Reinforcement Learning Homework 6

Deadline 26th November 2024
Students Sahiti Chebolu, Surabhi S Nath, Xin Sui

Question 1.

(a)

In linear function approaximation, we have:

$$x(S) = \begin{pmatrix} x_1(S) \\ x_2(S) \\ . \\ . \\ x_n(S) \end{pmatrix}$$

In tabular methods, each state is represented by a unique entry in a table. This can be represented as a feature representation with only one feature *active* and the rest *inactive*. In other words, we can use a one-hot representation of size = n (number of states).

$$x(S = s_1) = \begin{pmatrix} 1 \\ 0 \\ . \\ . \\ 0 \end{pmatrix}, x(S = s_2) = \begin{pmatrix} 0 \\ 1 \\ . \\ . \\ 0 \end{pmatrix} \ . \ . \ ., x(S = s_n) = \begin{pmatrix} 0 \\ 0 \\ . \\ . \\ 1 \end{pmatrix}$$

The value for the states can still be accessed with $x(S)^T w$.

(b)

$x_i(s) = \prod_{j=1}^{k} s_j^{c_{i,j}}$

$c_{i,j}$ can take values from the set {0, 1, ... n}.
Therefore, each $c_{i,j}$ has n+1 possibilities. Since the product is over a k dimensional state space, one of the n+1 possibilities is sampled k times. Therefore the dimensionality of the feature space will be: $(n + 1)^k$.

Question 2.

For an environment as rich as Super Mario, the feature space can be quite vast. Ideally, neural networks can be used for automated feature extraction. But here are some potential interpretable features that could be used:

1. x_1 = Mario's x position
2. x_2 = Mario's y position
3. x_3 = Mario's velocity
4. x_4 = Mario's direction of movement
5. x_5 = Nearest reward's (coin/power-up) x position

6. x_6 = Nearest reward's (coin/power-up) y position
7. x_7 = Distance to reward
8. x_8 = Angle to reward (coin/power-up)
9. x_9 = Nearest opponent's (turle/mushroom) x position
10. x_{10} = Nearest opponent's (turle/mushroom) y position
11. x_{11} = Nearest opponent's (turle/mushroom) velocity
12. x_{12} = Nearest opponent's (turle/mushroom) direction
13. x_{13} = Distance to nearest opponent
14. x_{14} = Angle to nearest opponent

Question 3.

(a)

The trajectory of the system (angular position, angular velocity, action (torque), reward) are plotted separately against time:

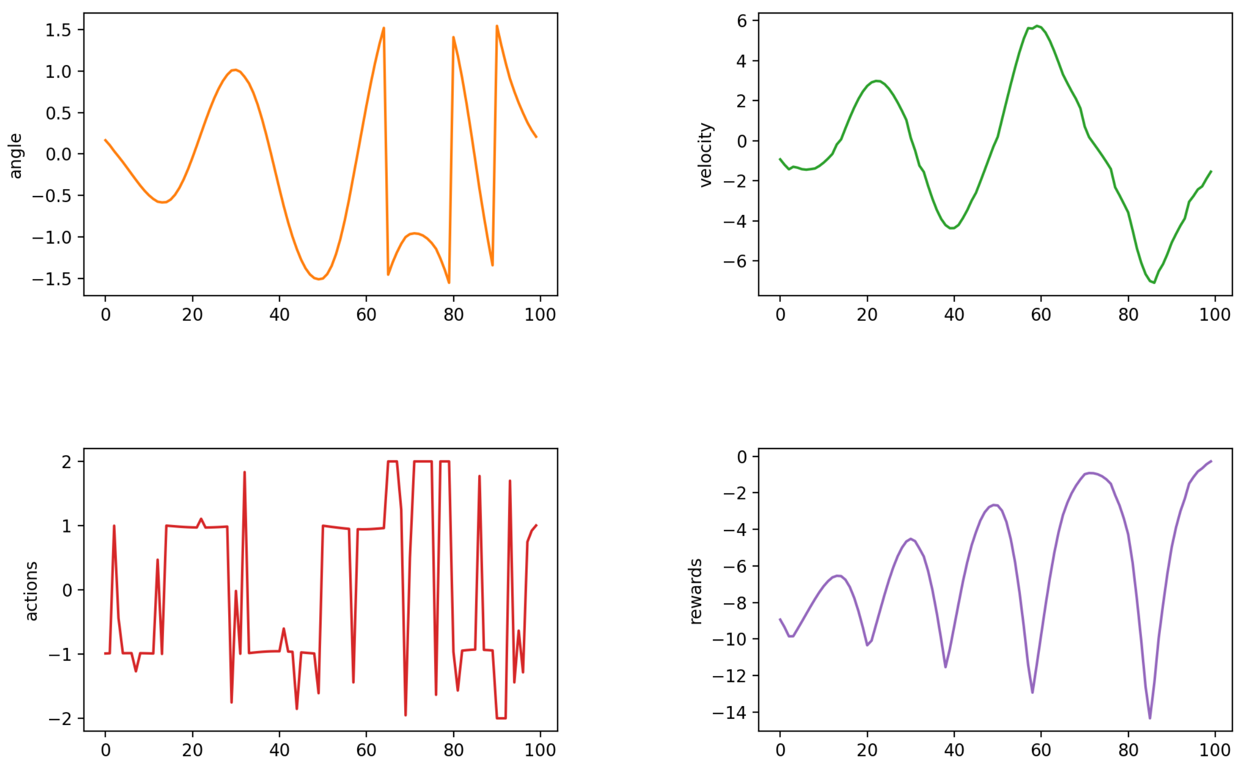


Figure 1: Trajectory of system run for 100 timesteps

(b)

Code for fitting value function modified in jupyter-notebook

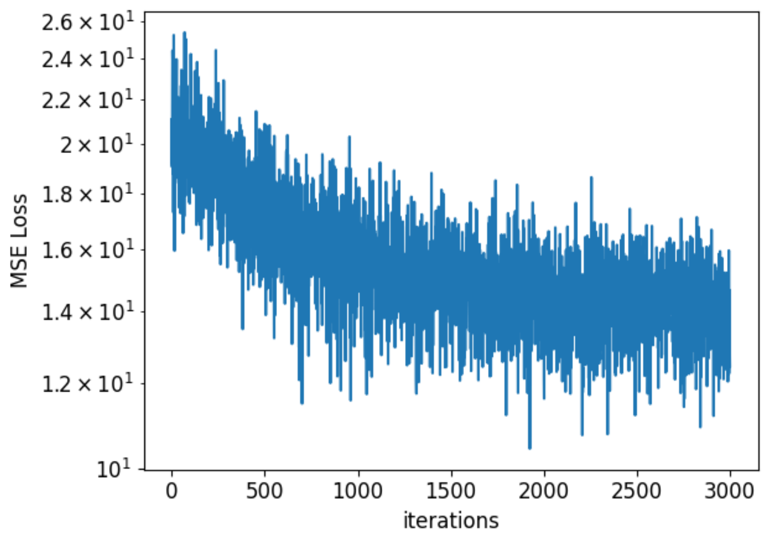
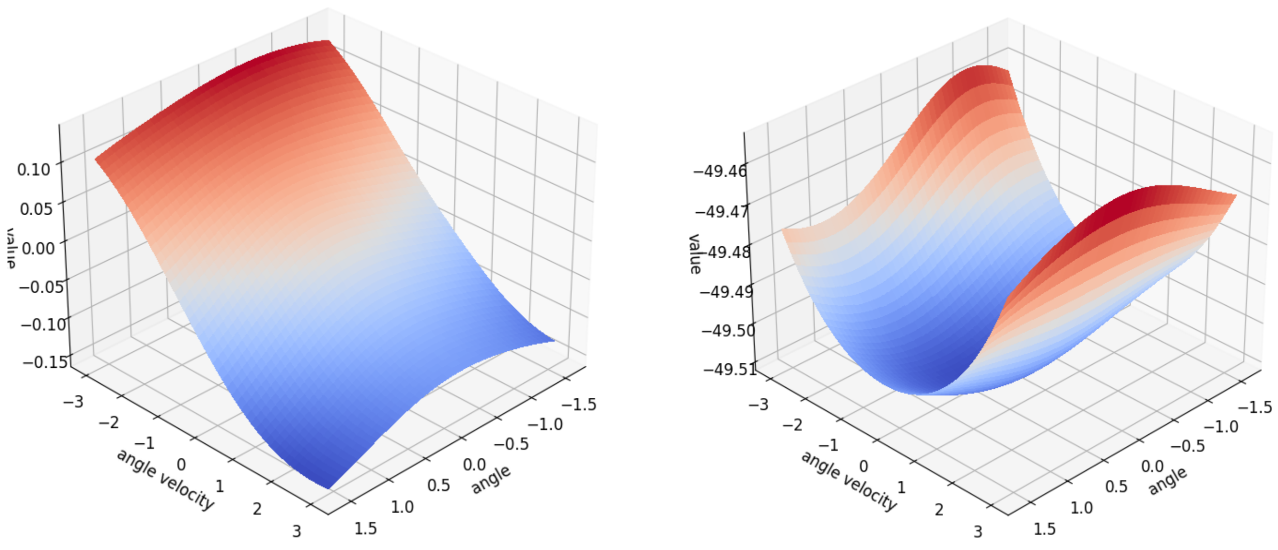


Figure 2: MSE loss for value approximation network through 3000 iterations of learning for $\gamma = 0.95$

(c)



(a) Value function before learning

(b) Learnt value function after 3000 iterations

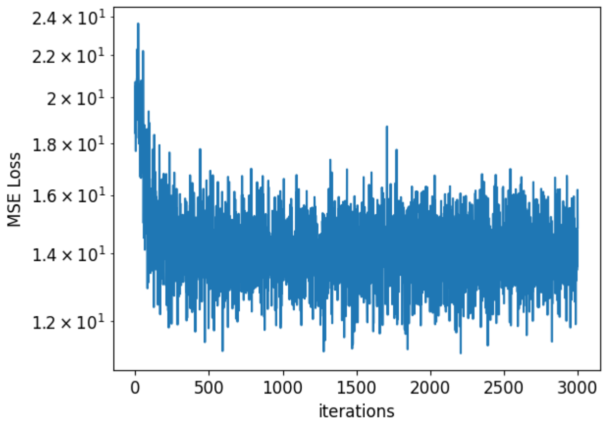
Figure 3: Approximated value function for $\gamma = 0.95$ before and after learning

(d)

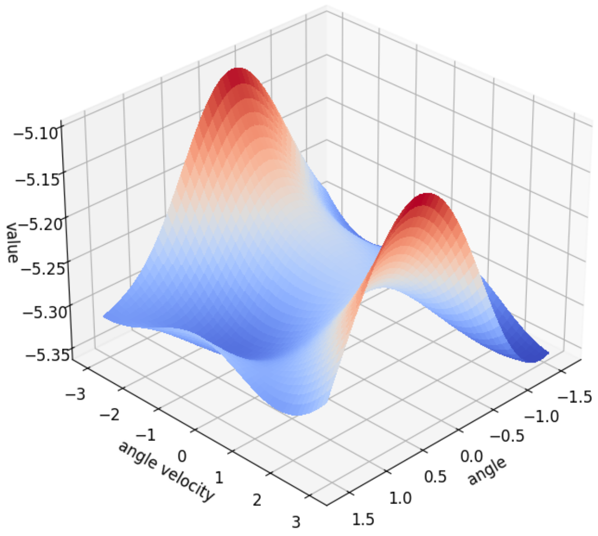
In the given policy, the amount of torque applied depends on the angular velocity of the pendulum. While applying torque is immediately costly, it is necessary to keep the pendulum at angle = 0. Hence, the value function is higher for higher magnitude of velocity, especially when angle is close to 0. For very low velocity and angle close to zero, it is the lowest because, here the policy specifies low torque hich will take the pendulum to undesirable positions.

(e)

In contrast, when discount factor is low, the future is discounted more and the value function reflects the immediate reward function more. Hence, in this case, value is higher when angle is closer to 0 (since immediate reward is highest here), although it is still higher for higher magnitude of angular velocities. The value function is lowest for very high velocities and angles far away from 0. This is because the immediate reward is the lowest here. Further, the value function converges more quickly here compared to discount factor = 0.95.



(a) MSE loss for value approximation network for $\gamma = 0.5$



(b) Learnt value function after 3000 iterations

Figure 4: Value approximation for $\gamma = 0.5$