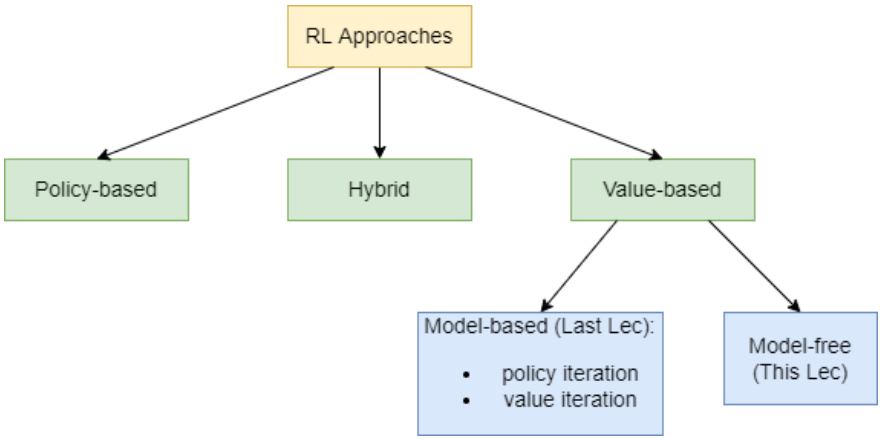
Statistical Techniques in Robotics (16-831, S22) Lecture #16 (Monday, Mar 21)

Model-Free Value Prediction

Lecturer: Kris Kitani *Scribes: Siqi Chai, Jiajie Xu (Group B)*

1 Review

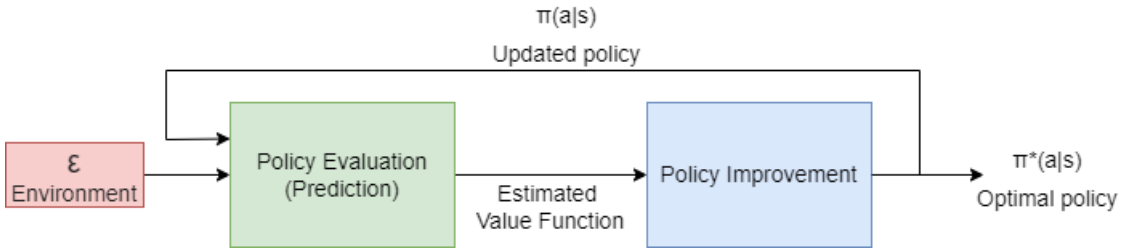
In the previous lecture, we discussed value-based model-based control, also known as model-based reinforcement learning. It is named "value-based" because our agent is explicitly learning value function V to choose the best policy. And it's called "model-based" because we assume that state transition dynamics $p(s'|s,a)$ and reward function $r(s',a,s)$ are known from the environment. We covered two algorithms: policy iteration and value iteration.



The other two types of reinforcement learning approaches:

- Policy-based: learn parameterized policy that select action without consulting value function [1]
- Hybrid: combing value-based and policy-based methods. Current paradigm for research in this area

Value-based control consists of policy evaluation and policy improvement steps



The key to policy evaluation step is Bellman equation:

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s) \sum_{s'} p\left(s' \mid s, a\right) \left[r\left(s', a, s\right) + \gamma V^{\pi}\left(s'\right)\right]$$

$$\begin{aligned} Q^{\pi}(s, a) &= \sum_{s'} p\left(s' \mid s, a\right) \left[r\left(s', a, s\right) + \gamma V^{\pi}\left(s'\right)\right] \\ &= \sum_{s'} p\left(s' \mid s, a\right) \left[r\left(s', a, s\right) + \gamma \sum_{a'} \pi\left(a' \mid s'\right) Q^{\pi}\left(s', a'\right)\right] \end{aligned}$$

Value-based control does not require accurate model prediction to predict optimized policy.

1.1 Policy Iteration

In policy iteration, the agent evaluates V given our current π, loop till V for each state converges (policy evaluation), and then update policy for each state by choosing the one that maximize V (Policy improvement), which causes V to change again. Continue this loop until π converges. V' in this algorithm refers to the old value function.

Algorithm 1 POLICY_ITERATION $(\pi, r(s), p\left(s' \mid s, a\right), \gamma)$

$\pi \leftarrow \text{rand}(\mathcal{A})$
$V \leftarrow \text{rand}(\mathbb{R})$
$V' \leftarrow \text{rand}(\mathbb{R})$
while $\pi' \neq \pi$ **do**
    while $\max_s |V(s) - V'(s)| \geq \epsilon$ **do** ▷ Policy evaluation
        $V' \leftarrow V$
        for $s \in \mathcal{S}$ **do**
            $Q(s, a) = r(s) + \gamma \sum_{s'} p\left(s' \mid s, a\right) V'\left(s'\right) \quad \forall a$
            $V(s) = \sum_a \pi(a \mid s) Q(s, a)$
        end for
    end while

    $\pi'(s) = \arg\max_a Q(s, a) \quad \forall s$ ▷ Policy improvement
end while
return π

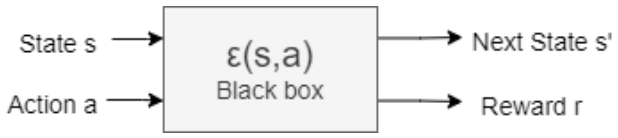
1.2 Value Iteration

Value Iteration share the same block diagram as policy iteration. But the key difference between these two is that Value Iteration does not wait for V to converge for the current π before updating π, i.e. for each state, π is updated right after $V(s)$ is calculated. Value iteration also takes direct advantage of dynamic programming. This makes value iteration converges faster compared with policy iteration.

2 Summary

2.1 Model-free Environment and Prediction

In model-free environment, we do not have access to transition dynamics and reward function from the environment. We can only interact with environment to collect interaction samples, which are in the form of quadruplets $\{s', r, s, a\}_{n=1}^{N}$



We need a new algorithm in the policy evaluation block in 1 to calculate value function. There are two methods, with a debate between which one is better in performance.

- system identification: Estimate $p(s'|s, a)$ and $r(s', a, s)$ functions to fit N datapoints $\{s', r, s, a\}_{n=1}^{N}$, learn the dynamics and convert to model-based prediction problem, then use algorithms from last lecture
- model-free prediction: directly measure/estimate the value function

Next, we will cover several model-free prediction algorithms. The difference between on-policy and off-policy is how you draw samples (using current vs other policy)

Algorithms	Monte Carlo	Temporal Differencing
on-policy	Monte Carlo	N-step TD, TD-lambda
off-policy	Importance Sampling Monte-Carlo	Importance Sampling Temporal Differencing

2.2 Monte Carlo

The Monte Carlo algorithm is the most basic reinforcement learning algorithm. The goal is to learn the value function from the samples. We assume that we do not have access to the transition dynamics $p(s'|s, a)$ and the reward function $r(s', s, a)$. We are given the policy $\pi(a|s)$ and the problem is to find the value function:

$$Q(a, s)$$

The problem itself is like test driving the policy in a environment. The reward function is recovered through interaction with the environment. Recall the state value function we introduced before:

$$V^{\pi}(s) = E_{\pi,P}[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s]$$

The value of a state is the expected return under the MDP (Policy, dynamics, initial state distribution). We can rewrite it with a short-hand notation G representing the cumulative discounted reward over one episode,

$$V^{\pi}(s) = E_{\pi,P}[G|s_0 = s]$$

Note that, we do not have access to the transition dynamics, P. Thus, in a model free setup, we cannot directly compute the value function through this formulation. To solve this problem in this setting, we use Monte Carlo estimate, that is, to interact with the environment many times and use the average return as the reward function. Suppose we have a grid game defined as above. We

-1	+2	-1	+10, Win
-1, s0	-8	+2	-8
-1	-1	-1	-10, Die

start from s_0, which is the first grid on the second row. In this case, if we want to estimate the value of state s_0, we can use Monte Carlo estimate. We play the game for N times, and record the cumulative reward of each trajectory. After N games, we take the average over all the cumulative reward for each round, and use that value as the estimated rewad of s_0:

$$V(s_0) = \frac{1}{N} \sum_{i=1}^{N} G(s_0)$$

2.2.1 First Visit MC Prediction

The above procedure we seen is called the First Visit MC Prediction algorithm. On line 5 of the algorithm, in each episode, we drawn a trajectory following the distribution specified by environment ε and policy π.Given a trajectory in each episode, we iteratively compute the award of each state along the trajectory. Note on line 7, we prevent updating a pre-calculated state award by enforcing the update to only happen on first visit to a state. If it is a first visit to a state (line 8, 9), we calculate the cumulative reward of this state by summing up all the upcoming reward along the trajectory after this state. We also add the counter N for this state by 1. Eventually, we return the average rewards as the Monte Carlo estimate of the reward function for policy π

Algorithm 2 FirstVisit-MC-Prediction

1: π: a policy
2: ε: The environment
3: **procedure** FIRSTVISIT-MC-PREDICTION(π, ε)
4: **for** $e = 1, \cdots, E$ **do**
5: $\{s^{(t)}, a^{(t)}, r^{(t)}\} \sim \varepsilon|\pi$
6: **for** $t = 0, \cdots, T$ **do**
7: **if** First Visit to $s^{(t)}$ in episode e **then**
8: $G(s^{(t)}) \leftarrow G(s^{(t)}) + \sum_{i=t}^{T} r^{(i)}$
9: $N(s^{(t)}) \leftarrow N(s^{(t)}) + 1$
10: **end if**
11: **end for**
12: **end for**
13: **return** $V(s) \leftarrow G(s)/N(s) \quad \forall s$
14: **end procedure**

2.2.2 First Visit Incremental MC Prediction

A variant of the above algorithm is the First Visit Incremental MC Prediction. The only difference to the previous algorithm is that here we incrementally update the value function, instead of averaging it by the very end. This is also called the 'offline' update. Note that on line 10, the update here is actually taking the weighted average. That is:

$$\begin{aligned} V(s) &= \frac{N(s)-1}{N(s)}V(s) + \frac{1}{N(s)}G_e(s) \\ &= V(s) - \frac{1}{N(s)}V(s) + \frac{1}{N(s)}G_e(s) \\ &= V(s) + \frac{1}{N(s)}(G_e(s) - V(s)) \end{aligned}$$

The reason we do this weighted average update is because it can generalize to a changing environment. If the environment changes during the algorithm roll out, we may want to forget older episode and adapt to newer ones. We can do so by weighting new updates with a α factor, that is:

$$V(s) = V(s) + \alpha(G_e(s) - V(s))$$

Algorithm 3 FirstVisit-Incremental-MC-Prediction

1: π: a policy
2: ε: The environment
3: **procedure** FirstVisit-Incremental-MC-Prediction(π, ε)
4: **for** $e = 1, \cdots, E$ **do**
5: $\{s^{(t)}, a^{(t)}, r^{(t)}\} \sim \varepsilon|\pi$
6: **for** $t = 0, \cdots, T$ **do**
7: **if** First Visit to $s^{(t)}$ in episode e **then**
8: $G(s^{(t)}) \leftarrow G(s^{(t)}) + \sum_{i=t}^{T} r^{(i)}$
9: $N(s^{(t)}) \leftarrow N(s^{(t)}) + 1$
10: $V(s) \leftarrow V(s) + \frac{1}{N(s^{(t)})}(G_e(s^{(t)}) - V(s^{(t)}))$
11: **end if**
12: **end for**
13: **end for**
14: **return** $V(s)$
15: **end procedure**

2.2.3 Every Visit MC Prediction

Based on the previous algorithm, if we further remove the constraint that only update the reward on first visit, we have Every Visit MC Prediction. This is the general form of Monte Carlo estimation. However, looking at line 7, we can see that this update is causing the algorithm slow. This step makes the time complexity of each episode of $O(T^2)$. We can avoid this by caching the rewards. If we change line 6 to compute backward, that is $t = T, \cdots, 0$, we can speed things up.

Can we use this algorithm for a infinite horizon problem? The answer is no due to the fact on line 8. We need episodic returns to complete the estimate.

Can we replace the pre-sampling of trajectories on line 4 with a online updated calculation? The answer is yes. We can use the TD-algorithm introduced later.

Algorithm 4 EveryVisit-MC-Prediction

1: π: a policy
2: ε: The environment
3: **procedure** EveryVisit-MC-Prediction(π, ε)
4: **for** $e = 1, \cdots, E$ **do**
5: $\{s^{(t)}, a^{(t)}, r^{(t)}\} \sim \varepsilon|\pi$
6: **for** $t = 0, \cdots, T$ **do**
7: $G^{(t)} \leftarrow \sum_{i=t}^{T} r^{(i)}$
8: $V(s^{(t)}) \leftarrow V(s^{(t)}) + \alpha(G^{(t)} - V(s^{(t)}))$
9: **end for**
10: **end for**
11: **return** $V(s)$
12: **end procedure**

2.2.4 Properties of Monte Carlo prediction

- Model free approach: we do not need the value function, we estimate it.
- Monte Carlo is known for high variance for long trajectory estimation. This happens because at each state along a trajectory, there will be some randomness. Long trajectory can accumulate these randomness, causing high variances.
- works only for finite horizon problem.
- Uses full return to estimate the value function, as opposed to bootstrapping.
- Yields unbiased estimate, due to above property (zero bias in mean)
- Converges to the MSE solution

2.3 N-step TD

2.3.1 Temporal Difference Prediction

In Temporal Difference Prediction algorithm, we introduce the idea of bootstrapping (from dynamic programming) to incrementally estimate values after every action, based on another estimate.

Recall the value function update equation from Monte Carlo:

$$V(s)^{(t)} \leftarrow V(s^{(t)}) + \alpha(G^{(t)} - V(s^{(t)}))$$

The empirical reward $G^{(t)}$ can be estimated as the sum of immediate reward $r^{(t)}$ and discounted estimated value for the next time step $V(s^{(t+1)})$:

$$G^{(t)} = r^{(t)} + \gamma V(s^{(t+1)})$$

This sum is called **1-step TD target**, and $r^{(t)} + \gamma V(s^{(t+1)}) - V(s^{(t)})$ is called **TD error**. The 1-step TD update function can be written as:

$$V(s^{(t)}) \leftarrow V(s^{(t)}) + \alpha(r^{(t)} + \gamma V(s^{(t+1)}) - V(s^{(t)}))$$

If we incrementally substitute value function with the sum of immediate reward and estimated value, we get the total reward for N-step TD:

$$G^{(t)}(N) = r^{(t)} + \gamma r^{(t+1)} + \cdots + \gamma^{(N-1)} r^{(N-1)} + \gamma^N V(s^{(t+N)})$$

Notice that if we have $N = T - t$, meaning that we sum all reward functions till time step T and don't do any estimation, the total reward function $G^{(t)}(\infty)$ becomes the same as Monte Carlo algorithm.

Let's go back to the grid game and use it as an 1-step TD example. But this time at each state, we have a "memo" to tell us the estimated value function of the state (marked blue). This estimate needs to be initialized before time step 0, and will be updated for each action. In the example, $\gamma = 1$, $\alpha = 0.9$

-1 (10)	+2 (11)	-1 (9)	+10, Win (10)
-1 (8) (8.9), s0	-8	+2 (9)	-8
-1 (7)	-1 (8)	-1 (9)	-10, Die

At s0, suppose the policy decide to go up, then using one step TD estimate:

$$G(1) = r + \gamma \cdot V(s') = -1 + 1 \cdot 10 = 9$$

Update value estimate at state 0 (marked red):

$$V(s0) \leftarrow V(s0) + \alpha(G(1) - V(s0))$$

$$V(s0) = 8 + 0.9 \cdot (9 - 8) = 8.9$$

The pseudocode for the algorithm above:

Algorithm 5 1-STEP-TD-Prediction (π, α, γ)

for $e = 1, \ldots, E$ **do**
  $\left\{s^{(t)}, a^{(t)}, r^{(t)}\right\}_{t=0}^{T} \sim \mathcal{E} \mid \pi$
  for $t = 0, \ldots, T-1$ **do**
    $G^{(t)}(1) \leftarrow r^{(t)} + \gamma V(s^{(t+1)})$
    $V(s^{(t)}) \leftarrow V(s^{(t)}) + \alpha \left[G^{(t)}(1) - V(s^{(t)})\right]$
  end for
end for
return $V(s)$

In the algorithm:

- E is the total episode number, we want it to be a sufficiently large number, but there is no hard convergence condition
- For the sampling process in line 2, we don't need to sample the whole trajectory at each time step t, due to the estimation of $V(s^{(t+1)})$

Substitute line 4 of the above algorithm from TD 1-step to TD N-step, we get N-step TD prediction algorithm:

Algorithm 6 N-STEP-TD-Prediction (π, α, γ, N)

1: **for** $e = 1, \ldots, E$ **do**
2: $\left\{s^{(t)}, a^{(t)}, r^{(t)}\right\}_{t=0}^{T} \sim \mathcal{E} \mid \pi$
3: **for** $t = 0, \ldots, T-1$ **do**
4: $G^{(t)}(N) \leftarrow \sum_{i=t}^{t+N-1} \gamma^{i-t} r^{(i)} + \gamma^N V(s^{(t+N)})$
5: $V(s^{(t)}) \leftarrow V(s^{(t)}) + \alpha \left[G^{(t)}(1) - V(s^{(t)})\right]$
6: **end for**
7: **end for**
8: return $V(s)$

2.3.2 Properties of N-step TD Prediction

1. TD prediction is model-free
2. Uses bootstrapping to estimate value function
3. Compared with Monte Carlo:
   - Do not need reward for the full episode at time step t, only the next N steps
   - For the above reason, Monte Carlo is called an offline/batch algorithm, while N-step TD is called online/incremental algorithm
   - For the same reason, N-step TD can be used for non-terminating problem (infinite horizon)
   - N-step TD is more efficient, but the estimation is biased
   - TD prediction has low variance due to short sequence length
4. Need to select the optimal N depending on the problem (hyperparameter)

References

[1] A. Barto and R. S. Sutton. *Reinforcement Learning: An Introduction.*