Vince Olsen - CS510 Reinforcement Learning

Applying Q-Learning with Autodidactic Iteration to the Rubik's Cube Puzzle

Summary

Solving a Rubik's Cube using reinforcement learning is difficult because there is a large state space and traditionally only one of those states has a reward, the goal state, which has the reward for solving the puzzle. Autodidactic Iteration (ADI) takes that one reward state and smears it out over the states that lead up to that reward state. This is done by starting at the goal state and incrementally visiting the states that can lead to that reward state and giving each state a reward, and reducing the size of that reward the farther the state is from the goal state. This creates a funnel of rewards that an agent can greedily follow to the goal state.

Prior work applying reinforcement learning to Rubik's Cubes was reviewed. The work by McAleer et al (https://openreview.net/pdf?id=Hyfn2jCcKm) was implemented and improved upon. An algorithm using Q-learning with ADI was optimized and then used to solve Rubik's Cubes. Values for the parameters of the algorithm were iterated over and the results graphed. The optimal parameter values were then tested on increasing complexity levels of the puzzle where the puzzle was randomly scrambled an increasing number of times prior to solving. Agent performance was compared using ADI and not using ADI.

The agent performed about 6 times better using ADI than not using ADI from scramble depths of 4 to 12.

Methodology

The Rubik's Cube puzzle was implemented as a series of arrays representing the sides and rows with each square being represented by a char corresponding to a color. The available actions (moves) were implemented as functions that move the cube from one state to another. Functions to shuffle the cube and execute random moves were implemented. The agent was

implemented to interact with the puzzle and the Q-table and to choose actions to get from the current state to the next state.

Q-learning was used to learn an action-value function, sometimes called a Q-function. The action-value function represents the expected utility of taking a given action when in a given state. A Q-table was used to map states to the values for each of the 12 available actions from each state. The state space is huge, so the Q-table was only pre-populated with the states near the goal state using ADI. As the agent visited a state, if the key for that state was not in the Q-table, it was added with 0 values for each possible action.

The agent then takes turns selecting an action based on the Q-table, executing that action, and entering a new state. In any given state, the agent balanced exploration and exploitation by selecting the greedy action a percentage (epsilon) of the time, and randomly selecting an action a percentage of the time. If there was more than one best action (or they are all 0), the agent randomly selects among the best actions.

In most reinforcement learning applications, the reward is issued at the end of each turn, such as how many points were earned. When solving a Rubik's cube, there is only one goal state, so there is only one reward at the end of the puzzle. In the absence of rewards, the learning algorithm makes random choices. The state space is very large and there is no guarantee that the puzzle will be solved with random actions. This can make a reinforcement learning algorithm very inefficient at solving a Rubik's cube. The problem of sparse reinforcement is overcome by taking the binary reward of the goal state and smearing it out over the states that lead to the goal state, essentially creating a funnel of ever increasing rewards that leads the agent to the goal state. This turns the very small target of the goal space into a very large target that is easier for the learning agent to find.

This "smeared-out" reward space was created by implementing a parameterized ADI. I started at the goal state with a depth parameter and a reward coefficient parameter. I then worked backward from the goal state using the available actions to find the successor states. I applied a reward to each state and action combination that was weighted according to the proximity to the goal state and stored these values in a dictionary for quick look up. The reward was positive if the action took you from the current state to the next state closer to the goal, and the reward

was negative otherwise. ADI made my reinforcement learning agent approximately 6 times more efficient at solving the Rubik's cube from initial scramble depths of 4 to 12.

The Q-learning algorithm parameters of learning rate (eta) and discount factor (gamma) were optimized based on how far from the goal state the agent could start and still solve the puzzle. To optimize my hyperparameters, I tested 0.1 increments of both eta and gamma in the range [0, 0.9]. Starting by optimizing eta, I held gamma, the ADI depth and the number of shuffles for the test cubes constant. The Q-Table was pre-trained using ADI to a depth of 4 from the goal state, and the starting state of both the training and test cubes were 6 moves away from the goal state. The ADI training depth of 4 was chosen to balance computational efficiency and effectiveness. The agent was trained over 5000 epochs then tested with 1000 test cubes. The eta value that resulted in the most solved cubes was then used to test the 10 gamma values.

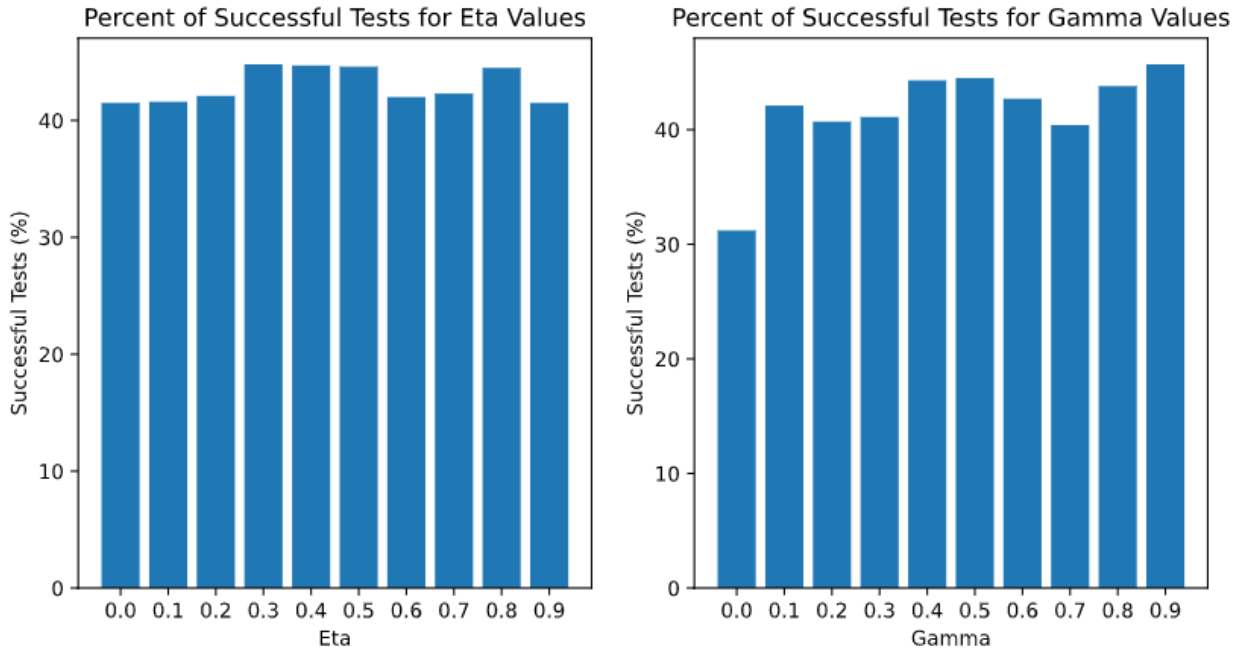
Once the hyperparameters were chosen, I wanted to test how many moves away from the ADI depth the agent would be able to solve the cube. To do this, I trained the agent over 5000 epochs then tested with 1000 test cubes for cubes that were between 0 and 9 moves from the ADI depth.

Experiments were made to add heuristics to Q-table to improve upon ADI. This would give values to actions that lead to states that are known by human Rubik's Cube solvers to be closer to the final state. These states include having a higher number of matching colors on a side, having the corners match the center (X's), having the edge centers match the center (crosses), and having an increased number of solved sides. Implementation of this functionality was not completed or fully evaluated due to a lack of time. This is an area for future investigation.

Results

An eta value of 0.3 resulted in a 44.8% success rate for tests which was the highest among the values tested. The eta values 0.4, 0.5 and 0.8 were the next closest with success rates of 44.7%, 44.6% and 44.5% respectively. The eta values with the lowest success rates were 0 and 0.9 with a success rate of 41.5%. The full results of this experiment can be found in Figure 1.
A gamma value of 0.9 resulted in a 45.7% success rate for tests which was the highest among the values tested. The gamma values of 0.5, 0.4, 0.8 were the next closest with success rates of

44.5%, 44.3% and 43.8% respectively. The gamma value with the lowest success rate was 0, with a success rate of 31.2%. The full results of this experiment can be found in Figure 2.



Figures 1 & 2. Hyperparameter optimization. Successful test percentage after 5000 training epochs.

After hyperparameter tuning, I tested the agent's ability to solve cubes that had been scrambled various amounts. Results of agent's with ADI are shown in Figure 3 and without ADI are shown in Figure 4. As expected in Figure 3, 100% of cubes were solved when they were at the level of pre-training, in comparison to 16.6% without pre-training. As both experiments reached a depth of 9 past the pre-training depth, both agents performed very poorly, with 0.4% and 0.3% of test cubes being solved.

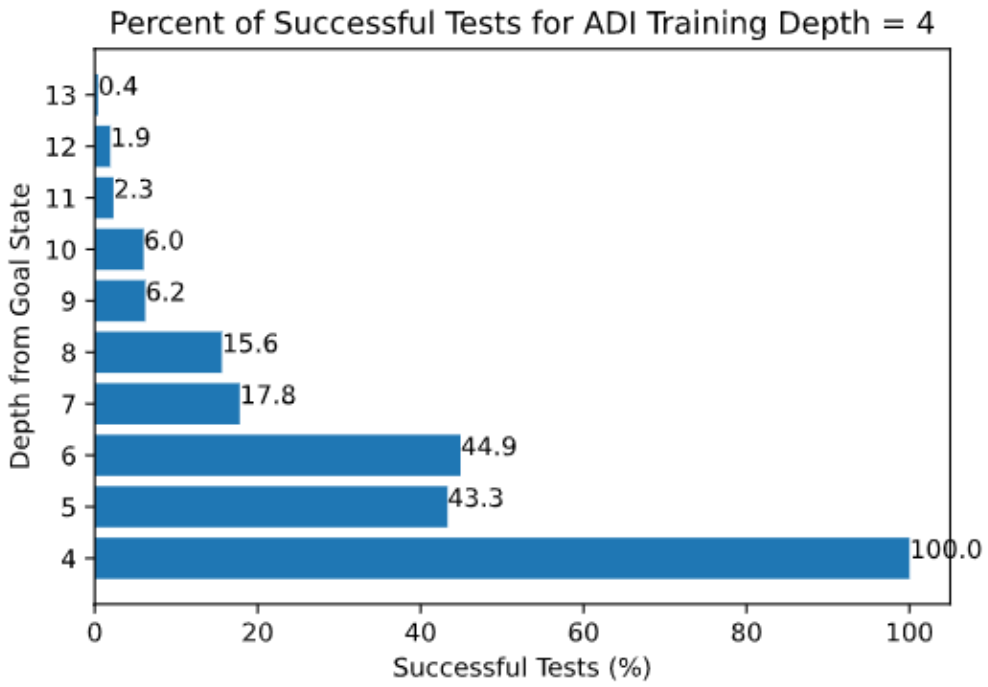


Figure 3. Agent's performance with varying depths from the ADI depth.

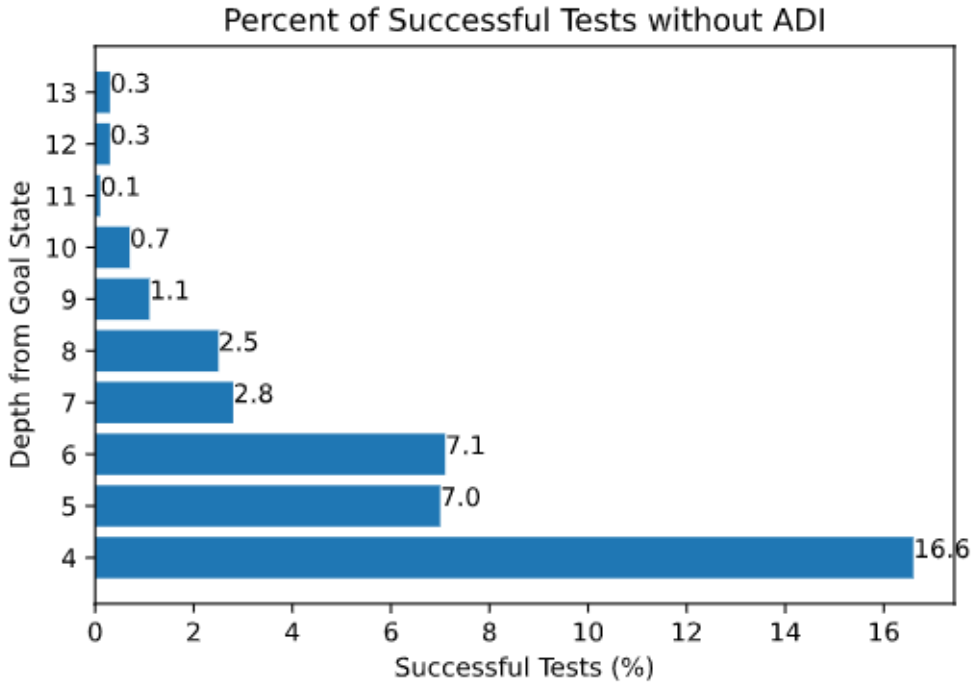


Figure 4. Agent's performance with varying depths without ADI.

Conclusion

The optimal Q-learning parameters found were a learning rate of eta = 0.3 and a discount factor of gamma = 0.9. Autodidactic Iteration improved the reward distribution to make the agent 6 times more likely to solve the puzzle. These results were significant and appear to be an improvement on the prior work.

Source code

https://github.com/vinceolsen/rubiks_cube_learner

Sources

Stephen McAleer, Forest Agostinelli, Alexander Shmakov, Pierre Baldi. “SOLVING THE RUBIK’S CUBE WITH APPROXIMATE POLICY ITERATION”, 2019, https://openreview.net/pdf?id=Hyfn2jCcKm