Statistical Techniques in Robotics (16-831, S22) Lecture #16 (Monday, March 21, 2022)

Model-Free Value Prediction

Lecturer: Kris Kitani *Scribe: Michelle Zhao*

1 Review

1.1 Markov Decision Process

A Markov Decision Process (MDP) defines a fully observable environment where outcomes are partly random and partly determined by the control of a decision-making agent. $\mathcal{S}$ is the set of states. $\mathbb{A}$ defined the set of actions available to the agent. The transition function $p(s'|s,a)$ determines how the state changes based on a action taken by the agent. $r : \mathcal{S} \rightarrow \mathbb{R}$ is the reward function. The agent's policy is denoted $\pi : \mathcal{S} \rightarrow \mathbb{A}$.

1.2 Reinforcement Learning Objective

The objective in reinforcement learning is to train a policy, or decision-making algorithm, to learn optimal behavior in some environment. In Reinforcement Learning (RL), algorithms interact with an environment, E, to learn a policy, π, which determines a state to action mapping of learned behavior. In each interaction with the environment, the RL agent observes a state of the environment s. The agent policy π determines an action to take a. Upon taking action a, the environment returns a reward r to the agent, and transitions to the next state s', which serves as the next input into the agent policy. The agent's aim is to maximize its total reward received.

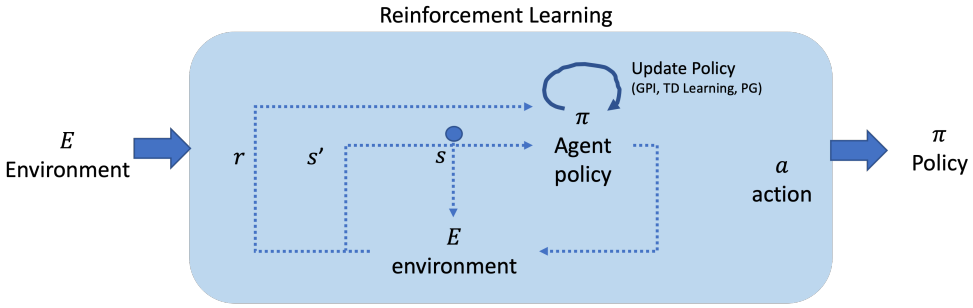

Figure 1: Reinforcement learning loop

1.3 Reinforcement Learning Approaches

RL approaches can be broken down into three categories of methods: (1) Policy-based, (2) Value-based, and (3) Hybrid.

- Policy-based: In Policy-based methods, the agent maintains an explicit policy representation

that maps state to action $\pi : s \rightarrow a$. Examples of policy representations include neural networks.

- Value-based: In Value-based RL, the agent does not maintain an explicit policy function. Instead, it stores a state value function, representing expected reward in different states, or a state-action value function, representing expected reward for different actions in different states. The agent then acts according to the action with the highest state-action value. An example of a value-based approach is value iteration.
- Hybrid: Hybrid approaches incorporate both policy-based and value-based approaches, modeling both the value function and policy function. An example of hybrid approaches are actor-critic methods.

1.4 Value Based Control

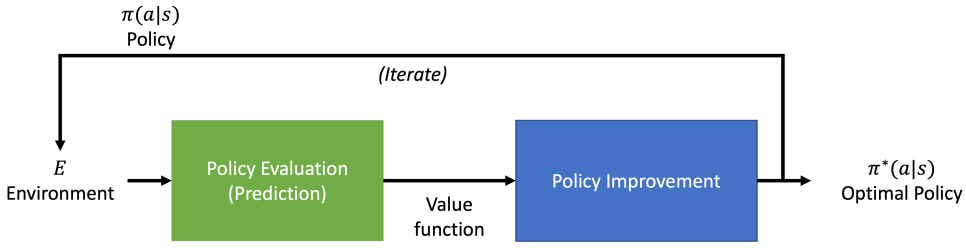

Figure 2: Value-based Control

In value-based control, an explicit value function is used to select actions. The last lecture introduced Policy Iteration and Value Iteration, two value-based control algorithms, defined below. In Policy Iteration, the first step is policy evaluation, otherwise known as the prediction step. In the policy evaluation step: the agent takes in some state of the environment and outputs a value function: the Q-values for each state-action. What makes this value-based control is that the agent maintains an explicit estimation of the value function. In the second step, the agent improves the policy, by taking an arg-max over actions under the new Q-values, $\pi(a|s) \leftarrow \arg\max_a Q(s,a) \forall s$.

Algorithm 1 Policy Iteration

$V \leftarrow rand(\mathbb{R})$
$V' \leftarrow rand(\mathbb{R})$
while $\max_a |V(s) - V'(s)| \geq \epsilon$ **do**
  $V' \rightarrow V$
  for $s \in S$ **do**
    $Q(s,a) = r(s) + \gamma \sum_{s'} p(s'|s,a)V'(s')$
    $V(s) = \sum_a \pi(a|s)Q(s,a)$
  end for
end while
for $s \in S$ **do**
  $\pi(s) = \arg\max_a Q(s,a)$
end for

In Value Iteration, the policy and value function are improved in the same loop. The algorithm does not wait for V to converge for a given policy π before updating the policy.

Algorithm 2 Value Iteration

$\pi \leftarrow rand(\mathbb{A})$
$V \leftarrow rand(\mathbb{R})$
$V' \leftarrow rand(\mathbb{R})$
while $\max_a |V(s) - V'(s)| \geq \epsilon$ **do**
  $V' \rightarrow V$
  for $s \in S$ **do**
    $Q(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s')$
    $\pi(s) = \arg\max_a Q(s,a)$
    $V(s) = Q(s, \pi(s))$
  end for
end while

2 Model-Based vs Model-Free RL

There are two ways the agent can receive the environment, which yields two different types of value prediction algorithms. In Model-based RL, the agent has access to a fully-specified model of the environment. This means that agent knows how states transition to other states based on the actions taken. The agent also has access to how rewards are provided in each state based on the actions taken. One example of this is a gridworld environment, where the reward-providing goals, are visible, and the agent knows that actions moving in any direction will take them to the corresponding adjacent cell.

In the Model-free scenario, the agent is limited with interactions with the environment. The agent does not have direct access to the transition dynamics or reward function, and instead must learn behavior through repeated interactions with the environment.

2.1 Model-Based Prediction

In Model-based prediction, the agent uses a model of the environment to compute the value function. In some cases in model-based prediction, the agent will have access to and use a fully specified joint state-reward transition model, $p(s', r|s, a)$. The agent takes an action and the environment hands the agent both the next state and a reward, which are determined from a joint distribution. Using this specified model, we can derive the state transition dynamics $p(s'|s, a)$ and reward function, $r(s, a)$. The state transition dynamics are computed by marginalizing out the reward $p(s'|s, a) = \sum_r p(s', r|s, a)$. The reward function can then be computed by utilizing the state transition dynamics. The p in the expectation refers to the joint state-reward distribution $p(s', r|s, a)$.

$$\mathbb{E}_p[r^t|s^{t+1} = s', a^t = a, s^t = s] = \sum_r r \times \frac{p(s', r|s, a)}{p(s'|s, a)} = r(s', a, s) \tag{1}$$

$$\mathbb{E}_p[r^t|a^t = a, s^t = s] = \sum_{r,s'} r \times p(s', r|s, a) = r(s, a) \tag{2}$$

If the state-reward transition model is not a joint model, an agent might have access to or estimate a state transition model $p(s'|s,a)$ and reward function $r(s',s,a)$.

In some model-based RL problems, the agent will estimate the model of the environment to learn behavior.

2.2 Model-Free Prediction

In model-free RL, the agent does not use a model of the environment to determine its behavior. We can assume the agent only has indirect access to the transition and reward dynamics by interacting with the environment. We use interactions with the environment to compute the value function. Repeated interactions with the environment yield multiple samples

$$\{s', r, s, a\}_{n=1}^{N}$$

where s represents the current state, a represents the action taken by the agent, r represents the reward received from the environment upon taking action a, and s' represents the next state.

There are 2 ways in which we can compute the value function using sampled interaction with the environment:

1. System Identification: The system identification approach makes an approximation of the state transition dynamics and reward function based on repeated interactions. Once the algorithm has estimated the dynamics, it can convert the problem to a model-based prediction problem. System identification approaches a model-free environment with a model-based prediction approach.

2. Model-free prediction: The approach directly estimates the value function based on interactions with the environment. This lecture will focus on Monte-Carlo and N-Step TD methods for model-free prediction.

Algorithms	Monte Carlo	Temporal Differencing
On-Policy *The Q(s,a) function is learned by taking actions using the current policy π.*	• Monte Carlo	• N-Step TD • TD-Lambda
Off-Policy *The Q(s,a) function is learned by from actions not based directly on the current policy π.*	• Importance Sampling Monte-Carlo	• Importance Sampling Temporal Differencing

Figure 3: Model-free prediction methods covered. For context, Deep Q-Learning is an example of a temporal differencing method, and AlphaGo, which uses Monte-Carlo tree-search, is an example of a Monte-Carlo method.

3 Monte-Carlo Prediction

The value of a state is the expected (average) return under the MDP, given the agent policy, state transition dynamics, and initial state distribution.

$$V^{\pi}(s) = \mathbb{E}_{\pi,P}[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s] \tag{3}$$

$$V^{\pi}(s) = \mathbb{E}_{\pi,P}[G | s_0 = s] \tag{4}$$

where the return $G = \sum_{t=0}^{T} \gamma^t r_t$ is the cumulative discounted reward over a single episode.

In a model-free scenario, we cannot compute this expectation because we don't have access to the transition dynamics model. Therefore, we must compute the return by interacting with the environment many times and use the average return as a Monte-Carlo estimate of the value function.

$$V(s) = \frac{1}{N} \sum_{i=1}^{N} G_i(s) \tag{5}$$

where N is the number of interaction samples, and $G_i(s)$ is the return of state s in interaction i.

We will cover 4 Monte-Carlo prediction algorithms:

1. First-Visit MC Prediction
2. First-Visit Incremental MC Prediction
3. First-Visit Dynamic MC Prediction
4. Every-Visit MC Prediction

3.1 First-Visit MC Prediction

Algorithm 3 First-Visit MC Prediction (π)

$G \leftarrow \mathbf{0}$
for $e = 1, ..., E$ **do**
  $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \varepsilon | \pi$
  for $t = 0, ..., T$ **do**
    if first visit to state $s^{(t)}$ in episode e **then**
      $G(s^{(t)}) \leftarrow G(s^{(t)}) + \sum_{i=t}^{T} r^{(i)}$
      $N(s^{(t)}) \leftarrow N(s^{(t)}) + 1$
    end if
  end for
end for
Return $V(s) \leftarrow \frac{G(s)}{N(s)}$

E represents the number of episodes. For each episode, given some policy π, the agent can roll-out a trajectory of samples $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \varepsilon|\pi$ by acting according to policy π for T timesteps. Next, we compute the value function by looping over states visited in the interaction sequence. For each state s^t in episode e, if it is the first visit to state s^t in the sequence, then the future reward is added to the returns of this state. The counter for visits to this state $N(s)$ is incremented. The policy stays the same for the entire time. The final value function for each state is computed at the end of the algorithm.

3.2 First-Visit Incremental MC Prediction

The incremental version of the First-Visit MC Prediction algorithm helps the algorithm generalize to the more common value function update. At every timestep in each episode, if the state s^t in episode e is the first visit to state s^t in the sequence, the return $G(s^{(t)})$ is updated, as well as the value function $V(s)$.

Algorithm 4 First-Visit Incremental MC Prediction (π)

$V \leftarrow \mathbf{0}$
for $e = 1, ..., E$ **do**
  $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \varepsilon|\pi$
  for $t = 0, ..., T$ **do**
    if first visit to state $s^{(t)}$ in episode e **then**
      $G(s^{(t)}) \leftarrow \sum_{i=t}^{T} r^{(i)}$
      $N(s^{(t)}) \leftarrow N(s^{(t)}) + 1$
      $V(s) \leftarrow V(s) + \frac{1}{N(s^{(t)})}(G(s^{(t)}) - V(s^{(t)}))$
    end if
  end for
end for
Return $V(s)$

The incremental updates are a weighted average of the return up until the current point and the return the agent has just received.

$$V(s) = \frac{N(s) - 1}{N(s)} V(s) + \frac{1}{N(s)} G_e(s) \tag{6}$$

$$V(s) = V(s) - \frac{1}{N(s)} V(s) + \frac{1}{N(s)} G_e(s) \tag{7}$$

$$V(s) = V(s) - \frac{1}{N(s)} (G_e(s) - V(s)) \tag{8}$$

3.3 First-Visit Dynamic MC Prediction

The incremental updates are advantageous in that it provides intermediate estimates of the value function, rather than needing to wait for the algorithm to complete to obtain the value function.

If the environment is stationary, $1/N(s)$ in an appropriate weighting factor for the value function update. If the environment model is not stationary, it may be advantageous to forget old episodes using a constant factor. An α parameter can be used to control how much we should prioritize recent episodes. Higher values of α give more weight to the most recent episodes. The algorithm does not need to keep track of visits to each state.

Algorithm 5 First-Visit Dynamic MC Prediction (π, α)

1: $V \leftarrow \mathbf{0}$
2: **for** $e = 1, ..., E$ **do**
3: $\quad \{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \varepsilon|\pi$
4: $\quad$ **for** $t = 0, ..., T$ **do**
5: $\quad\quad$ **if** first visit to state $s^{(t)}$ in episode e **then**
6: $\quad\quad\quad G^{(t)} \leftarrow \sum_{i=t}^{T} r^{(i)}$
7: $\quad\quad\quad V(s) \leftarrow V(s) + \alpha(G^{(t)} - V(s^{(t)}))$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: **end for**
11: Return $V(s)$

3.4 Every-Visit MC Prediction

In Every-Visit MC Prediction, instead of making updates to the returns and value function when the state is in its first visit in the episode, we update the value function on every visit to each state. Similar to First-Visit Dynamic MC Prediction, the algorithm does not need to keep track of visits to each state.

Algorithm 6 Every-Visit MC Prediction (π, α)

1: $V \leftarrow \mathbf{0}$
2: **for** $e = 1, ..., E$ **do**
3: $\quad \{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \varepsilon|\pi$
4: $\quad$ **for** $t = 0, ..., T$ **do**
5: $\quad\quad G^{(t)} \leftarrow \sum_{i=t}^{T} r^{(i)}$
6: $\quad\quad V(s^{(t)}) \leftarrow V(s^{(t)}) + \alpha(G^{(t)} - V(s^{(t)}))$
7: $\quad$ **end for**
8: **end for**
9: Return $V(s)$

Notes on Every-Visit MC Prediction

- Line 5 is inefficient. The algorithm can be optimized for efficiency by iterating backwards from the end of the episode and cache returns.
- Monte Carlo (MC) cannot be used for non-terminating problems (infinite horizon), because MC relies on knowing all returns for an entire episode, thus requiring a terminating episode.
- You can compute the value function after each action is sampled by the policy by estimating returns using an intermediate value function estimate. This is temporal difference learning.

3.5 Properties of MC Prediction

- MC prediction is model-free.
- Monte Carlo estimates of return have high variance. When trajectories are short, there isn't as much variance. However, each step incorporates randomness, or stochasticity, which compounds as trajectories get longer, leading to extremely high variance. Thus, MC estimates have high variance due to the compounding factors of long sequences.
- MC prediction works for finite horizon problems.
- MC prediction uses full returns (does not use bootstrapping) to estimate the value function, which yields an unbiased (zero-bias) estimate.
- MC prediction converges to the MSE (mean-squared error) solution.

4 Temporal Difference (TD) Prediction

Temporal Difference (TD) Prediction allows us a way to extend MC prediction for online updates to the value function based on actions taken by the agent policy. It also allows us to extend MC prediction to the continuous learning case (single long episode). Temporal difference prediction learns from raw experience like MC prediction but also introduces the idea of bootstrapping (from dynamic programming) to incrementally update the values after each action.

MC prediction requires the entire episode to be complete in order to update the value function, and thus can't handle continuous problems. In TD learning, the empirical return can be decomposed into immediate rewards and value function updates.

The Monte-Carlo update equation is

$$V(s) \leftarrow V(s) + \alpha(G^{(t)} - V(s^{(t)})) \tag{9}$$

The empirical return $G^{(t)}$ can be expanded into

$$G^{(t)}(1) = r^{(t)} + \gamma V(s^{(t+1)}) \tag{10}$$

where $r^{(t)}$ is the immediate reward and $\gamma V(s^{(t+1)})$ is the estimated value. $r^{(t)}+\gamma V(s^{(t+1)})$ represents the 1-step TD target.

The 1-step TD update is:

$$V(s) \leftarrow V(s) + \alpha(r^{(t)} + \gamma V(s^{(t+1)}) - V(s^{(t)})) \tag{11}$$

$(r^{(t)} + \gamma V(s^{(t+1)}) - V(s^{(t)}))$ represents the 1-step TD error.

The empirical return $G^{(t)}$ can be expanded further into the 2-step TD target:

$$G^{(t)}(2) = r^{(t)} + \gamma r^{(t+1)} + \gamma^2 V(s^{(t+2)}) \tag{12}$$

The 2-step TD update is:

$$V(s) \leftarrow V(s) + \alpha(r^{(t)} + \gamma r^{(t+1)} + \gamma^2 V(s^{(t+2)}) - V(s^{(t)})) \tag{13}$$

$(r^{(t)} + \gamma r^{(t+1)} + \gamma^2 V(s^{(t+2)}) - V(s^{(t)}))$ represents the 2-step TD error.

The N-step TD update is:

$$G^{(t)}(N) = r^{(t)} + \gamma r^{(t+1)} + ...\gamma^{N-1} r^{(N-1)} + ... + \gamma^N V(s^{(t+N)}) \tag{14}$$

$$V(s) \leftarrow V(s) + \alpha(r^{(t)} + \gamma r^{(t+1)} + ...\gamma^{N-1} r^{(N-1)} + ... + \gamma^N V(s^{(t+N)}) - V(s^{(t)})) \tag{15}$$

The infinite-step TD update converges to the Monte-Carlo update:

$$G^{(t)}(\infty) = r^{(t)} + \gamma r^{(t+1)} + ...\gamma^{N-1} r^{(N-1)} + ... + \gamma^T r^{(T)} \tag{16}$$

4.1 1-Step TD Prediction

The key idea of TD prediction is to leave a memo of the value at each state and update the estimate of the state memo value every time the state is visited. The one-step TD target does not require access to future rewards.

Algorithm 7 1-Step TD Prediction (π, α)

$V \leftarrow \mathbf{0}$
for $e = 1, ..., E$ **do**
  $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \varepsilon|\pi$
  for $t = 0, ..., T-1$ **do**
    $G^{(t)}(1) \leftarrow r^{(t)} + \gamma V(s^{(t+1)})$
    $V(s^{(t)}) \leftarrow V(s^{(t)}) + \alpha(G^{(t)}(1) - V(s^{(t)}))$
  end for
end for
Return $V(s)$

4.2 N-Step TD Prediction

In general, you can use any N-steps estimate of the reward. In N-step TD, instead of just a 1-step reward, we use an N-step reward. We trust the value function estimate to capture all of the rewards beyond the N-steps. Similar to 1-step TD, the N-step TD target does not require access to future rewards.

Algorithm 8 N-Step TD Prediction (π, α, N)

$V \leftarrow \mathbf{0}$
for $e = 1, ..., E$ **do**
  $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \varepsilon|\pi$
  for $t = 0, ..., T-N$ **do**
    $G^{(t)}(N) \leftarrow \sum_{i=t}^{t+N-1} \gamma^{i-t} r^{(i)} + \gamma^N V(s^{(t+N)})$
    $V(s^{(t)}) \leftarrow V(s^{(t)}) + \alpha(G^{(t)}(N) - V(s^{(t)}))$
  end for
end for
Return $V(s)$

4.3 Properties of TD Prediction

- The appropriate value for N depends on the problem.
- If N is large, the algorithm can be optimized for efficiency by iterating backwards from the end of the episode and caching returns.
- TD prediction can be used for non-terminating problems (infinite horizon) because the value function is updated after each action, using bootstrapping.
- TD prediction is model-free.
- TD prediction uses bootstrapping (guess of a guess) to estimate the value function. Thus, the estimates are biased.
- TD value estimates are low variance, due to the sequences being short in length.
- Bias-variance tradeoff. TD uses bootstrapping to compute estimates of the value function at each state visited. The TD estimates are not over entire episodes which decreases variance in estimates, but estimate is biased because we use an estimated value function for estimating the state values. Hence, TD trades off better variance for bias in this approach.
- TD prediction converges to the maximum likelihood solution. The expected reward is weighted by the probability of the state transition.

5 Summary

In this lecture, we covered Model-Free Value-based Reinforcement Learning. We covered Monte-Carlo and Temporal Differencing prediction. In model-free RL, the agent does not use a model of the environment dynamics (state transition dynamics and reward function) to learn a policy. The agent can use interactions with the environment to compute Monte-Carlo estimates of the value function. There are multiple variants of Monte-Carlo prediction. Temporal Difference (TD) Prediction extends MC prediction for online updates to the value function based on actions taken by the agent policy, allowing for the algorithm to handle infinite horizon problems.

References

[1] Leslie Kaebling. Value Iteration Notes, 1996. https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node19.html

[2] "Artificial Intelligence." Artificial Intelligence - Foundations of Computational Agents – 9.5.3 Value Iteration, https://artint.info/html/ArtInt_227.html#:~:text=Value%20iteration%20is%20a%20method,uses%20an%20arbitrary%20end%20point.