

Motivation:

- Soar-RL uses an implementation of Q-learning
- Q-learning requires that continuous spaces be discretized for learning
  - Scales poorly (table size)
  - Results depend on details of *a priori* discretization



	a1	a2	a3	a4
S1	0	0	0	0
S2	0	0	0	0
S3	0	0	0	0
S4	0	0	0	0
S5	0	0	0	0

(images: Ketan Doshi, medium.com)

Objectives / Requirements

Limitations of Q-learning → Many continuous versions of RL in use today

- Are there better options than Q-learning for Soar?

Requirements:

1. Support discrete/continuous state representation
2. Accommodate Soar's distributed action-reward representation

Soar-specific desiderata:

- Sample efficient learning: Cannot assume 1000s of opportunities to learn effective policy
- Online, incremental and experiential learning: Avoid algorithms requiring offline analysis
- Dynamic variable selection: Prefer run-time commitment to relevant variables

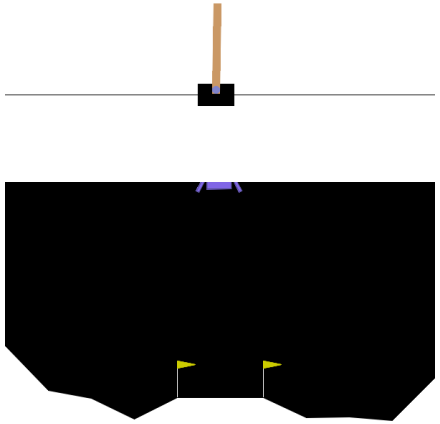
Two Paths of Research

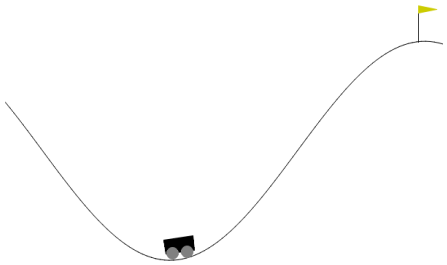
1. Survey of RL Methods
   - Characterize learning performance (somewhat independent of Soar requirements)

2. Modify (base) Deep Q-Networks (DQN)
   - Better understand/evaluate design requirements

- Research Environment: Gymnasium (nee OpenAI Gym)
  - Cartpole, Lunar Lander, Mountain Car
  - Took advantage of Soar/Gym interface

  Thanks, Tim Saucer! (https://github.com/timsaucer/SoarGym)



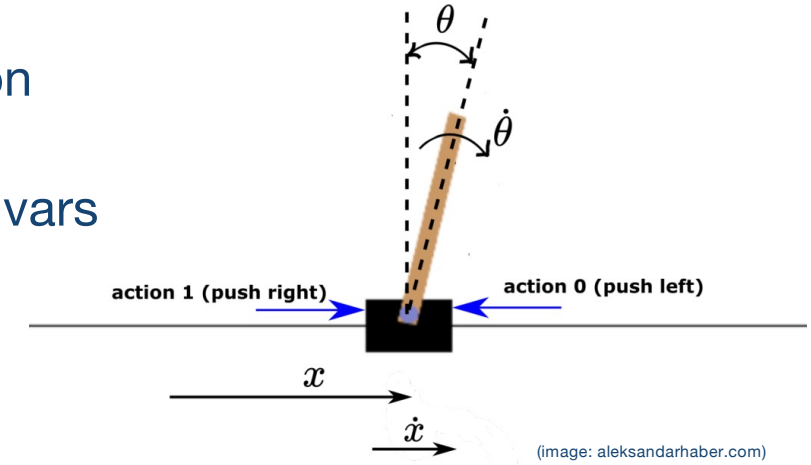


(images: Gynasium Documentation)

Combinations of Discrete and Continuous Variables

- We chose environments with discrete action spaces ("push left")
- Want mix of discrete and continuous state vars
- Approach: Discretize some state variables

Discrete Values		Q
d_p	d_c	
left	left	$f_{ll}(\theta, \dot{\theta}, x, \dot{x})$
left	right	$f_{lr}(\theta, \dot{\theta}, x, \dot{x})$
right	left	$f_{rl}(\theta, \dot{\theta}, x, \dot{x})$
right	right	$f_{rr}(\theta, \dot{\theta}, x, \dot{x})$



Variations:

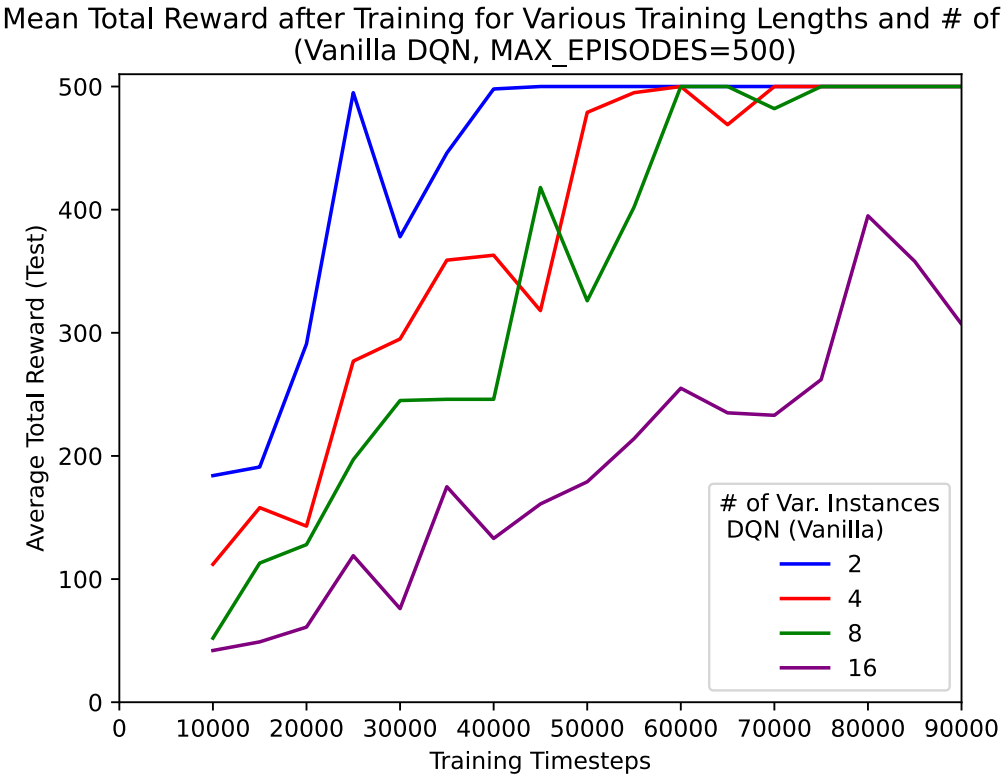
- 1-4 discrete variables (8 variable instances)
- Discrete & continuous variable unchanged
- Discrete & Abs(continuous variable)
- Add additional random variables

Survey of RL Methods

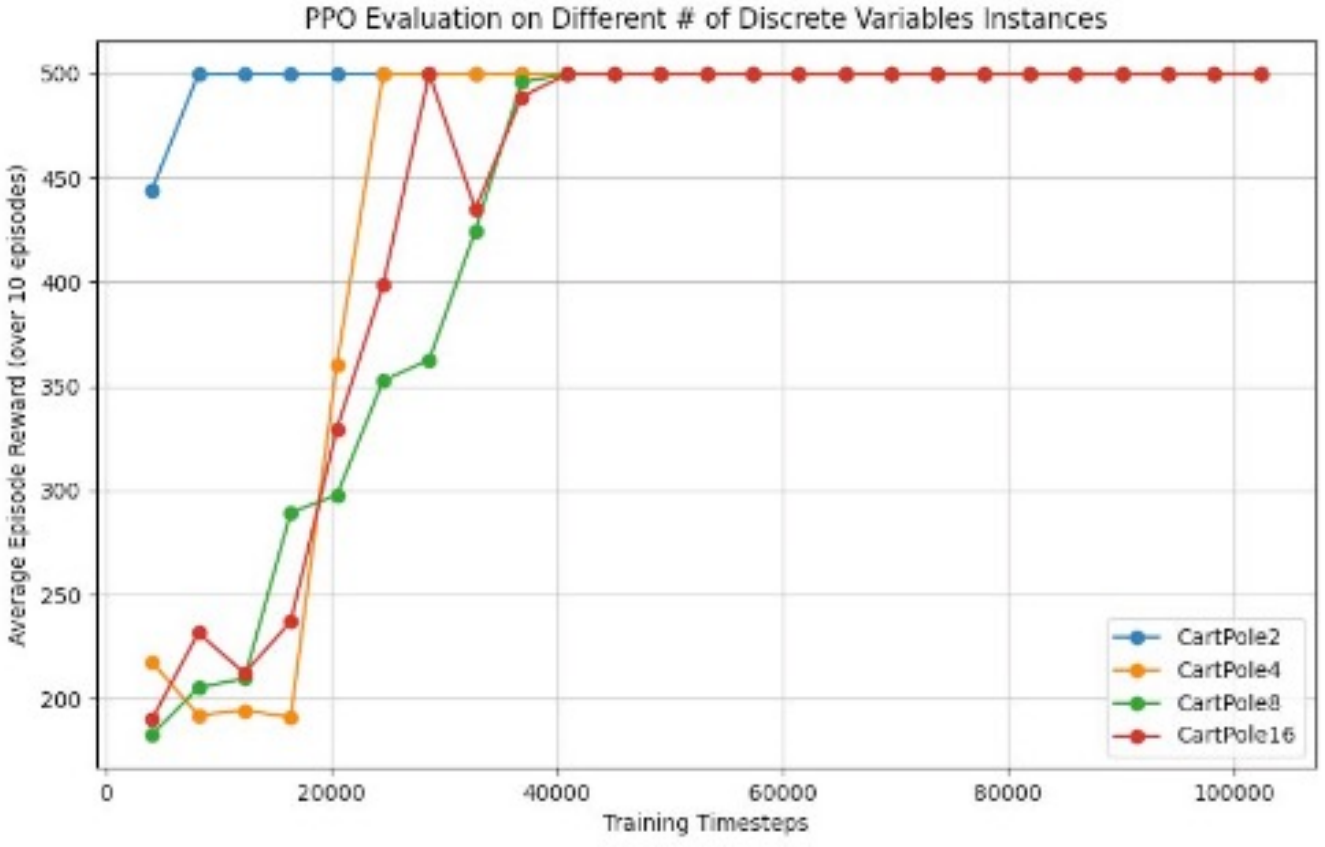
- Goal: Characterize sensitivity & scaling performance of RL algorithms
- Relax Soar-specific requirements:
  - Wider survey of RL algorithms:
  1. Proximal policy optimization (PPO): Sample efficient, stable, batch method
  2. Advantage Actor-Critic (A2C): Sample efficient, batch method, learns distinct value function
  3. Double DQN (DDQN): Small batch method, simple change to DQN that improve stability
  - Static one-hot encoding of discrete variables
- Standardized size of state space across environments
  - 6 continuous variables (including random variables where needed)
  - 1/2/4/8 binary (discrete) variables (→ 2/4/8/16 discrete variable instances)

Outcomes/Expectations

- Compare cumulative reward vs. timesteps for combinations of algorithms and # of discrete variable instances

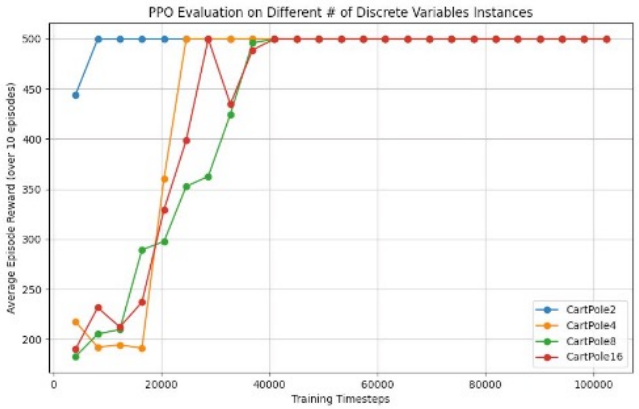


Example: Cartpole PPO

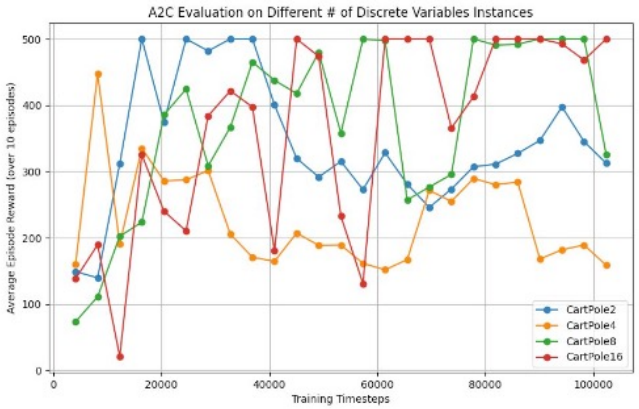


Cartpole: Three Methods

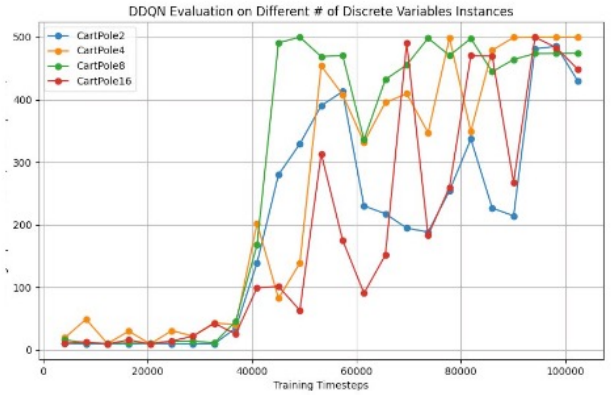
PPO



A2C



DDQN



- PPO learns relatively quickly (most sample efficient)
- PPO somewhat exhibits expected scaling dynamic (more variables → longer to learn)
- A2C exhibits volatility in learning, reverse scaling?
- DDQN learns very (very) slowly, scaling unclear

Cartpole vs. Lunar Lander

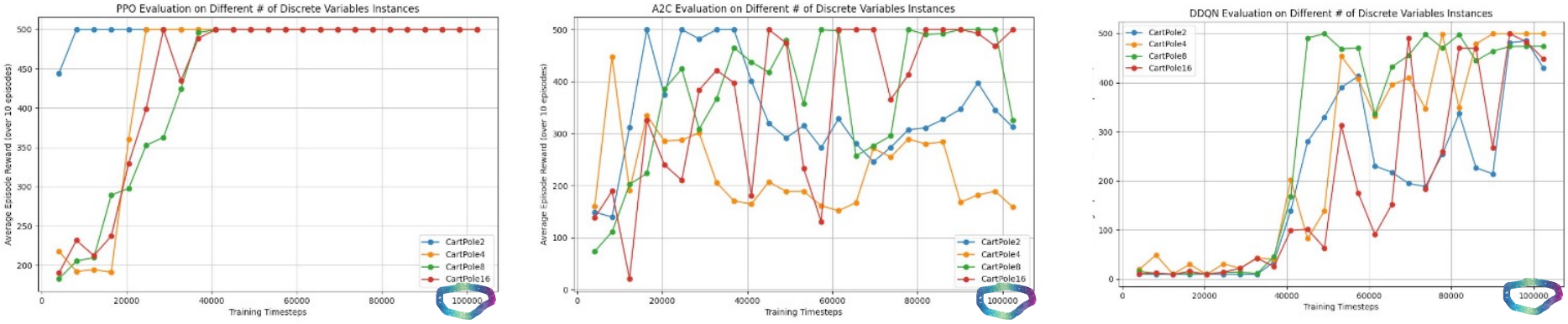


Figure 1. Survey of PPO, A2C, and DDQN on CartPole with varying # of discrete variables

PPO A2C DDQN

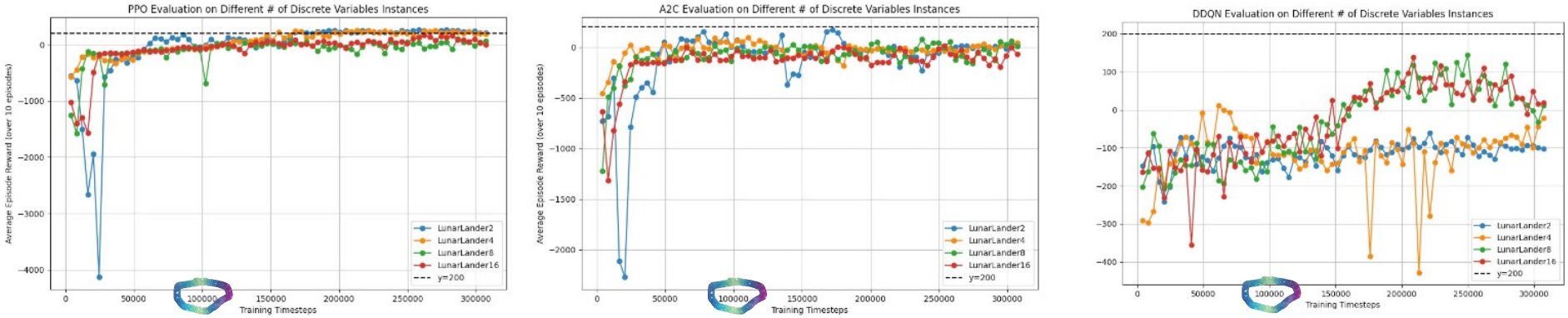
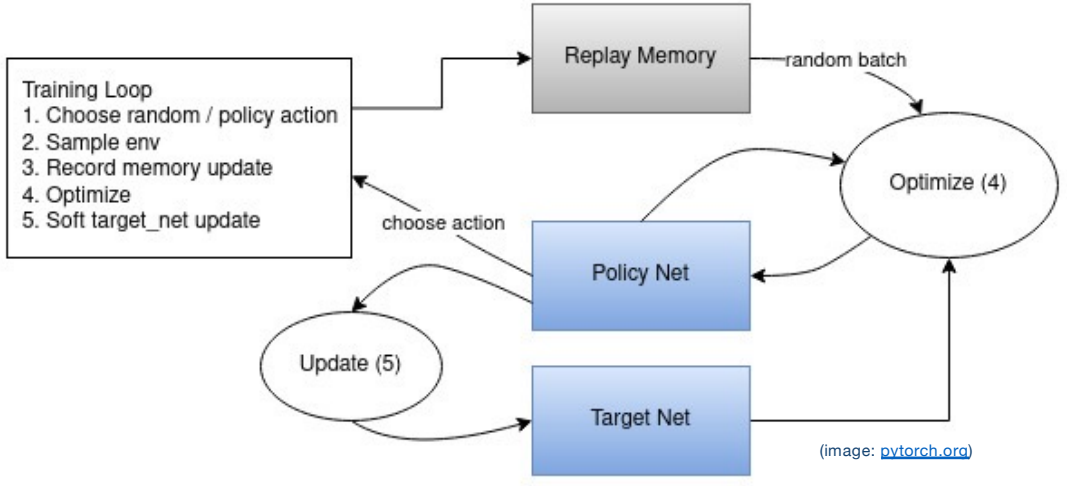


Figure 2. Survey of PPO, A2C, and DDQN on Lunar Lander with varying # of discrete variables

Deep Q Network (DQN) Exploration

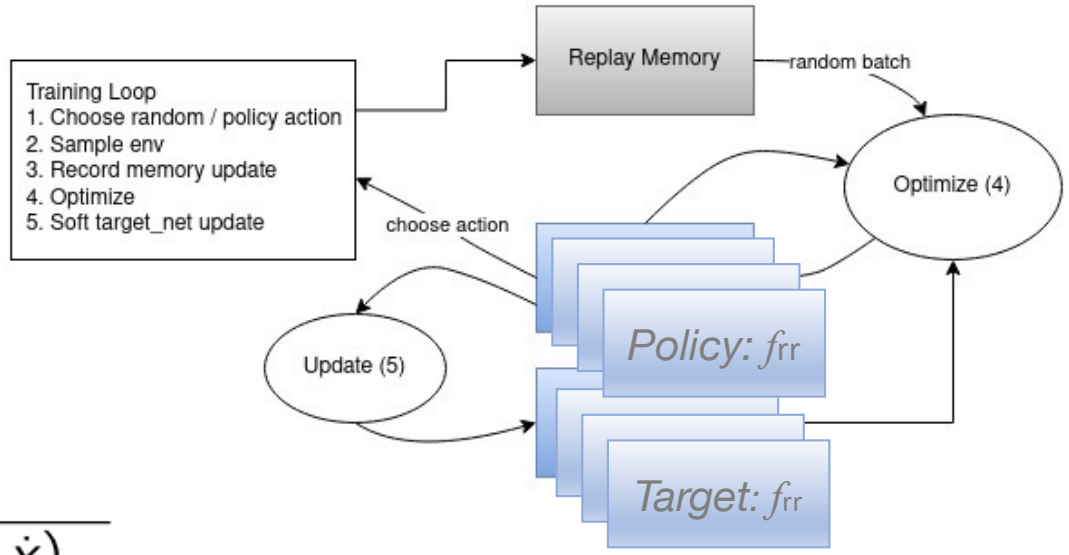
- DQN: Developed by DeepMind (e.g., Atari game playing)
- Closely conforms to Q-learning problem formulation
  - Instead of Q table, DQN learns a function $f(S,a)=Q$
  - Takes advantage of NNs as good function approximators
- Simplest "step" from Q-learning to continuous RL for Soar?



- Experience replay: Store of observations
- Policy Net: Network used to take actions
- Target Net: "Fixed" network for computing differences
- Batch mode?
  - Soft updates make it feasible to update on every cycle

Architecture Inspired by Soar Requirements

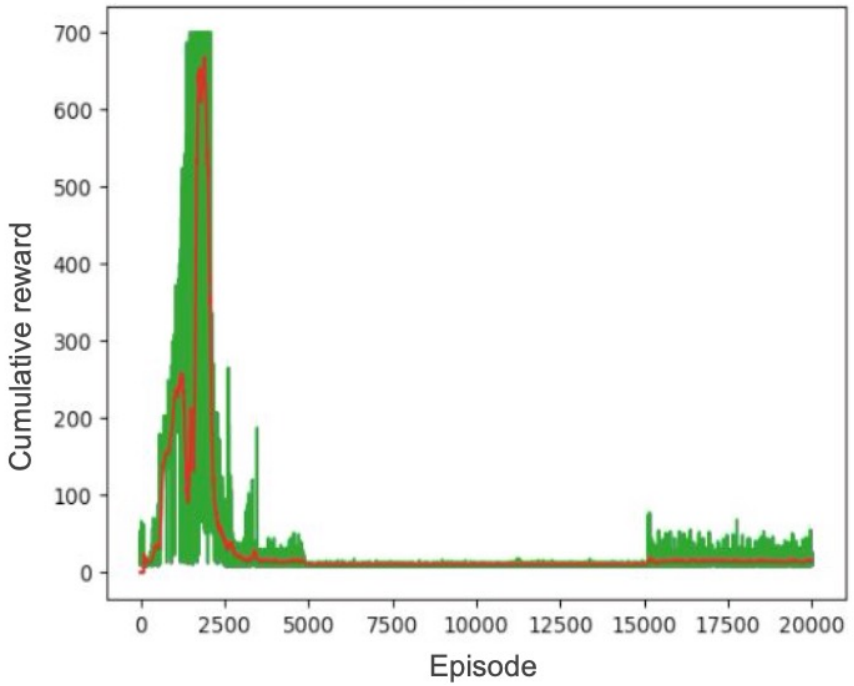
- Concept: Create a distinct policy/target pair for every unique Q function needed
- Supports dynamic addition of new variables/var combinations



Discrete Values		Q
d_p	d_c	
left	left	$f_{ll}(\theta, \dot{\theta}, x, \dot{x})$
left	right	$f_{lr}(\theta, \dot{\theta}, x, \dot{x})$
right	left	$f_{rl}(\theta, \dot{\theta}, x, \dot{x})$
right	right	$f_{rr}(\theta, \dot{\theta}, x, \dot{x})$

But Poor Learning Outcomes…

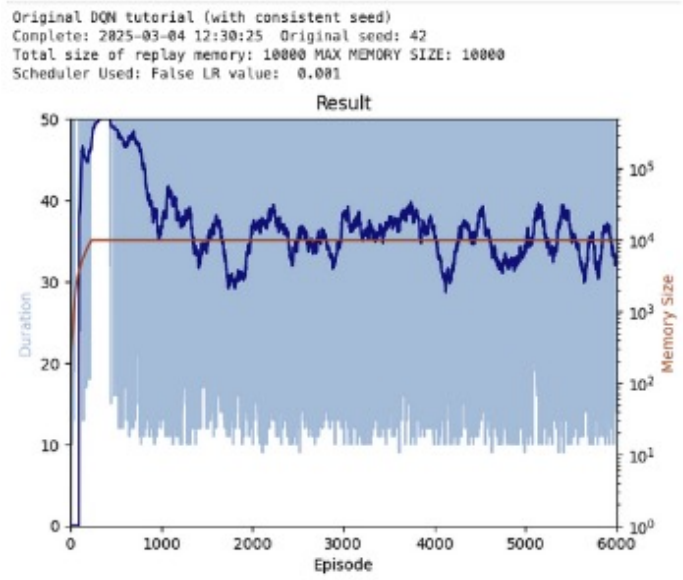
(Two discrete, four continuous vars, abs of discrete vars)



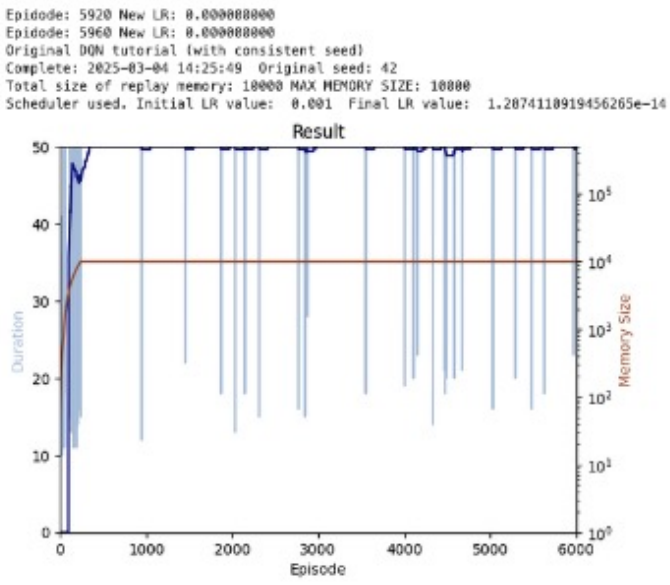
Extensive exploration of issues here

- Explore/Exploit policy
- Learning rates
- Memory size/queueing behavior
- Environment assumptions (treatment of reward on termination)
- …

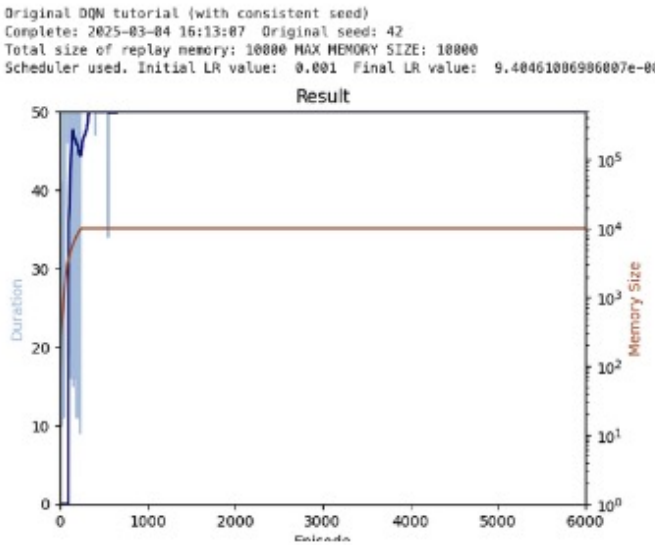
One Glimmer… Learning Rate Scheduling



Basic cartpole, No schedule



Basic cartpole, Fixed schedule



Basic cartpole, Decrease on plateau

Conclusions

Nuggets

- Improved understanding of state of art RL algorithms and how they map to Soar's requirements
- Possibly feasible to integrate DQN (or DDQN) with Soar (meets basic requirements?)

Coal

- Within time/resources available, did not find a compelling continuous RL candidate for integration in Soar
- Inconsistent algorithm behavior over domains and number of vars. No one algorithm to rule them all?
- Soar incremental, online learning requirement contrasts with state-of-art batch-focused DRL methods. Is there a compromise?