Project #7: Reinforcement Learning

Paul Fornia | JHU 605.649
December 9, 2018

Abstract

I explore reinforcement learning, specifically the Value Iteration and Q-Learning algorithms. Unlike supervised Machine Learning algorithms explored so far, which attempt to model underlying functions present in large labeled datasets, reinforcement learning seeks to find a policy of actions to reach a goal state based on a set of rules known as a transition model. Value Iteration creates a policy based on access to this transition function. The Q-Learning algorithm relaxes this requirement and learns more "purely" from trial-and-error. I test the hypotheses that a) both algorithms will produce policies that reliably finish the race, b) the Value Iteration algorithm will produce faster policies than the Q-Learning algorithm, and c) that the "bad crash" scenario will lead to policies that will act more risk averse, stay further away from walls, and finish the race in slightly more iterations, even when crash restarts are not enforced.

Intro and Problem Statement

Reinforcement learning is the process of learning a policy to navigate states (including starting states and goal states) by performing actions. A transition function dictates which state the agent will go to based on the initial state and the action. Reinforcement learning algorithms must either have access to this transition function, or learn it implicitly through trial-and-error.

I implement and test the *value iteration* and *q learning* algorithms on a racetrack problem. The states of the world consist of the location and velocity of a vehicle, and the possible actions are directional acceleration. Direction that the car is "pointing", and other states of the vehicle are ignored. The transition function is a simple physics model (new velocity equals old velocity plus acceleration in the x and y directions) with two caveats. First, there is an element of non-determinism, where with probability 0.2, the effective acceleration is zero, regardless of action taken (perhaps to simulate a skid or spinning tires). Second, the world map consists of walls and boundaries, which modify the transition model. Under the "normal crash" scenario, traveling into a wall (or boundary) will set the velocity to zero, and set the location to the nearest open location. Under the "bad crash" scenario, the agent will travel back to the starting location, and velocity will be set to zero, effectively restarting the race (but with keeping the accumulated costs of states already traveled to).

The goal of this project is to study the behaviors of these two reinforcement learning approaches. I test the following hypotheses:

Hypothesis 1

Both the Value Iteration and Q Learning Algorithms with produce policies that will reliably finish both race tracks. In particular, 50 random simulations will be run on each of tracks 'R' and 'L', with both crash types (R only), and both algorithms (permutations detailed in table of results section; 300 races total) with no race failures. Here, I define a race as a failure if either a) 5 consecutive turns are taken with zero velocity (e.g., stuck on a wall), or b) 500 turns are taken without crossing the finish line.

Hypothesis 2

For all four experiments (taking the average finish time over 50 trails for each experiment), testing both tracks with the two types of crashes, the value iteration policy will complete the race faster than the q-learning policy. This hypothesis stems from the simple observation that the value iteration algorithm has access to more information, namely the precise state-to-state probabilistic transition model.

Hypothesis 3

For both algorithms, the crash scenario that simply stops the car at the wall will perform better (faster average of 50 simulations) than the "bad crash" scenario in which the car must restart at the beginning state. This is trivial, since the restarting crashes take up more turns; however, I further hypothesize that this will hold even when the simulation itself does not enforce the bad crash scenario. This is because the "bad crash" policy should be more conservative and sacrifice some speed in order to take a less risky policy.

The Algorithms

Both reinforcement learning algorithms iteratively search over all states and actions, while keeping track of three lists:

Values:
A "Value" is maintained for each *state*. The value of a state is based on the expected discounted reward that will result if a state is visited.

Q:
The Q list is similar to values, but is maintained at a more granular level. A Q value is tracked for each state *and action* pair. The Q value of a state-action can be approximately interpreted as the value expected by taking the action from the specified state.

Policy:
The policy list is the prescribed action that is to be taken for any given state (e.g., if at state A near the finish line, perform acceleration B towards the finish line). One example "greedy" policy could be calculated as the argmax of the Q list for each state. Another policy (called epsilon-greedy) could add some random non-optimal choices to promote exploration of states. The implementation used in this implementation is a simple argmax greedy policy.

Value Iteration

The Value Iteration algorithm is used when the learner has access to the transition function. The algorithm works as follows:

- Until Convergence (biggest value change falls below threshold):
    - For each State:
        - For each action:
            - Update Q value to reflect expected value from transition function.
        - Update value based on argmax of Q for that state (i.e., according to the action the policy would prescribe).
- Output policy based on argmax of Q.

Q-Learning

The Q-learning algorithm is similar, except that it assumes no access to the underlying transition model, and therefore must explore actions by trial and error. It works as follows:

- For fixed number of “episodes”:
    - Choose random initial state.
        - Repeat until goal state or max episode length:
            - Pick an action based on Q (e.g., argmax or ε-greedy)
            - Update Q for old-state
            - Move to new state by chosen action
- Output policy based on argmax of Q.

Experimental Design

The outputs from the algorithms are the policy lists as described above. Once a policy is determined, it can be tested via simulation. Unlike the loops within the algorithms, many states in the simulation may never actually be visited during simulation. Rather, the agent simply starts at the initial state, and follows the prescribed policy (and transition model or simulation) to travel from one state to the next, keeping track of the total reward or other similar metric (in this case, simply time to finish a race). In a deterministic world, this would be sufficient. Due to the non-determinism of this problem, the simulation is repeated 50 times and average time to compete is recorded.

For hypotheses 1 and 2, the simulation must be matched to the training algorithm. That is, if normal crashes are used in the training, then normal crashes are used in the simulation. For hypothesis 3, the algorithm is used from the “bad crash” scenario, but the simulation used the normal crash scenario.

Results

The following table shows the number of turns needed to compete each simulation. The results are averaged over 50 trials, and all these simulations complete the race (no time-outs, or getting "stuck" on a wall). For value iteration, training continues until the max change in *values* is smaller than 0.001 (at which point convergence is reached very quickly, so similar results can be produced from much smaller threshold values). For q-learning, training is run for 500 thousand episodes. The learning curves below show sensitivity to this parameter.

Track	Value Iteration	Value Iteration (Bad Crashes)	Value Iteration (Bad Crashes, not enforced)	Q-Learning	Q-Learning (Bad Crashes)	Q-Learning (Bad Crashes, not enforced)
R-Track	**23.54**	**35.06**	**28.16**	**26.36**	**37.54**	**35.52**
L-Track	**11.2**			**11.52**		

- Hypothesis #1 is confirmed. All 300 trial experiments completed without timing-out or getting stuck against a wall.
- Hypothesis #2 is confirmed. For all experiments, the value iteration policies outperformed their analogous Q-learning policies.
- Hypothesis #3 is confirmed. The policies that were trained under the "bad crash" scenario are more conservative and produce slower races than the "normal crash" policies, even if the restart crash was not enforced during the simulation.

Algorithm Behavior and Conclusions

Handling Crashes

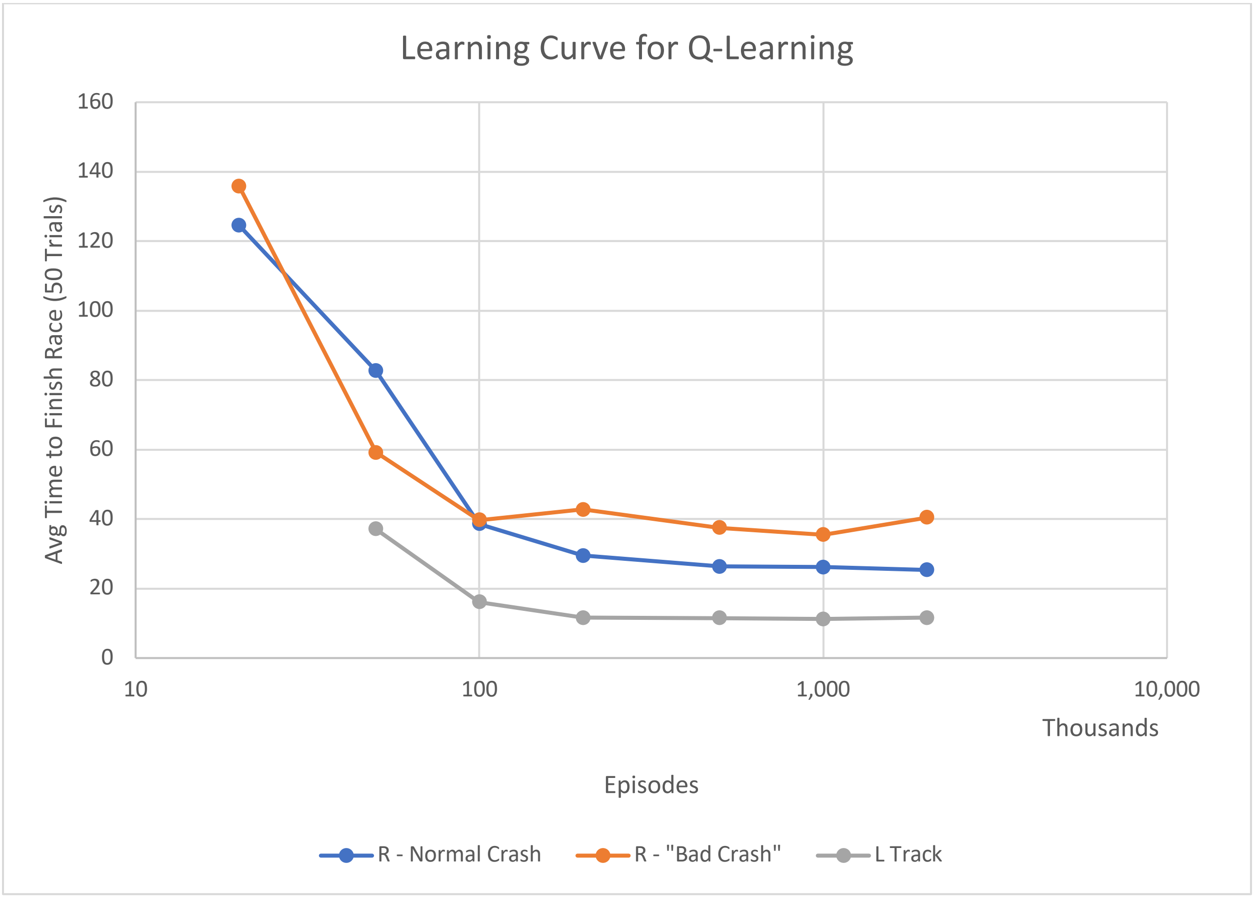
One of the trickier implementation aspects of the "normal crash" scenario was determining where to reset the agent in the event of a crash. At first, the algorithm simply went back to the previous state's position. This, however, was unrealistic, since the agent can move up to 5 tiles in a single action. Next, a method was written to find the closest open tile, and move the agent to that tile. That fixed the previous issue by setting the agent to a tile immediately next to a wall, rather than back several spaces.

Unfortunately, this implementation had one major flaw, which was the ability to "jump" over portions of the track. This was a problem with the R track: In early versions, the RL learned to jump into the wall at the small gap near the middle of the track. The "nearest" tile would then be identified to the right of the wall, skipping the upper portion of the track entirely. In the final implementation, this was fixed by finding the nearest open tile in the general direction opposite to the velocity. In other words, if the car is moving down and to the right, a crash resets to the nearest tile that is up and to the left of the crash.

Convergence

For the value iteration algorithm, I use a termination criterion based on the maximum change in state value. By observing the max value over iterations, it appears that the max change decays slowly while the rewards propagate over the states. Once this propagates completely across states, there appears to be a steep and exponential drop-off of theses changes. In other words, an extremely small max change threshold (e.g., 1/1 million) only takes slightly more iterations than a moderately small max change (e.g., 1/10). So convergence is not sensitive to this threshold in value iteration.

For the Q-Learning algorithm, there is no such obvious termination criteria. It is therefore more important to observe the learning curves to ensure convergence. The below graphs show how simulations perform from policies trained on a varying number of episodes (log scale). It appears that performance tapers around 500,000 training episodes for the R track, and around 200,000 for the L track.



Summary

I explore two reinforcement learning algorithms, value iteration, and q-learning. Both of these algorithms are successful at creating policies that can navigate a simple racing simulation. I find that the value iteration policies tend to outperform the q-learning algorithms, likely due to their access to the underlying transition model. I also find that by simulating a “bad crash” scenario, in which crashes cause the agent to restart the race, both algorithms can adapt by creating a more conservative policy, that is a bit slower, but that more carefully avoids walls.

References

Alpaydin, Ethem. Introduction to Machine Learning (Adaptive Computation and Machine Learning series). (Ch. 18) The MIT Press. Kindle Edition.

Sutton, R. S., Barto, A. G.. *Reinforcement Learning: An Introduction*. (2005) (Ch. 4, 6) The MIT Press.

Watkins, C.J.C.H. (1989). Learning from delayed rewards. PhD Thesis, University of Cambridge, England.