Reinforcement Learning Approaches for Atari Breakout

Vincent Pierre Berges, Reid Pryzant, Priyanka Rao
CS 221 Final Project

Motivation

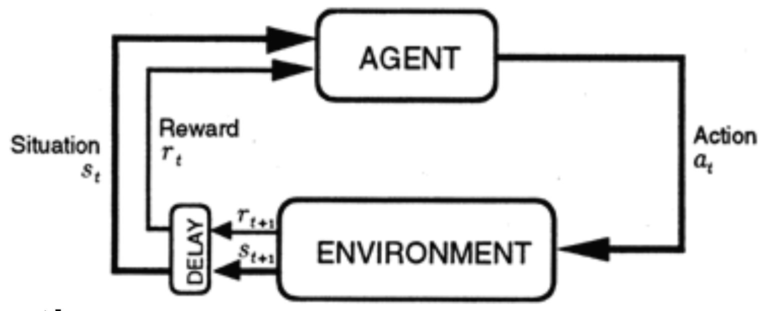
How do different RL approaches compare in a custom implementation of Atari Breakout?

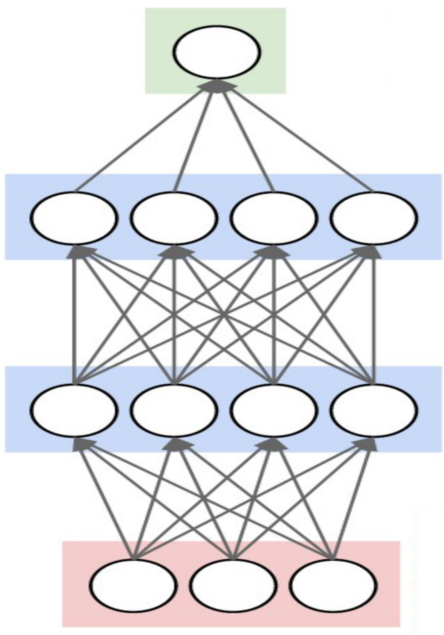
Model

Discrete Feature Set	• Game state • Ball and paddle location • Ball angle • Brick indicators
Multiple Continuous Feature Sets	• Ball and paddle locations and velocities • Distance of ball relative to walls, bricks, and paddles • Interaction features
Pixel Intensities	• 3D vector representation of pixel RGB values • Used with a 5-layer neural network

Challenges

- Featurizing a huge state space
- Delayed rewards
- Exploration vs exploitation
- Determining relevance of hyperparameters
- Learning from losing vs from hitting bricks
- Highly correlated states
- Experience replay vs SARSA(λ)
- Training neural network

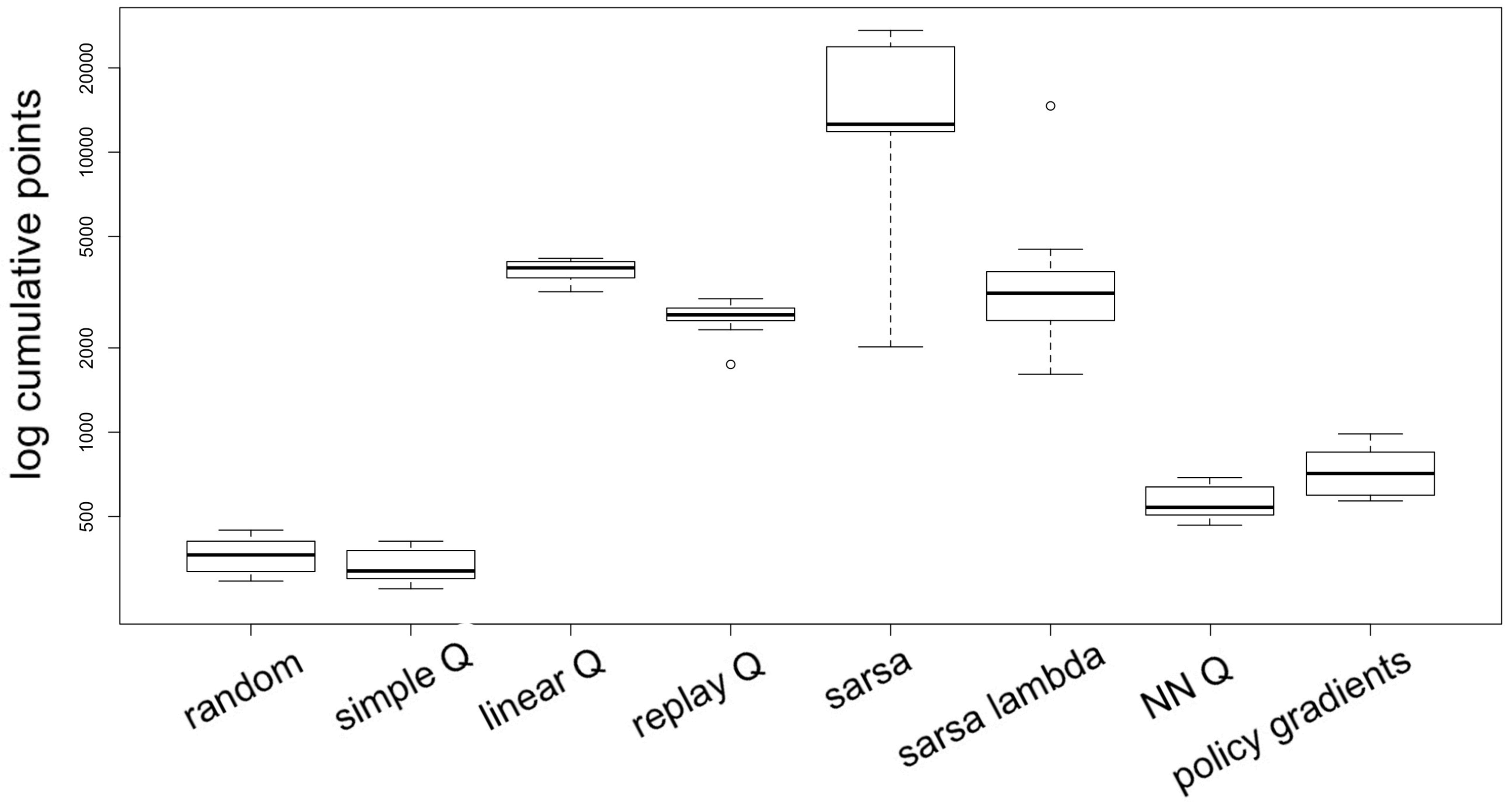




Algorithms

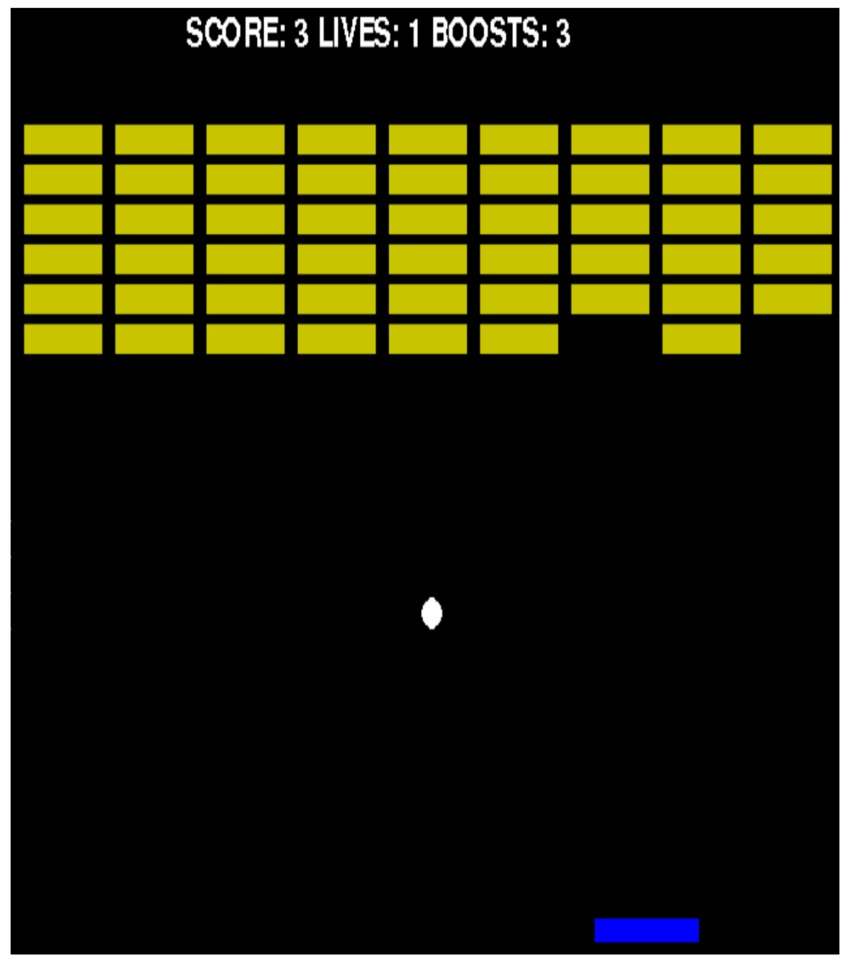
Baseline	• Follows a random policy
SARSA(λ)	• Maps (s, a) to Q values of *current* policy • Combine past rewards, more recent = more important • Maintain an eligibility trace to assign blame to parameters $min_w \sum_{s,a,r,s',a'} (\hat{Q}_\pi(s,a;w) - (r + \hat{Q}_\pi(s',a';w)))^2$
Q-learning with Replay Memory	• Maps (s,a) to Q-values of *optimal* policy • Estimate Q(s,a) with linear and neural network function approximators • Bootstrap estimate of future value by sampling from experience • Cache parameters of target function for stable updates $min_w \sum_{s,a,r,s'} (\hat{Q}_{opt}(s,a;w) - (r + max_{a'} \hat{Q}_{opt}(s',a';w)))^2$
Policy Gradients	• Directly learn parameters θ of a policy π_θ (vs ε-greedy) • Use a neural network for the policy, but wait to fill in gradients until eventual reward is received $max_\theta \; E\left[\sum_{k=0}^{H} \gamma^k r_k\right]$

Results



Cumulative points over multiple runs of 250 test games (ε = 0) after 2000 training games (ε = 0.5). 3 points are awarded per broken brick and 1000 points for winning. Experiments were conducted with γ = 0.993, η = (1 / x), |memory| = 5000, update cycle = 750, eligibility trace threshold = 0.1, trace decay = 0.98. The SARSA agent won 46 games.

Analysis



- Almost all algorithms outperformed baseline by 2x
- Q learning w/out function approximation struggled because large state space was inadequately explored
- Replay memory added no benefit – for Breakout, correlations between adjacent game states sometimes *help* agent performance. Delays between actions (ex. returning a ball) correlate to delays in rewards (ex. breaking a brick).
- Agents leveraging nonlinear policy & value networks generally underperformed.
- Neural network did not help despite hyperparameter tuning and different network structures – too little info captured in feature sets
- Future work: investigate each model (especially the more opaque ones) to understand their performance in Breakout vs other games

Contact Information

rpryzant@stanford.edu, vpberges@stanford.edu, prao96@stanford.edu

References

[1] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." *arXiv preprint arXiv: 1602.01783* (2016).
[2] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529-533.
[3] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).
[4] Silver, David. "COMPGI13: Reinforcement Learning". University College London online course. London, UK. 2015. Lecture Series.