Reinforcement Learning for Atari 2600
(JStella Learning Environment)

Ian Cichy <cichy.ian@uwlacrosse.edu>  
Martin Allen <mallen@cs.uwlax.edu>

Abstract

Stella, a Java based open-source Atari 2600 emulator, allows users to play some of the most iconic video games of all time. Building upon this system, substantial modifi-
cations were made to develop the JStella Learning Environment (JSE). The
JSE provides the ability to develop and run many different types of machine learn-
ing algorithms on a wide variety of Atari 2600 games. To make this possible, the
original Stella system was overhauled to allow other code to interface with it. Game
information had to be made available for agents to access, and the game had to be
controllable by an agent sending it actions. Focus then shifted to testing multiple
algorithms that would robust enough to work on different games and improve
performance compared to human players. Q-learning has been used as a simple but
elegant solution for the game Breakout.

State Space Analysis

Breakout has 6 rows of 16 bricks each. The bottom two rows are worth 1 point
each, the middle two 4 points, and the top two 7 points, for a maximum of 432 points.
The following tests were each given 1,600 30-second episodes to play Breakout.
Each episode consisted of 38,000 actions sent after every frame of the game. The
test started by taking random actions and slowly reducing randomness until episode
500, where the agent was no longer random. Both tests used the same set of three
actions: left, right, and no move (which causes the paddle to stay in one place).

State Space Analysis Continued

Test C2 (previous column, bottom) had over 6 billion possible states due to the
number of combinations of x and y positions. Even with this large number of states
the agent was still able to learn how to play the first few frames of the game and get
a few points. The reward function for test C2 is the same as test D9.

Reinforcement Learning in Dec-POMDPs

Agents observe, act, and receive reward in an uncertain world.
Our model is the Decentralized partially observable Markov decision process:

\[ M = \{ \{a_i\}, S, \{A_i\}, P, \{O_i\}, O, R, T \} \]

- Each \(a_i\) is an agent.
- \(S\) is a finite set of world states.
- \(A_i\) is a finite set of actions available to \(a_i\).
- \(P\) is a Markovian state transition probability function.
- \(O_i\) is a finite set of observations for \(a_i\).
- \(O\) is the joint observation function for state transitions.
- \(R\) is a global (thus cooperative) reward function.
- \(T\) is the time-horizon of the problem (finite or not).

\[ Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \left[ r_{t+2} + \gamma \max_a Q(s_{t+2}, a) - Q(s_i, a_i) \right] \]  

One step Q-Learning

Test A1, only having 6 x and 28 state-action pairs, quickly learned to stay under
the ball at all times. This allowed it to nearly clear the board on many occasions
and perform much better than human players.

Test B2 had a much larger state space with 117,000 states and 350,000 state-action
pairs. The goal was to make this agent smarter and have it predict where the ball
was going. It was also penalized for unnecessary movement and rewarded for hitting
the ball rather than staying under it.

Test D9 had 299,926 states for 600,000 state-action pairs. The goal was to give
the agent more detailed information compared to test B2 in hopes that it would lead to
better prediction of movement and higher scores.

References