Reinforcement Learning for Mountain Car Game

Kyle Weidner Tyler Sanbar
University of Oklahoma
CS 4033

Abstract
In this report, we highlight the hypotheses, experiments, and results of experiments designed to teach an agent how to successfully climb a mountain.

Project Domain

The MountainCar-v0 environment consists of a car initially at the bottom of a valley between two hills, and with a goal state on the top of the right hill. The car is not able to reach the goal by forward movement alone, and must learn how to create forward momentum by driving backwards and forwards between the two hills. The observation space consists of a horizontal (x) position between -1.2 and 0.6, and horizontal velocity between -0.07 and 0.07. The action space consists of accelerating with a force of 0.001N to the left or right, or not accelerating. The environment gives a -1 reward at each time step. The car is given an initial random position from -0.4 to -0.6, and velocity of 0. The episode terminates after 200 time steps, or when the car reaches a terminal position at x=.5, which is given a reward of 0. This environment is a continuous state space, so to discretize it, we rounded the x values to one decimal place, and velocity values to two decimal places. This results in 18 possible position states, and 140 velocity states for a total of 18*140=2520 states.

Literature Review

Our first paper (Custode and Iacca 2020) included the Mountain Car in tests they performed on their techniques which incorporate decision trees with Q-Learning. The authors show their performance in relation to other state of the art performances on this test, as well as an interesting look at the decomposition of the policy results. Their best achieved score was -101.72 by Orthogonal DT. Other state of the art scores range from -102.61 to -128.87. They source a Tabular SARSA method by Amit which we use as well as achieving a -105.99 score. The following figure helped us conceptualize how the policy would be organized. However, since we would not be incorporating any of these decision tree methods, it would not be possible to compare our policy results, so we will focus on performance discussions.

Our second paper, Robust Deep Reinforcement Learning with Adversarial Attacks, explores the robustness of reinforcement learning algorithms. The paper talks in detail about the ways in which reinforcement learning agents can fail by being attacked, and defines an attack on an agent as "increased probability of taking '"worst" possible action in that state" (Pattanaik, Tang, et. al. 2017). During the implementation of our Monte Carlo agent, it became apparent that the third action, in which the agent did not accelerate in any direction, consistently had the least Q value of all actions, making it the "worst" action. When our E-Greedy policy would explore using a random action, our agent was inadvertently being attacked if the third action was picked. To resolve this, we limited our policy to only exploring with left or right acceleration and saw significant improvements.

Experiment Review

Hypotheses

We hypothesized that we could use Q-Learning, Sarsa, and Monte Carlo ES (exploring starts) reinforcement learning techniques to train an agent to find a policy that will reach the goal state before 200 time steps have passed by using a discretized representation of the state space. We chose these because they utilize state action values, and are model-free which is required for this environment since we do not have a model. We also hypothesize that out of these three algorithms, Q-Learning would learn the fastest, but would be outperformed by Monte Carlo ES and Sarsa, with Monte Carlo ES performing the best.

Experiments

For all of our experiments, we used the following parameters. Number of iterations - 100,000. E-Greedy policy with Epsilon starting at 1, with .999 exponential decay down to .1. Gamma values (Sarsa and Q) - .999 Alpha values, we ran experiments on all of the following for each algorithm - [.001, .01, .1, .5, .9] We represented our states in the form (position, velocity), and stored action values within each of these state representations. All environment variables were the same between each of the experiments.

Results

For each of the experiments, we stored cumulative reward per episode data along with the associated alpha values. In all Mountain Car experiments, you will expect a reward of -200 for each episode until the policy is able to reach the goal state, where you will then get a reward corresponding to the time steps it takes to reach the goal. As you can see in each of the figures, this does occur indicating that each of our algorithms were successful in learning for some alpha values, and mostly increased their scores before leveling out in a somewhat stable policy.

Figure 1 shows each of the algorithm's highest alpha score runs.

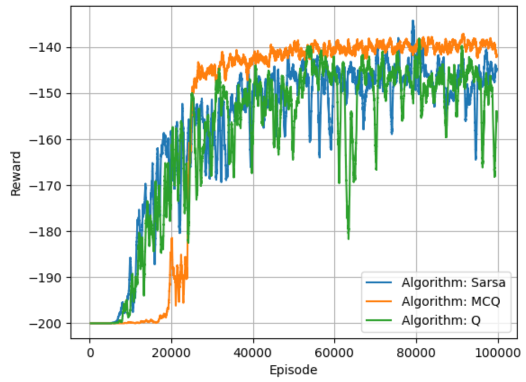

Figure 1: Cumulative reward averaged (over 500 episodes) of Sarsa, Monte Carlo ES, and Q-Learning

MCES was able to achieve the most stable policy, as well as highest win rate out of the three algorithms, with Sarsa and Q performing similarly, although Q slightly below and with more negative variance. MCES did take longer to learn at first, as shown in figure 1 which lists first successful runs, but then increases quite quickly to its stable rate above both the others.

	Alpha	**Average Cumulative Reward**
MCES	0.1	-159.69
Q-Learning	0.01	-199.97
Sarsa	0.01	-199.99

Figure 2: Cumulative rewards from each model using their optimal alpha

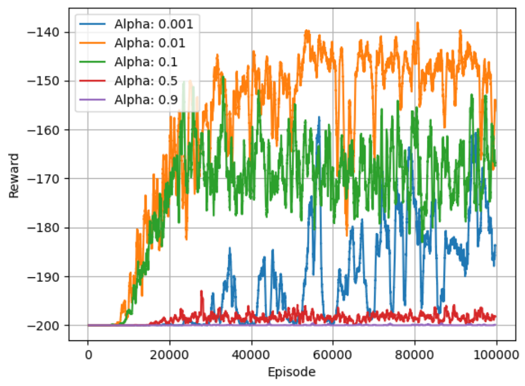

Figure 2: Q-Learning's cumulative rewards with varying alphas

Although MCES was the slowest learning of all our implementations, MCES achieved the best convergence of all our models. The MCES algorithm makes two assumptions, that each episode will have exploring starts and policy evaluation will be done with an infinite number of episodes. Our environment from OpenAI gym provides MCES with a random starting state to satisfy the first assumption, as for the second assumption we simply ran a significant number of episodes and progressed the value function towards optimality on each evaluation step.

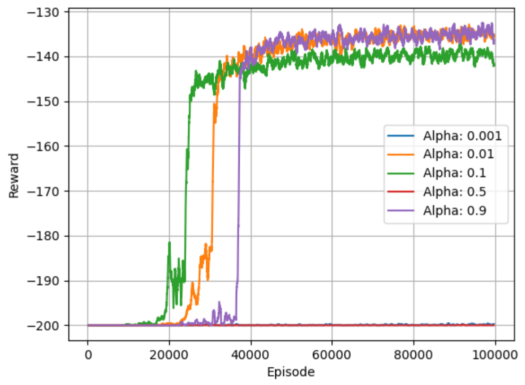

Figure 3: Monte Carlo with Exploring Starts performance with varying alphas

Our implementation of Sarsa provided us with an on-policy model to compare to our other two off-policy experiments.

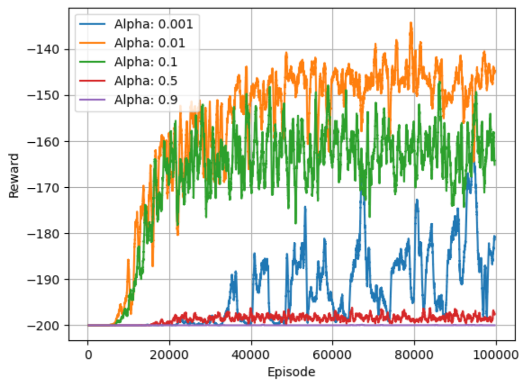

Figure 4: Sarsa's cumulative rewards with varying alphas

Comparison with Prior Work

Mountain Car v0 is a classic machine learning game, but it has many different variations on environment rules which people use, so it can be difficult to find perfect comparisons. All of the published research we found including the AI Gym environment we used have innovated novel Reinforcement Learning approaches, so strict comparison to our Sarsa, Q and MCES algorithms was not possible. However, we consistently found that our algorithms were outperformed. Our first paper (Custode and Iacca 2020) demonstrates a mean score of -105.99, while ours was -159.69.

Conclusion

In conclusion, we found Sarsa, Q-Learning, and Monte Carlo to be suitable algorithms for solving the Mountain Car problem, but were unable to find optimal policies. The continuous nature of the problem makes an optimal policy impossible to find with discretized states, so exploring further solutions to continuous machine learning algorithms should provide significant improvements.

Contributions

Tyler implemented the SARSA learning method. Kyle implemented Q-Learning and MCES.

References

L. L. Custode and G. Iacca, Evolutionary learning of interpretable decision trees. arXiv, 2020. doi: 10.48550/ARXIV.2012.07723.

A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary, Robust Deep Reinforcement Learning with Adversarial Attacks. arXiv, 2017. doi: 10.48550/ARXIV.1712.03632.