Statistical Techniques in Robotics (16-831, S22) Lecture #15 (Wednesday, March 16)

Value-Based Model-Based Control

Lecturer: Alex LaGrassa *Scribes: Alex Pletta, Benjamin Younes*

1 Review

In the last lectures, we previously learned about Markov Decision Processes (MDP).

1.1 Markov Decision Process

A Markov Decision Process [1] is a framework for modeling how an agent can decide to take actions in given states to obtain a reward.

1.1.1 Notation Definition

$s \in \mathcal{S}$	State
$a \in A_s$	Action
$p(s'\|s,a)$	State transition dynamic
$r(s',s,s)$	Reward function (general case)
$p_0(s)$	State prior
$\pi(a\|s)$	Policy
γ	Discount factor

An MDP trajectory is composed of a sequence of states and actions taken by the agent for a time horizon of T.

$$\zeta = (s_0, a_0, s_1, a_1, ..., s_T, a_T) \tag{1}$$

One can form the probability of a state-action trajectory $p(s_0, a_0, s_1, a_1, ..., s_T, a_T)$ and an associated reward probability of that trajectory $r(s_0, a_0, s_1, a_1, ..., s_T, a_T)$. This can be factorized in many ways, most traditionally as the following:

The traditional probability factorization is:

$$p(s_0, a_0, s_1, a_1, ..., s_T, a_T) = p_0(s_0) \prod_t p(s_{t+1}|s_t, a_t) p(a_t|s_t) \tag{2}$$

where $p_0(s_0)$ is the prior state, $p(s_{t+1}|s_t, a_t)$ is the state transition dynamic, and $p(a_t|s_t)$ is the policy.

The traditional reward factorization can take several forms, including:

1. Reward as a function of the previous state, current state, and action

$$r(s_0, a_0, s_1, a_1, ..., s_T, a_T) = r(s_0, a_0, s_1) + r(s_1, a_1, s_2) + ... \tag{3}$$

2. Reward as a function of the previous state and action

$$r(s_0, a_0, s_1, a_1, ..., s_T, a_T) = r(s_0, a_0) + r(s_1, a_1) + ... \tag{4}$$

3. Reward as a function of the state

$$r(s_0, a_0, s_1, a_1, ..., s_T, a_T) = r(s_0) + r(s_1) + ... \tag{5}$$

An MDP policy $\pi(a|s)$ is a strategy of selecting actions at each state that result in a trajectory. The state transition dynamic describes the likelihood of arriving at a certain state after following a certain action. This captures the unmodeled system dynamics, such as those introduced by disturbances, that could lead to a transition to an unexpected state. The state transition dynamic $p(s'|s, a)$ is the probability of arriving at state s' after taking action a at state s.

An MDP reward performance can be evaluated using the state value function, which is the sum of total expected reward over a trajectory from following a policy π for starting in a state s_0. Note that each reward is calculated depending on the selected factorization. The factorization is based on the Markov assumptions inherent in MDP problems.

$$V^\pi(s) = \mathbb{E}_p[r_0 + r_1 + ...|s_0 = s] \tag{6}$$

The value function can be defined over different reward horizons:

1. Infinite horizon return
$$V^\pi(s) = \mathbb{E}_p[r_0 + r_1 + ...|s_0 = s] \tag{7}$$

2. Finite horizon return
$$V^\pi(s) = \mathbb{E}_p[r_0 + r_1 + ... + r_T|s_0 = s] \tag{8}$$

3. Infinite horizon discounted return

$$V^\pi(s) = \mathbb{E}_p[r_0 + \gamma r_1 + \gamma^2 r_2 + ...|s_0 = s] \tag{9}$$

The most common reward horizon is the infinite horizon discounted return, with a discount factor γ that typically is $\gamma \in [0.9, 0.99]$.

The MDP reward performance can also be defined using the State-Action Value Function. The State-Action Value Function defines the expected return of a trajectory given an initial state s and initial action a with a policy π. This function assumes the action a is taken at state s and then the policy π is followed for the rest of the trajectory. Within the infinite horizon discounted return formulation, the State-Action Value Function can be written as:

$$Q^\pi(s, a) = \mathbb{E}_p[r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + ...|s_0 = s, a_0 = a] \tag{10}$$

The relationship between the Value Function and State-Action Value Function is linked by the sum over the possible initial actions with the given policy:

$$V^{\pi}(s) = \sum_{a} \pi(a|s) Q^{\pi}(s,a) \tag{11}$$

2 Summary

In today's lecture, we discussed the relationship between MDP's and closed-loop control to find optimal policies for a given task through policy iteration and value iteration. In class we covered an example that used a tabular environment with discrete states and actions (e.g. a checkerboard where the agent can move up, down, left, right, or stay in-place), but the concept of MDP's and optimal policies extend to continuous representations as well.

2.1 Open Loop vs. Closed Loop Control Brief Summary

In class we used open vs. closed loop control with a visual demonstration of pouring a proper amount of liquid into a glass. The demonstration was structured such that the lecturer was given two attempts to pour liquid into the glass, where the open loop case did not have visual feedback (the lecturer had their eyes closed) and the closed loop case did have visual feedback (the lecturer had their eyes open).

This demonstrated that open loop control can have reasonably good performance it the model of the environment is good, but that closed loop control is more robust and can often be much more accurate. In the context of reinforcement learning, we will use closed loop control to update our model after receiving reward feedback from the environment.

2.2 Bellman Equation

The Bellman Equation is a recursive dynamic programming equation that derives the optimal reward over all actions that could be taken at a given state [2]. This equation is defined as the following for a given policy π.

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a)(R(s,a) + \gamma V^{\pi}(s')) \tag{12}$$

The Bellman Optimality Equation is a special form of the Bellman Equation that holds for the *optimal* policy π^* that maximizes the expected reward.

$$V^{\pi^*}(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a) + \gamma V^{\pi^*}(s')) \tag{13}$$

We will use the Bellman Optimality Equation to solve for optimal policies given states through both policy and value iteration.

2.3 Model-Based Value-Based Control

We will model and update our system using closed loop control over both policy and value iteration. In both approaches, we will continuously loop over all states and improve our policy. Our control loop will cycle back and forth between a prediction stage and then an evaluation stage.

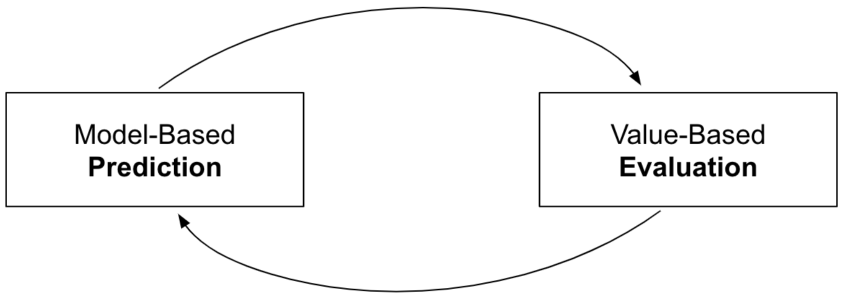

Figure 1: Model-Based Value-Based Closed-Loop Control.

2.3.1 Policy Iteration

In Policy Iteration [1], the agent evaluates the value V or action value Q for a given policy, and then improves the policy using the value V or action value Q.

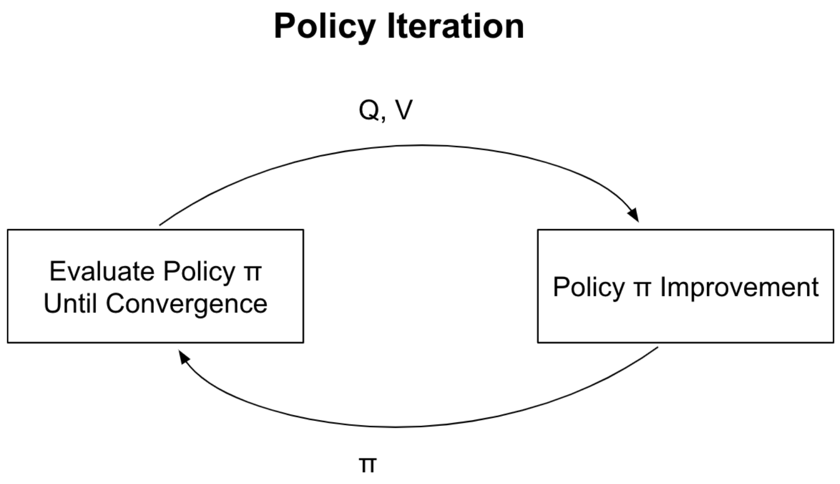

Figure 2: Policy Iteration Closed-Loop Control.

- Evaluation: The iterations begin with a random policy π. This policy is then evaluated over all states. Note that the policy evaluation needs to be repeated until the values are stable for that policy.
- Improvement: The policy actions are then updated using the calculated values.

This control loop iterates between evaluation and policy improvement updates until convergence, which occurs when the policy stays the same within some threshold according to Algorithm 1 below.

Algorithm 1 Policy Iteration Algorithm

```
 1: while $\pi$ is unchanged                    ▷ Loop until final policy convergence do
 2:   $V \leftarrow \text{rand}(\mathbb{R})$
 3:   $V' \leftarrow \text{rand}(\mathbb{R})$
 4:   while $\max_s|V(s) - V'(s)| \leq \epsilon$          ▷ Iterate with policy until values converge do
 5:     $V' \leftarrow V$
 6:     for $s \in \mathcal{S}$ do
 7:       $Q(s,a) = r(s) + \gamma \sum_{s'} p(s'|s,a)V'(s') \quad \forall a$
 8:       $V(s) = \sum_a \pi(a|s)Q(s,a)$
 9:     end for
10:   end while
11:   for $s \in \mathcal{S}$                            ▷ Update the policy to improve do
12:     $\pi(s) = \text{argmax}_a Q(s,a)$
13:   end for
14: end while
```

It can be proven that the converged policy iteration algorithm will arrive at the optimal policy because every iteration improves the policy as defined in the update step. We will show this by comparing the value function V to the action value function Q using policy π and updated policy π'.

Note that as defined by the value and action value functions, $V^\pi(s) \leq Q^\pi(s, \pi')$. We can leverage this inequality to then relate $V^\pi(s)$ to $V^{\pi'}(s)$ by using induction over the state action pairs.

$$\begin{aligned}
V^\pi(s) &\leq Q^\pi(s, \pi') && (14)\\
&= \mathbb{E}_a[r(s,a) + \gamma V^\pi(s')] && (15)\\
&\leq \mathbb{E}_a[r(s,a) + \gamma Q^\pi(s', \pi')] && (16)\\
&= \mathbb{E}_{a,a'}[r(s,a) + \gamma r(s',a') + \gamma^2 V^\pi(s'')] && (17)\\
&\leq ... && (18)\\
&\leq \mathbb{E}_{a,a',a''}[r(s,a) + \gamma r(s',a') + \gamma^2 r(s'',a'') + ...] && (19)\\
&= V^{\pi'}(s) && (20)
\end{aligned}$$

We can then say that

$$V^\pi(s) \leq V^{\pi'}(s) \tag{21}$$

so each iteration of the policy update will produce a new policy that has a value that is at least as good as the previous policy. Therefore, once the policy converges to a stable policy that final policy will be the optimal policy.

■

2.3.2 Value Iteration

In Value Iteration [1], the value V or action value Q and the policy π are calculated at the same time. The policy π and the state values are initialized randomly.

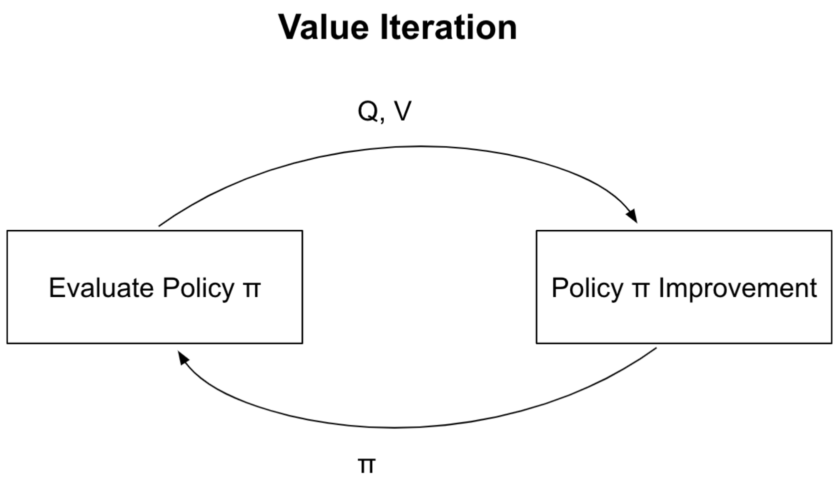

Figure 3: Value Iteration Closed-Loop Control.

- Evaluation: The evaluation step calculates the value function using the current best action for each state.
- Improvement: The improvement step takes the best action from each state that maximizes the Bellman Optimality Equation.

The value iteration algorithm (Algorithm 2) repeats until the values for each state no longer change. At this point the actions in each state are the optimal policy, because each action was calculated using the Bellman Optimality Equation.

Algorithm 2 Value Iteration Algorithm

```
1: $\pi \leftarrow \mathcal{A}$
2: $V \leftarrow \text{rand}(\mathbb{R})$
3: $V' \leftarrow \text{rand}(\mathbb{R})$
4: while $\max_s |V(s) - V'(s)| \leq \epsilon$          ▷ Iterate until values converge do
5:     $V' \leftarrow V$
6:     for $s \in \mathcal{S}$ do
7:         $Q(s,a) = r(s) + \gamma \sum_{s'} p(s'|s,a)V'(s') \quad \forall a$
8:         $\pi(s) = \text{argmax}_a Q(s,a)$
9:         $V(s) = Q(s,\pi(s))$                          ▷ Update values
10:    end for
11: end while
```

2.4 Policy vs. Value Iteration Takeaways

As seen, policy and value iteration both converge to the optimal policy. The selection to use one approach vs. the other depends on the problem being solved. General guidelines are that value iteration can typically converge faster, unless it is very expensive to compute the value V and action value Q. In this case, it may be faster to use policy iteration.

In the real-world, additional challenges can be that the problem may only be partially observable and that the system dynamics may not be modeled perfectly. However, the optimal policy can still often be discovered even with modeling imperfections, thanks to the closed-loop control architecture.

2.5 Connection to Reinforcement Learning

Although not covered directly in the lecture, we will now make the connection between the model-based value-based control and reinforcement learning (RL). We have been discussing our control loop in terms of an agent making an action and receiving a reward. We covered how to use policy and value iteration in combination with the Bellman Equation and Bellman Optimality Equation to find an optimal policy, with given transition dynamic and reward. We used a closed-loop control feedback to use this reward in the updating steps of the iterations.

In our formulation, we used a state transition dynamic and model of which state-action pairs would result in certain amounts of reward. This can also be called Model-Based RL, but reinforcement learning can be formulated in such a way that the agent does not necessarily need to use the transition dynamics and/or reward, as long as the agent finds some policy, in so-called Model-Free RL. In Model-Based RL, the agent leverages it's knowledge of the environment and reward model to improve the reward from its policy faster and more accurately. However, in Model-Free RL the agent can compute its policy using only the reward from that policy directly with no other modeling understanding. This can be more flexible to unknown or changing dynamics but can also be more challenging to train.

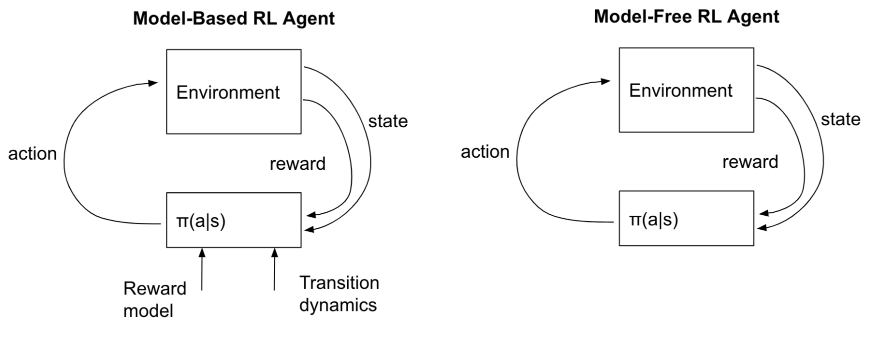

Figure 4: Model-Based vs. Model-Free Reinforcement Learning.

References

[1] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[2] R. Bellman, "On the theory of dynamic programming," *Proceedings of the National Academy of Sciences*, vol. 38, no. 8, pp. 716–719, 1952.