PROJECT 6
REINFORCEMENT LEARNING

DAVID ATLAS

Abstract. This paper will introduce basic concepts in reinforcement learning, including three algorithms for finding policies: Value Iteration, Q-Learning and SARSA. The paper will introduce a simulated racetrack environment, and present results on learning a policy in the environment. It is found that the agent is able to learn a policy ¡fill in here¿

1. Problem Statement & Hypothesis

This paper introduces several core concepts in reinforcement learning. Reinforcement learning attempts to solve the problem of determining an optimal policy for an agent at any given state in an environment, given some reward structure.

The basic problem in reinforcement learning is a Markov Decision Process (MDP). In an MDP, the optimal decision depends only on the current state of the agent. At each time step, after an action is taken, a reward is given.

Specifically, in this paper, the agent will be a racecar on a racetrack with several different elements:

- A set of starting locations for the car.
- A set of finish line locations for the car.
- A set of out-of-bounds markers for the car.

The problem of solving the policy involves determining the optimal acceleration at each location of the racetrack.

Additionally, there is an element of randomness involved - at each step, after an acceleration action is chosen, there is only an 80% chance of it happening as expected.

There are 3 tracks included in the problem, each with increasing size and difficulty:

(1) L-Track
(2) O-Track
(3) R-Track

The hypothesis is that all of the algorithms should be able to solve all of the tracks, as they all meet the Markov criteria (all information needed is encoded in the current state). However, the algorithms take quite a while to train, and so there may be difficulties finding an optimal policy for all of the tracks.

Note that each track has two variants - in the first variant, which will be called the harsh variant, if the car hits the out-of-bounds marker, it is returned to the starting location. In the second variant, which will be called the simpler variant, the car is simply returned to its last location with velocity of 0. It would be expected that the harsh variant is more difficult to train, as restarting from the beginning of the track would give potentially large policy values to states that are close to the finish line. As the randomness of the track could cause a crash on what would have otherwise been a good decision, it will take more iterations to converge to the correct values.

2. Description of Algorithms

2.1. **Notation.** The following concepts will be useful in describing the algorithms below:

- S is the set of states in the environment.
- A is the set of actions that can be executed by the agent.
- Policy $\pi(s)$ is a mapping from state s to action a. This can be thought of as the guidebook for the agent, telling it what to do at each state in the environment.
- $Q(s, a)$ is a mapping that dictates the expected reward for choosing action a from state s.

- γ is the discount rate, or the amount by which a future reward is discounted. A reward in t iterations gets discounted by a factor of γ^t.

2.2. **Bellman's Equation.** The solution to an MDP can be written as

$$V^*(s_t) = \arg\max_{a_t} Q^*(s_t, a_t) \tag{1}$$

$$= \arg\max_{a_t} E[r_{t+1} \sum_{t=1}^{\infty} \gamma^{i-1} r_t + i + 1] \tag{2}$$

$$= E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) \arg\max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}). \tag{3}$$

Equation 3 is known as Bellman's Equation. The terms can be thought of intuitively as the "correct" value for a given state-action pair $Q(s_t, a_t)$ at time t as the reward of executing the action (r_{t+1}), as well as the sum of all expected future rewards from states s_{t+1}, weighted by the likelihood of ending up in the state after executing action a_t from s_t. This accounts for any randomness in the environment that leads to probablistic transition dynamics after executing a_t from s_t.

2.3. **Value Iteration.** The Value Iteration algorithm[1] attempts to find $Q^*(s, a)$ by iteratively updating each entry in Q as the sum of the current value and the discounted value from executing each action. The algorithm appears as follows:

Algorithm 1: Value Iteration

```
Result: Q*(s, a)
Initialize Q(s, a) to arbitrary values;
for i ← 0 to Number of Iterations do
    for s ∈ S do
        for a ∈ A do
            Q(s, a) ← E[r | s, a] + γ Σ_{s'∈S} P(s' | s, a)V(s') ;
        end
    end
end
```

The intuition here is that at each iteration, each state-action pair updates the estimate of its value by looking at the states it would end up, and their respective values. It then discounts backwards and add the reward of the future state to its value. After many iterations through all the state-action pairs, the algorithm converges to values close to the optimal $Q^*(s, a)$.

2.4. **Q-Learning.** The next algorithm implemented was the Q-Learning algorithm[3], which (as might be expected) is another algorithm for learning $Q^*(s, a)$. Before introducing the algorithm, the ϵ-greedy action selection policy must be introduced. For a state s_t, the ϵ-greedy policy selects the optimal (as of time t) action with probability $1 - \epsilon$, while selecting a random action with probablity ϵ. This can be thought of as a tuning parameter, where the agent will spend more time experimenting when ϵ is higher. This is useful in iterative training because it would be undesirable for the agent to choose the optimal policy at the outset, as it will be randomly initialized.

With that, the Q-Learning algorithm is shown in Algorithm 2.

Algorithm 2: Q-Learning

Result: $Q^*(s, a)$
Initialize $Q(s, a)$ to arbitrary values;
for $i \leftarrow 0$ **to** *Number of Episodes* **do**
  $s \leftarrow s_0$ (a starting state);
  while s *is not a terminal state* **do**
    Choose a via ϵ-greedy selection;
    Get r and s' from applying a in s;
    $Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma \arg\max_{a'} Q(s', a') - Q(s, a))$;
    $s \leftarrow s'$
  end
end

The intuition behind algorithm **??** is to use the ϵ-greedy action selection to explore the space, while estimating $Q(s, a)$ as a mix between its current value and the approximate additional reward that would be come from taking action a, and the proceeding in accordance with the policy at the new state, s'. The big difference between Q-Learning and Value Iteration is that Q-Learning uses episodes, in which the agent makes a sequence of decisions from the beginning to the end, updating the policy along the way, whereas Value Iteration updates each state-action pair on each iteration. This means that for very large state spaces, Q-Learning may have an easier time learning a good policy, while Value Iteration would require alot of computational resources to update each state at each iteration.

2.5. **SARSA.** The last algorithm presented here is SARSA (State-Action-Reward-State-Action)[2]. This is known as an on-policy learning algorithm, as the policy is used to determine not only the immediate action a, but also the succeeding one, a'. This is in contrast to Q-Learning, in which the policy only determines a, and the ensuing reward is estimated by the approximation of the optimal choice at state $Q(s', a)$. However, the agent does not necessarily make that optimal choice. Beyond this difference, the algorithms are quite similar. SARSA is shown in Algorithm 3.

Algorithm 3: Q-Learning

Result: $Q^*(s, a)$
Initialize $Q(s, a)$ to arbitrary values;
for $i \leftarrow 0$ **to** *Number of Episodes* **do**
  $s \leftarrow s_0$ (a starting state);
  Choose a via ϵ-greedy selection;
  while s *is not a terminal state* **do**
    Get r and s' from applying a in s;
    Choose a' from ϵ-greedy selection in s';
    $Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma \arg\max_{a'} Q(s', a') - Q(s, a))$;
    $s \leftarrow s'$;
    $a \leftarrow a'$;
  end
end

2.6. **Experimental Approach.** There were 3 environments used in each experiment:

(1) L-Shaped Track
(2) O-Shaped Track
(3) R-Shaped Track

For each environment, Value Iteration, Q-Learning and SARSA were all applied to the environment. Each algorithm was allowed to train for a variable number of iterations, until performance converged to good solution. To be clear, a good solution here implies going from start to finish in a number of steps proportional to the difficulty of tracing such a path.

The algorithms have hyperparameters - all of the algorithms include a discount factor, which is the amount by which a future reward is discounted back to the present time step. Both Q-Learning and SARSA include

Track	Algorithm	Crash Variant	Discount Factor	Learning Rate	Number of Training Iterations	Average Steps to Solve
L	Value Iteration	Harsh	.7	N/A	250	NA
L	Value Iteration	Non-Harsh	.8	N/A	100	NA
L	Q-Learning	Harsh	.5	.2	100000	NA
L	Q-Learning	Non-Harsh	.5	.2	40000	NA

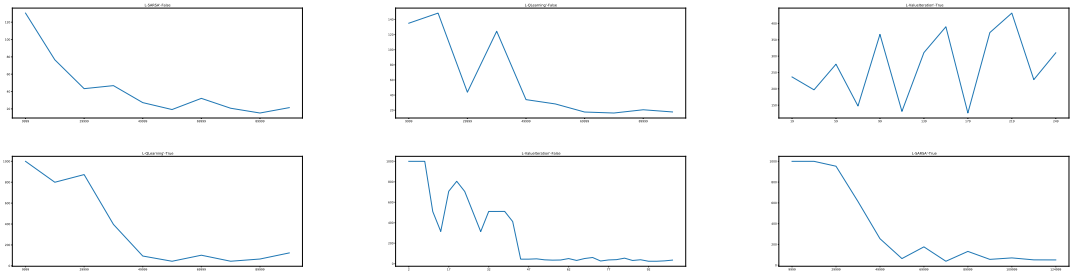

FIGURE 1. This figure shows the average number of steps needed to solve the track after a given number of training iterations for each of the 6 L-track algorithm and crash variant combinations. Note that the exploitation step is limited to 1000 steps at maximum. 10 exploitations were run for each starting point on the track.

a learning rate, which is the proportion of the temporal difference that is added back to the current tally of the expected reward in a given state. Additionally, Q-Learning and SARSA have an ϵ parameter that dictates the likelihood of choosing a random action in each state.

Due to the time-consuming nature of training the algorithms, it would be preferable to avoid exhaustive experimentation with respect to the hyperparameters. Therefore, for track-algorithm pairs that did not require any tuning to converge to a good solution, values were chosen based on some intuition about the relationships between the hyperparameters and the training process. This included all variants of all algorithms for the L-track and the O-track, and SARSA and Q-Learning for the R-track under the non-harsh crash variant. For the ϵ-greedy policy, the larger of $\frac{1}{T}$ and .25 was used to ensure sufficient exploration of the space.

For the remaining combinations, some light experimental tuning was used to find more preferable values. However, this approach was not useful, and the algorithms were not converging in a reasonable time. Therefore, a "pre-training" approach was used, in which the starting policy was the policy found by SARSA under the non-harsh variant.

2.7. **Algorithm Results.** The results of the algorithms can be found below in Table **??**

REFERENCES

[1] Richard Bellman. A markovian decision process. *Indiana University Math Journal*, pages 679–684, 1957.
[2] M. Niranjan G.A. Rummery. On-line q-learning using connectionist systems.
[3] Francisco S. Melo. Convergence of q-learning: A simple proof.