Reinforcement Learning and Optimal Control for Robotics ROB-GY 6323 Exercise Series 4

December 8, 2024

Shantanu Ghodgaonkar
Univ ID: N11344563
Net ID: sng8399

Question 1 Implement the Q-learning algorithm.

Answer We have been given the algorithm as,

Algorithm 1 Q-Learning Algorithm

1: **for** each episode **do**
2: Initialize the episode: $x_0 = [0, 0]$
3: **for** each step of the episode **do**
4: Select u_n using an ϵ-greedy policy
5: Compute the next state: x_{n+1}
6: Compute the target: $y_n = g(x_n, u_n) + \alpha \min_a Q(x_{n+1}, a)$
7: Perform one SGD step on the neural network parameters to minimize:

$$(Q(x, u) - y_n)^2$$

8: **end for**
9: **end for**

Implementing this in code, we have the loop as shown below. The code can be found in the attached jupyter notebook as well.

```
# Training loop with progress visualization using tqdm
for _ in tqdm(range(MAX_ITER), desc='Training in Progress'):
    # Initialize the state vector xi to [0, 0] and
    # move it to the selected device
    xi = torch.tensor(np.zeros((2,)), device=device, dtype=torch.float)

    # Loop over each step within the episode
    for _ in range(N):
        # Perform a forward pass through the Q-network to
        # get Q-values for current state xi
        forward_pass = q_function.forward(xi.unsqueeze(0))

        #   -greedy policy for action selection
        if torch.rand(1).item() < epsilon:
            # Exploration: select a random action
            # from the possible controls
            ui = torch.randint(0, 3, (1,)).item()
        else:
            # Exploitation: select the action with
            # the minimum Q-value (assuming minimization)
            ui = torch.argmin(forward_pass).item()

        # Compute the next state and target value
        # without tracking gradients
        with torch.no_grad():
            # Apply the selected action to get the next state
            # using the pendulum's dynamics
            xip1 = torch.tensor(pendulum.step(
                x=xi.cpu().numpy(),
                u=possible_controls[ui]),
                device=device,
                dtype=torch.float)

            # Calculate the target value y_i
            # y_i = g(x_i, u_i) + alpha * min_a Q(x_{i+1}, a)
            yi = torch.tensor(
                (g(x=xi.cpu().numpy(), u=possible_controls[ui]) +
                    (alpha * torch.min(
```

```
                q_function.forward(xip1.unsqueeze(0))).item())),
            device=device, dtype=torch.float)


        # Compute the loss between the current Q-value and the target y_i
        # forward_pass.squeeze()[ui] extracts the Q-value
        # for the selected action
        loss = loss_fn(forward_pass.squeeze()[ui], yi)

        # Zero the gradients before backpropagation
        optimizer.zero_grad()

        # Perform backpropagation to compute gradients
        loss.backward()

        # Update the Q-network parameters using the optimizer
        optimizer.step()

        # Update the current state to the next state
        xi = xip1
```

Question 2 Test that it works with and without pushes.

Answer This has been tested and the videos are present in the submission.

Question 3 Plot the cost per episode (to visualize learning)

Answer The plot is shown below -

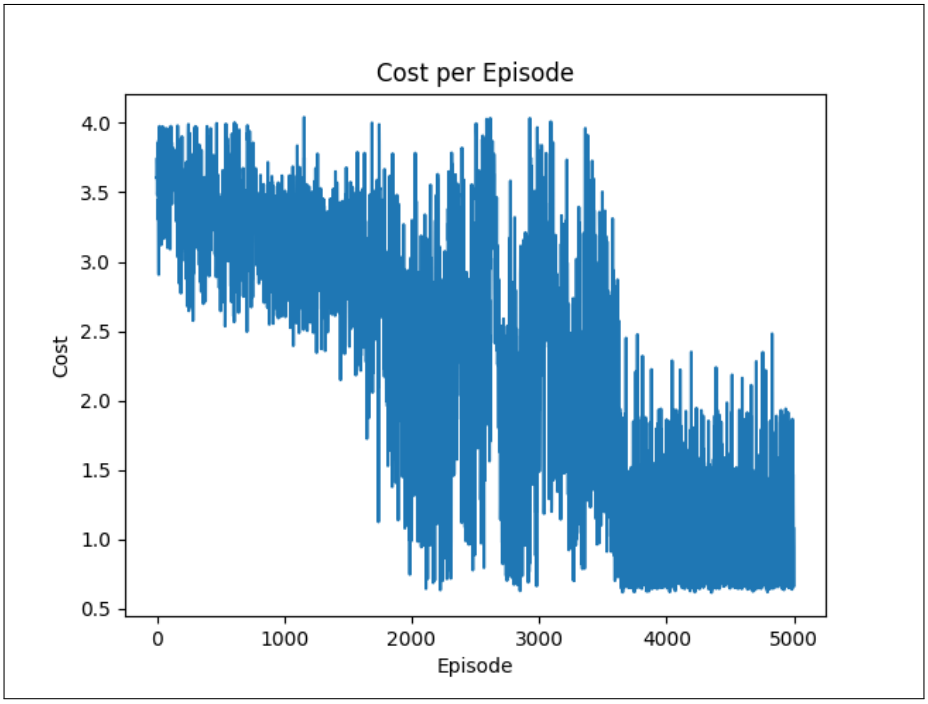

Figure 1: Cost Per Episode

Question 4 Plot the learned value function (in 2D as a function of pendulum position and velocity) as well as the policy.

Answer The plots are shown below -

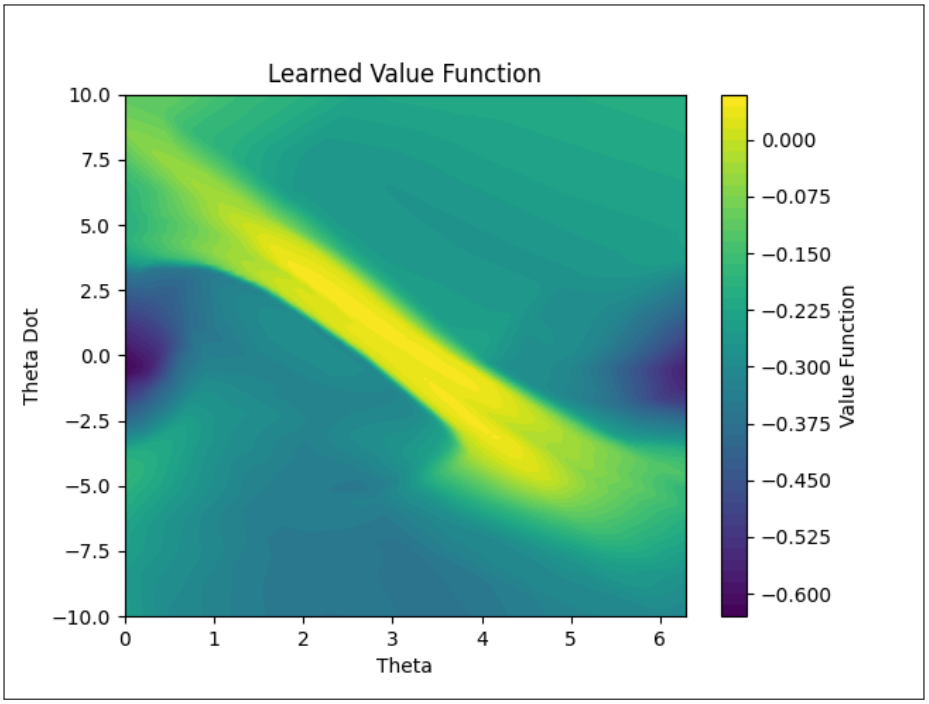

Figure 2: Value Function Plot

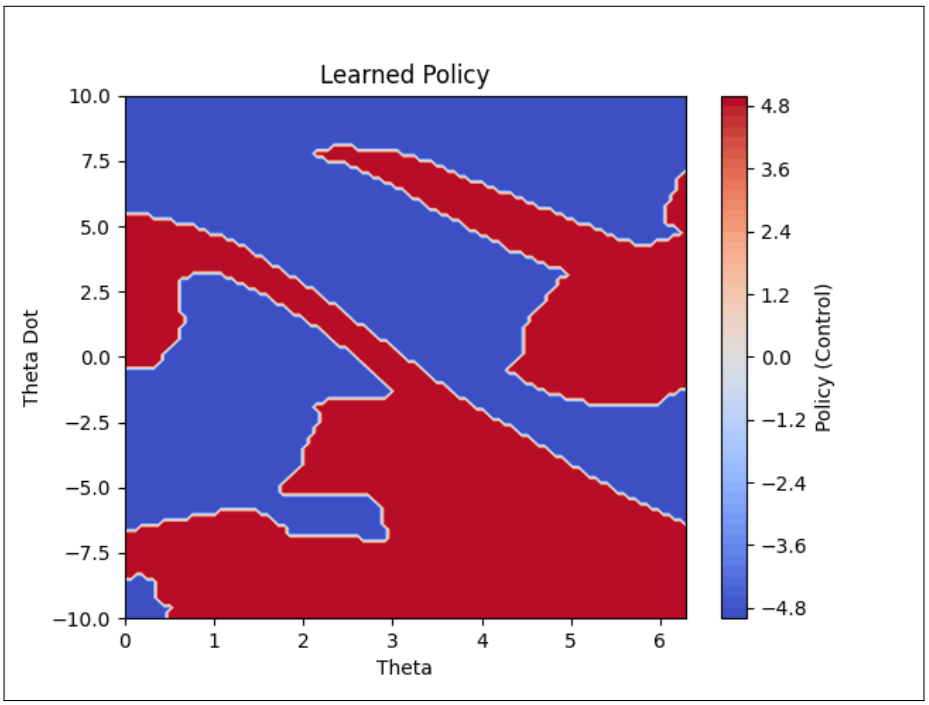

Figure 3: Learned Policy