Rajat Bhageria
ESE 650
Dr. Dan Lee
10 April 2018
Project 5 Report

Reinforcement Learning

1. Value Iteration:

1.1. Method:

1) Find the state transition matrix and the reward matrix
    a) For each possible state, action pair, call step function and average the number of times you see each reward and transition to state s'. Do this 500x to get a reasonable estimate of the transition and reward matrices.
    b) Although I could have manually done this, I decided to just call Step since with 500 iterations, I was finding the same probabilities (0.9 and 0.1) in the correct states.
2) Initialize the values as zero for each state
3) Initialize the policies as random int between 0 and 3 for each state
4) For numIterations = 5000:
    a) While delta > epsilon:
        i) Iterate over each state, action pair
        ii) Find the new state and reward using the step function for that s,a pair
        iii) Find the new max value as the expected future reward. This can be found by for each action, the max value = sum over all possible actions of state transition probability * (reward + discount * value of S').
        iv) Set the current value for the state action pair = to that max value.
        v) Set the current policy for that state = to the action that resulted in the max state
    b) Update delta as maximum of delta and the absolute value of the old values for that state and the current values of the state
5) Convert the Values to q-Values and save

1.2. Results:

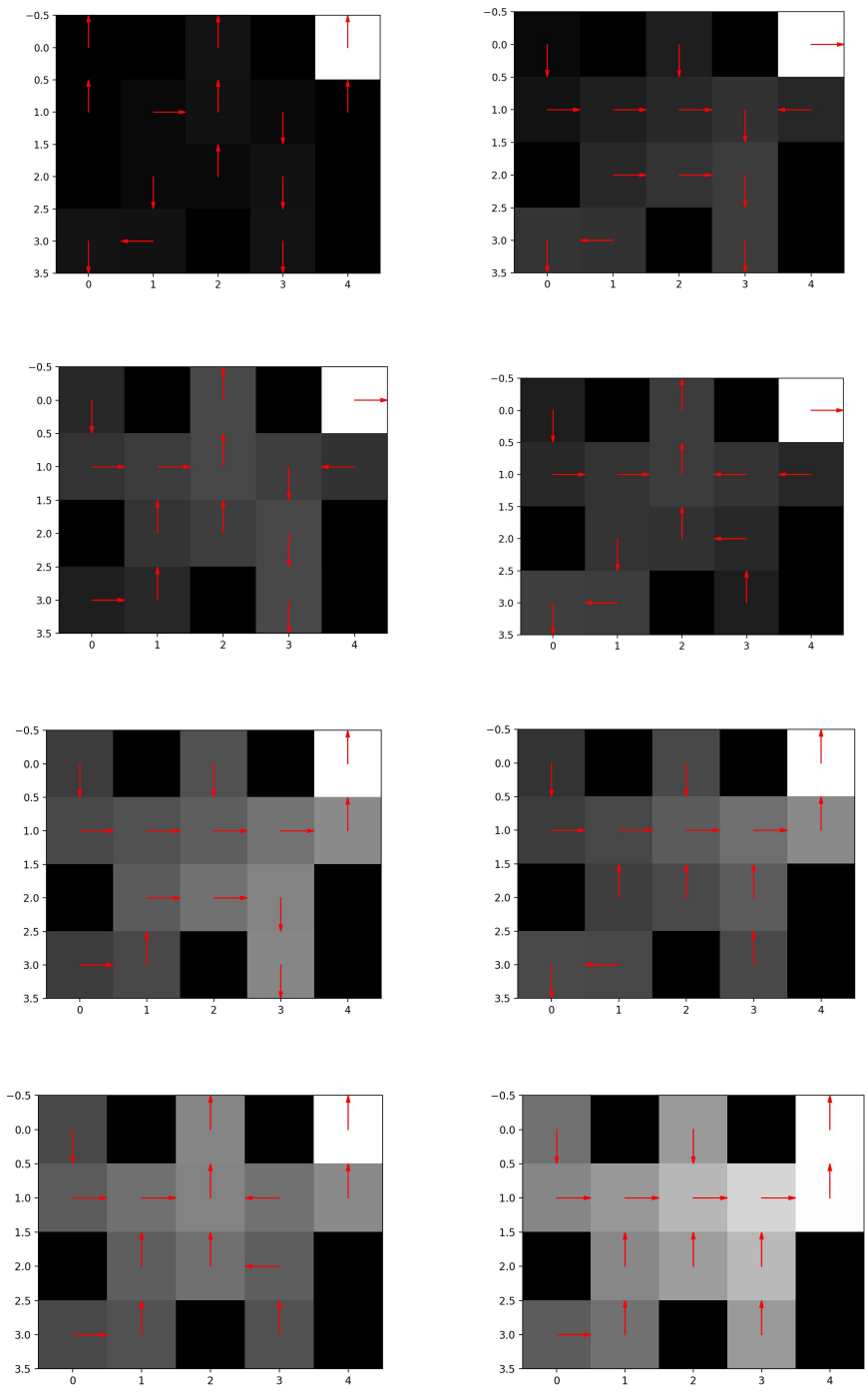


Figure 1: From left to right, top to bottom, the value_plots using VI where the bottom right figure Is the last figure showing the goal

1.2. Analysis:

As seen, the polices and the figures all seem to be correct for Value Iteration. Interestingly, after viewing the q-table itself for value iteration, the q values tend to go very high (higher than say 15 at times) even though the maximum reward is 3.

Value iteration works quite well but the problem is that you of course need the transition matrix and the reward matrix which we won't always have. Additionally, it seems a little more of a roundabout way of solving the control problem than q-learning.

2. Q-Learning:

2.1. Method:

1) Initialize the q values for each state, action pair as a random number between 0 and 2.8.
2) Set the q values for all the states where the agent is at the goal to be 0
3) Set the currentState = initialState
4) For numIter= 5000 iterations:
    a) While notDone (agent hasn't reached the goal):
        i) Epsilon = 100 / (100 + numVisitsCurrentState)
        ii) Pick a random value and if it's < epsilon, chose a random action
        iii) If it's greater than epsilon, find the optimal action based on the qValues as the argmax of all the possible Qs for that state
        iv) Get the current Q value based on the current state and the action.
        v) Take a step and get the reward and next state s'.
        vi) Find the optimal a' based on the argmax of all possible qValues for s'.
        vii) Get the qValuePrime for s' and a'.
        viii) Update the q value table for s and a as currentQValue + alpha*(reward + discount * qValuePrime - currentQValue).
        ix) Set current state = next state

2.2. Results:

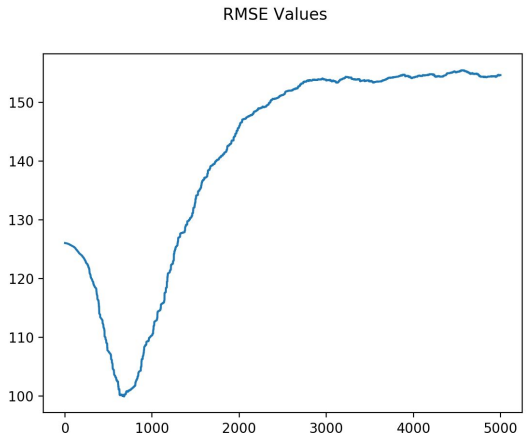


Figure 2: Root Mean Square Values comparing Q Learning and Value Iteration

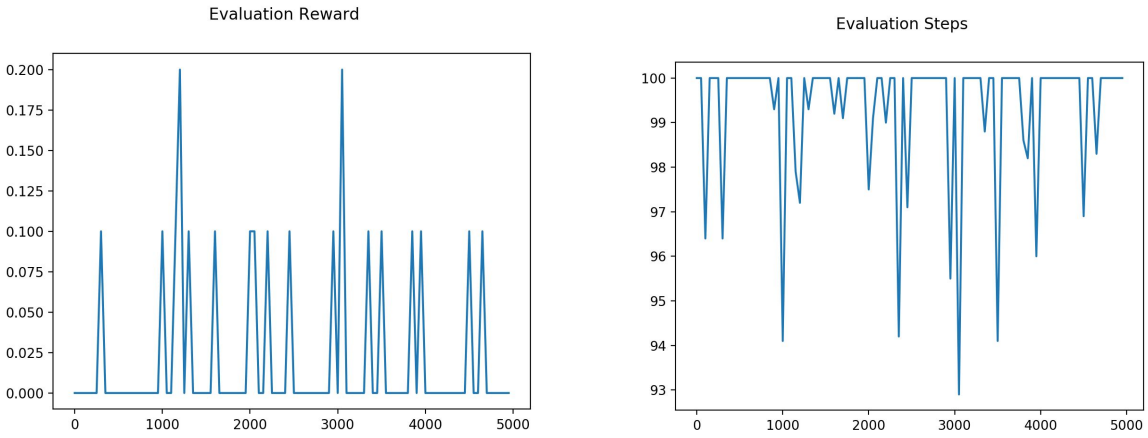
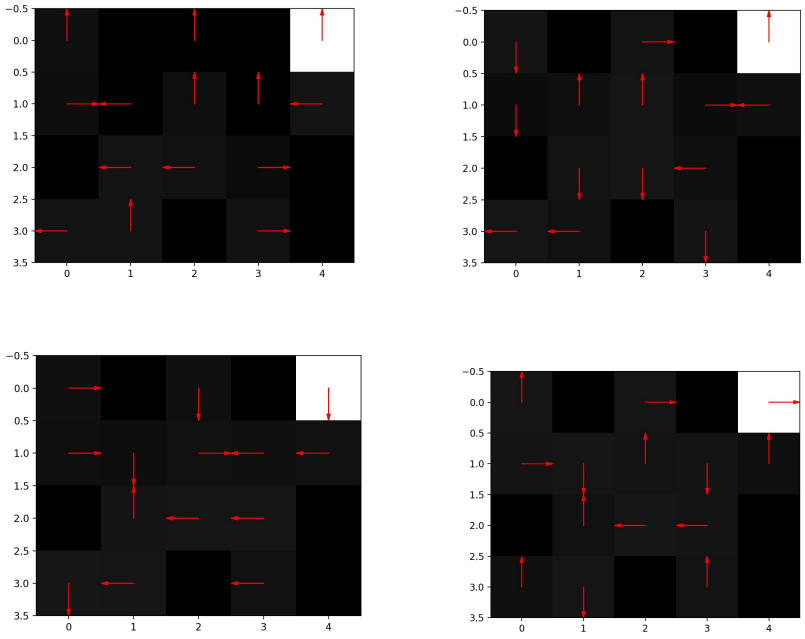


Figure 3: Evaluation Steps and Rewards for Q-Learning



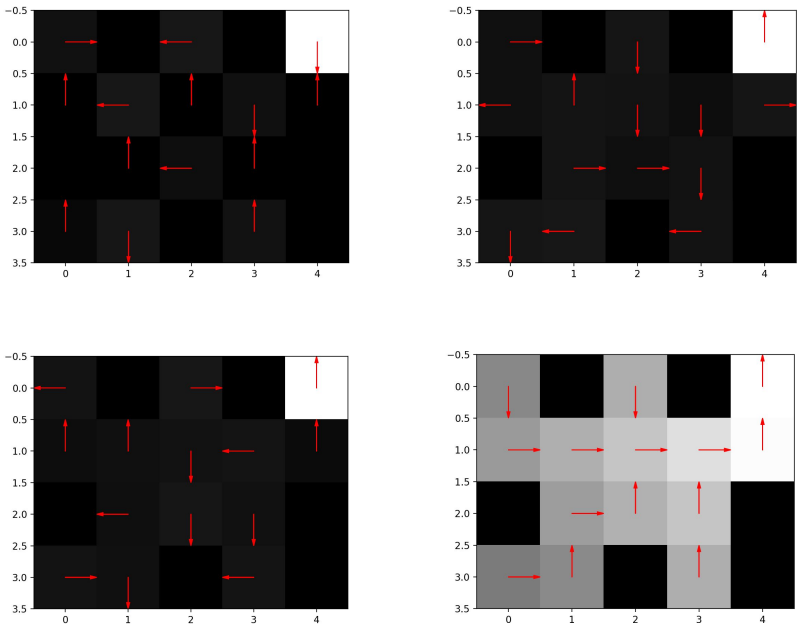


Figure 4: From left to right, top to bottom, the value_plots using Q-Learning where the bottom right figure is the last figure showing the goal

2.2. Analysis:

Q-Learning seems to be an "easier" method of finding the optimal controller. In this case, it seems like while the qValues were more reasonable in magnitude than during VI (their max was around 3 which is the max reward possible), the policies were a little off.

Note that the RMSE plot is interestingly off. As noted in section 1, the reason the magnitudes of the RMSE is so large is because for VI, even though the policies were correct, the magnitudes of the Q-Values were too large. Still, it's curious why the RMSE is increasing as opposed to decreasing with the number of iterations.

Ultimately I tried tuning many parameters and these plots are the best evaluation plots I was able to achieve:

- Changing the learning rate
- Changing the initial values of Qtable
    - Setting them all = 0
    - Setting only the ones that are in the goal state = 0
    - Setting them to random small values between 0.1 and 1
    - Setting them = the same values like .2
- Changing the epsilon for e-greedy policy
    - Setting it high or low and trying out different methods

    - Varying it based on the number of steps within a particular episode
    - Varying it based on the number of times the agent has visited a particular state
- Changing the number of iterations Q-Learning runs

3. OpenAI Gym Acrobat:

3.1. REINFORCE:

3.1.1. Method:

1) currObservation = InitialObservation
2) Instantiate theta with random values
3) For numIter episodes:
    a) For each time step:
        i) Phi = currObservation
        ii) Find the softmax probabilities using the theta function and the phi observation
        iii) Find the optimal action as the argmax of the softmax probabilities over the actions
        iv) Collect the trajectories of state, action, reward triplets for each of the times
    b) Find the gradient of the cost function
        i) For each t in the collected trajectories:
            (1) Find Gt, the expected total reward from t to the end
            (2) Find At= Gt - baseline where baseline is just a running average of the rewards
            (3) Find G-Hat which is just phi - the expected phi over the actionPrimes.
            (4) Reset theta for the action associated with that time as theta = theta + alpha * discount * At * ghat

3.1.2. Results:

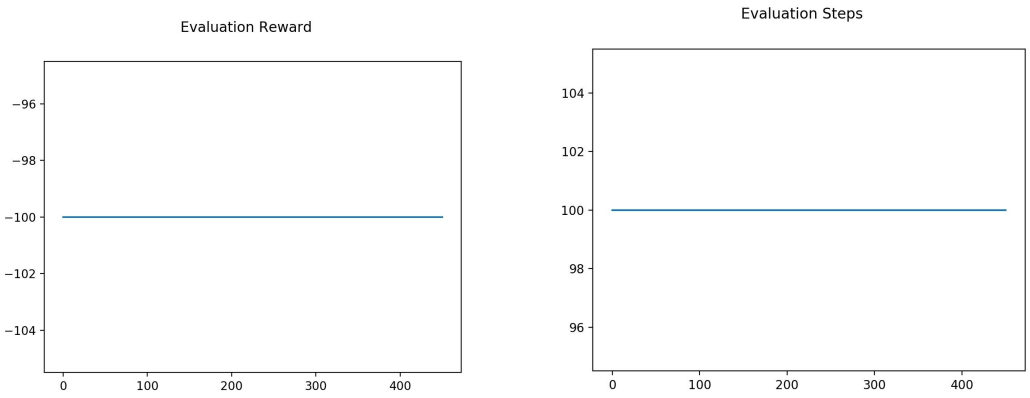


Figure 5: Evaluation steps and rewards for the REINFORCE Algorithm on the acrobat

3.2. Q-Learning:

3.2.1. Method:

- Discretize the state space:
    - For each of the state variables (6 in the case of the Acrobat):
        - Create a linspace from the min variable value to the max variable value with around 10 bins
- Discretize the initialValue (and set to currentValue) using np.digitize()
- Create a qTable of size (10,10,10,10,10,10,3) since we have 6 state variables each with 10 bins and 3 potential actions.
- Then simply run the same q-learning algorithm from part 2 is that anytime we get a new state vector s', we have to digitize it using np.digitize().

3.2.2. Results:

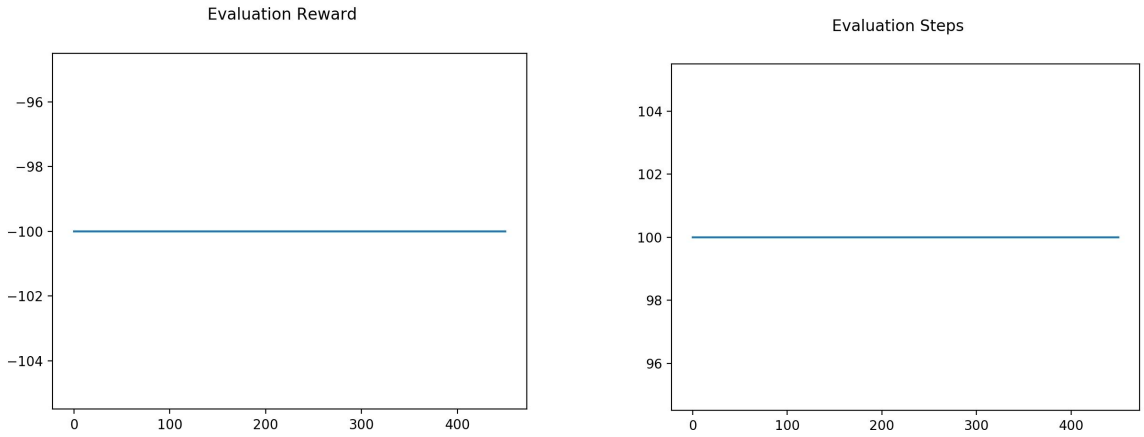


Figure 6: Evaluation steps and rewards for the Q-Learning Algorithm on the acrobat

3.3. Analysis:

It seemed like in this case, the Q-Learning was worked a lot better than the REINFORCE algorithm in getting to the goal state (or at least it converged faster). Of course, it seemed like the evaluation method wasn't working accurately so it's hard to tell.

Both algorithms didn't reach the goal in 500 iterations and both were simply shaked the first and second joints in a seemingly random way (since the epsilon was high and variant on time). Still it did seem like Q-learning was going to achieve the result faster since it was more stable and exploring more and inching towards the goal faster.

3.3. Value Iteration Question:

We cannot use PI/VI since we only have a sequence of state, action, reward triplets but we don't have a full transition and reward matrix per se though perhaps you could argue we could “guess” it by doing the step function over all the possible state and action pairs.

4. OpenAI Gym Car:

4.1 REINFORCE:

4.1.1. Method:

The method is exactly the same as the REINFORCE for the car problem except that we have to discretize the action space as well:

- We find the minimum of the action space and the max and create a linspace with say 100 bins.
- Then anytime we have a new action, we can just use np.digitize() to find the integer action to take and store it in an array.

Note that we use the softmax function approximator for state space and discretize the action space.

4.1.2. Results:

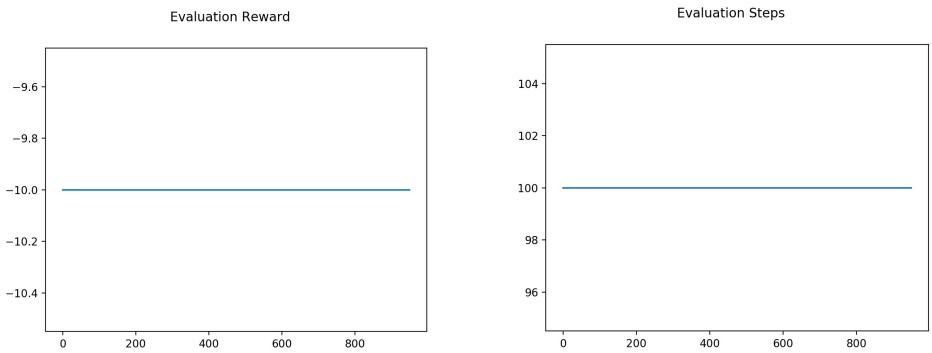


Figure 7: Evaluation Steps and Reward for the REINFORCE algorithm on the car

4.2. Q-Learning:

4.2.1. Method:

The method for Q-Learning for the Car is the exact same as the method for Q-Learning for the Acrobat as described in 3.2.1 but the only difference here is that we also have to discretize the actionSpace as well demonstrated in section 4.1.1. Note that for QLearning, we discretize both the action and state space. Of course also play around with parameters.

4.2.2. Results:

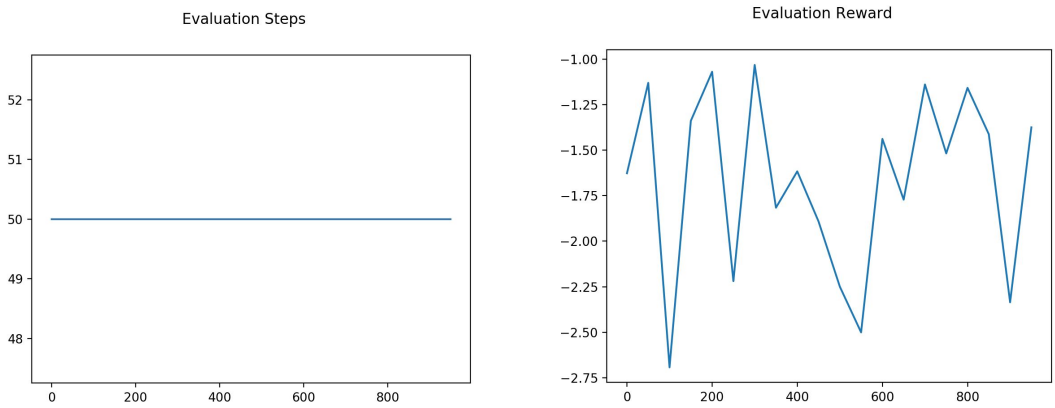


Figure 8: Evaluation Steps and Reward for the Q-Learning algorithm on the car

4.3. Analysis:

It seemed like in this case, the Q-Learning was worked a lot better than the REINFORCE algorithm in getting to the goal state (or at least it converged faster). Of course, it seemed like the evaluation method wasn't working accurately so it's hard to tell.

Another interesting thing to notice is that while QLearning has a high chance of reaching the goal, for REINFORCE, the policy seems to always push the car towards the left but it never ends up going beyond the trough of the valley.

I adjusted many parameters to get to the current results:

- Learning rate
- Epsilon from fixed to varying with time
- The number of iterations

However, I didn't test any other representations outside of softmax.

5. Future Work:

There are many future things to implement / fix:

- Reduce the magnitude of the q Values for the VI
- Ensure the correct policy is working for q Learning
- Fix the evaluation plots for the continuous REINFORCE
- Do hyper-parameter tuning and edit the feature vector till the acrobat can meet the goal for REINFORCE and Q-Learning
- Ensure that the car reaches the goal for REINFORCE
- Implement deep neural nets for REINFORCE to have a better estimate of thetas
- Test other feature vectors like RBF and Fourier for REINFORCE.
- Try using other action policies outside of e-greedy

6. Videos:

I've included four .mov files for a glimpse of the learning process for the car and acrobat (there's a version for the Q-Learning and the REINFORCE algorithm). Note that these videos are at the beginning of the learning process.