Tabular RL for Value Prediction

Reading: Algs for RL (Szepesvári), Sec 3.1

$\pi_{k-1} \to Q^{\pi_{k-1}} \to \pi_k := \text{greedy}(Q^{\pi_{k-1}})$.

The Value Prediction Problem

$J(\pi) = \mathbb{E}_\pi\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \,\middle|\, s_1 \sim d_{init}\right]$

$= \mathbb{E}_{s \sim d_{init}}[V^\pi(s)]$.

- Given $\boxed{\pi}$, want to learn V^π or Q^π
- Why useful? Recall that if we know how to compute Q^π, we can run policy iteration
- On-policy learning: data is generated by π
- Off-policy learning: data is generated by some other policy
- Will mostly focus on on-policy learning for now; all actions in data are taken according to $\boxed{\pi}$ (often omitted)
- When action is always chosen by a fixed policy, the MDP reduces to a Markov chain plus a reward function over states, also known as Markov Reward Processes (MRP)

Monte-Carlo Value Prediction

$V^{\pi}(s)$

- If we can roll out trajectories from any starting state that we want, here is a simple procedure
- For each s, roll out n trajectories using policy π
  - For episodic tasks, roll out until termination
  - For continuing tasks, roll out to a length (typically $H = O(1/(1-\gamma))$) such that omitting the future rewards has minimal impact ("small truncation error")
  - Let $\hat{V}^{\pi}(s)$ (will just write $V(s)$) be the average discounted return
- also works if we can draw starting state from an exploratory initial distribution (i.e., one that assigns non-zero probability to every state)
  - Keep generating trajectories until we have enough data points for each starting state

Implementing MC in an online manner

- The previous procedure assumes that we collect all the data, store them, and then process them (batch-mode learning)
- Can we process each data point as they come, without ever needing to store them? (online, one-pass algorithm)
- For $i = 1, 2, \ldots$
  - Draw a starting state $\boxed{s_i}$ from the exploratory initial distribution, roll out a trajectory using π from s_i, and let G_i be the (random) discounted return $= \sum_{t=1}^{H} \gamma^{t-1} r_t^{(i)}$
  - Let $n(s_i)$ be the number of times s_i has appeared as an initial state. If $n(s_i) = 1$ (first time seeing this state), let $V(s_i) \leftarrow G_i$ $\approx V^{\pi}(s_i)$.
  - Otherwise, $\boxed{V(s_i) \leftarrow \frac{n(s_i) - 1}{n(s_i)} V(s_i) + \frac{1}{n(s_i)} G_i}$
  - Verify: at any point, $V(s)$ is always the MC estimation using trajectories starting from s available so far

$$x_1, x_2, \cdots, x_n, x_{n+1}, \cdots \quad \Big| \quad \frac{1}{n}\sum_{i=1}^{n} x_i .$$

$$\bar{x} = ?$$

$$n = 0$$

$$n \leftarrow n+1 .$$

$$\bar{x} \leftarrow \frac{n-1}{n}\bar{x} + \frac{1}{n} x_n ..$$

$$\bar{x} = \frac{1}{n-1}\sum_{i=1}^{n-1} x_i .$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i . = \underbrace{\arg\min_{\theta} \frac{1}{n}\sum_{i=1}^{n} (\theta - x_i)^2}_{L(\theta)} .$$

$$\theta \leftarrow \theta - \alpha \nabla \theta .$$

$$\theta \leftarrow \theta - \alpha \cdot \underbrace{\nabla_\theta (\theta - x_i)^2}$$

$$\theta - \alpha \cdot (\theta - x_i)$$

$$\downarrow \qquad\qquad \downarrow \qquad \downarrow$$

$$V(s_i) \qquad V(s_i) \quad G_i .$$

Implementing MC in an online manner

$\alpha \propto \frac{1}{n}$.

- More generally, $V(s_i) \leftarrow (1 - \alpha)V(s_i) + \alpha G_i$
  - α is known as the step size or the learning rate
  - in theory, convergence require sum of α goes to infinity while sum of α^2 stays finite; in practice, constant small α is often used
  - G_i is often called "the target"
  - The expected value of the target is what we want to update our estimate to, but since it's noisy, we only move slightly to it
- Alternative expression: $V(s_i) \leftarrow V(s_i) + \alpha(G_i - V(s_i))$
  - Moving the estimate in the direction of error (= target - current)
- Can be interpreted as stochastic gradient descent
  - If we have i.i.d. real random variables $v_1, v_2, \ldots, v_n$, the average is the solution of the least-square optimization problem: $\min_v \frac{1}{2n} \sum_{i=1}^{n} (v - v_i)^2$
  - Stochastic gradient: $v - v_i$ (for uniformly random i)

Every-visit Monte-Carlo

- Suppose we have a continuing task. What if we cannot set the starting state arbitrarily?
- Let's say we only have one single long trajectory
  $s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3, s_4, \ldots$
  - (By "long trajectory", we mean trajectory length >> effective horizon $H = O(1/(1-\gamma))$)
- On-policy: $a_t \sim \pi(s_t)$, where π is the policy we want to evaluate
- Algorithm: for each s, find all t such that $s_t = s$, calculate the discounted sum of rewards between time step t and t+H, and take average over them as $V(s_i)$
- Convergence requires additional assumption: the Markov chain induced by π is ergodic—implying that all states will be hit infinitely often if the trajectory length grows to infinity

Every-visit Monte-Carlo

- You can use this idea to improve the algorithm when we can choose the starting state & the MDP is episodic
- i.e., obtain a random return for each state visited on the trajectory
- What if a state occurs multiple times on a trajectory?
  - Approach 1: only the 1st occurrence is used ("first-visit MC")
  - Approach 2: all of them are used ("every-visit MC")

$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$.

$V(s_t) \leftarrow \mathbb{E}[G_t \mid s_t]$.

$= (T^{\pi} V)(s_t)$

Alternative Approach: TD(0)

- Again, suppose we have a single long trajectory $s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3, s_4, \ldots$ in a continuing task

G_t

- TD(0): for t =1, 2, …, $V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$
  - TD = temporal difference
  - $r_t + \gamma V(s_{t+1}) - V(s_t)$: "TD-error"
  - The same structure as the MC update rule, except that we are using a different target here: $r_t + \gamma V(s_{t+1})$
  - Often called "bootstrapped" target: the target value depends on our current estimated value function V
  - Conditioned on s_t, what is the expected value of the target (taking expectation over the randomness of r_t, s_{t+1})?
    - It's $(T^{\pi}V)(s_t)$

$(T^{\pi}V)(s_t) = \mathbb{E}[r_t + \gamma V(s_{t+1}) \mid s_t]$

$$TD(0): \quad V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

$$\text{w/} \quad G_t := r_t + \gamma V(s_{t+1}).$$

$$V^\pi(s) = (T^\pi V^\pi)(s) = R(s,\pi) + \gamma \mathbb{E}_{s' \sim P(s,\pi)}[V^\pi(s')].$$

$$= \mathbb{E}_{r,s' | s,\pi}[r + \gamma V^\pi(s')].$$

$$V_0 \in \mathbb{R}^S. \qquad V_{k+1} \leftarrow T^\pi V_k$$

$$V_{k+1}(s) = \mathbb{E}_{r,s'|s,\pi}[r + \gamma V_k(s')].$$

$$s, \left\{a_i \sim \pi(\cdot|s),\ r_i = R(s,a_i),\ s_i' \sim P(\cdot|s,a_i)\right\}_{i=1}^{n}$$

$$\approx \frac{1}{n}\sum_{i=1}^{n}\left(r_i + \gamma V_k(s_i')\right).$$

for $s \in S$.

for $i = 1, 2, \ldots, n$

$$V_{k+1}(s) \leftarrow V_{k+1}(s) + \alpha \cdot \left(r_i + \gamma V_k(s_i') - V_{k+1}(s)\right).$$

$(s, a \sim \pi, r, s')$

for (s, a, r, s').

$| \; V_{k+1}(s) \leftarrow V_{k+1}(s) + \alpha(r + \gamma V_k(s') - V_{k+1}(s))$.

Understanding TD(0)

- $V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$
- Imagine a slightly different procedure
  - Initialize V and V' arbitrarily
  - Keep running $V'(s_t) \leftarrow V'(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V'(s_t))$
  - Note that only V' is being updated; V doesn't change
  - What's the relationship between V and V' *after long enough*?
  - $V' = T^{\pi} V$! We've completed 1 iter of VI for solving V^{π}
  - Copy V' to V, and repeat this procedure again and again
- TD(0): almost the same, except that ***we don't wait***. Copy V' to V after every update!
- (Algorithms that "wait" actually have a come back in deep RL!)
- Optional reading: synchronous vs asynchronous updates in dynamic programming (for planning)

TD(0) vs MC

- TD(0) target: $r_t + \gamma V(s_{t+1})$
- MC target: $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$
- MC target is unbiased: expectation of target is the $V^\pi(s)$
- TD(0) target is biased (w.r.t. $V^\pi(s)$): the expected target is $(T^\pi V)(s)$
  - Although the expected target is not V^π, it's closer to V^π than where we are now (recall that T^π is a contraction)
- On the other hand, TD(0) has lower variance than MC
- Bias vs variance trade-off
- Also a practical concern: when interval of a time step is too small (e.g., in robotics), $V(s_t)$ and $V(s_{t+1})$ can be very close, and their difference can be buried by errors (error compounding over time)

TD(λ): Unifying TD(0) and MC

- 1-step bootstrap (=TD(0)): $r_t + \gamma V(s_{t+1})$
- 2-step bootstrap: $r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$ $=: G_t$
- 3-step bootstrap: $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3})$
- ...
- ∞-step bootstrap (=MC=TD(1)): $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$

$(T^\pi)^\infty V = V^\pi$.

- n-step bootstrap: as n increases, more variance, less bias
- Exercise: what's the expected target in n-step bootstrap? $(T^\pi)^n V$
- TD(λ): weighted combination of n-step bootstrapped target, with weighting scheme $\boxed{(1-\lambda)\lambda^{n-1}}$
  - $\lambda = 0$: only n=1 gets full weight. TD(0)
  - limit λ -> 1: (almost) MC, see pg 24 of Szepesvári
  - "forward view" of TD(λ)

$$\mathbb{E}[G_t \mid s_t] \triangleq$$

$$G_t = \underbrace{r_t + \gamma r_{t+1} + \gamma^2 \overset{\phi}{V}(s_{t+2})}$$

$$\mathbb{E}[G_t \mid s_t] = \mathbb{E}[\quad \downarrow \quad \mid s_t]$$

$$= \mathbb{E}[r_t + \gamma(\underbrace{r_{t+1} + \gamma V(s_{t+2})}) \mid s_t]$$

$$= \mathbb{E}[r_t \mid s_t] + \gamma\, \mathbb{E}_{s_{t+1} \mid s_t}\left[\mathbb{E}[r_{t+1} + \gamma V(s_{t+2}) \mid \cancel{s_t}, s_{t+1}]\right]$$

$$(\mathcal{T}^\pi V)(s_{t+1})$$

$$= \mathbb{E}[r_t + \gamma (\mathcal{T}^\pi V)(s_{t+1}) \mid s_t]$$

$$(\mathcal{T}^\pi(\mathcal{T}^\pi V))(s_t)$$

TD(λ): Unifying TD(0) and MC

- Why the choice of $(1-\lambda)\lambda^{n-1}$?
  - Enables efficient online implementation
  - “Backward view” of TD(λ)

Algorithm 3 The function that implements the tabular TD(λ) algorithm with replacing traces. This function must be called after each transition.

---

function TDLAMBDA(X, R, Y, V, z)

Input: X is the last state, Y is the next state, R is the immediate reward associated with this transition, V is the array storing the current value function estimate, z is the array storing the eligibility traces

$\delta \leftarrow R + \gamma \cdot V[Y] - V[X]$
for all $x \in \mathcal{X}$ **do** $s \in S.$
    $\boxed{z[x]} \leftarrow \gamma \cdot \lambda \cdot z[x]$
    if $X = x$ **then**
        $\boxed{z[x]} \leftarrow 1 + \gamma \cdot \lambda \cdot z[x]$
    end if
    $V[x] \leftarrow V[x] + \alpha \cdot \delta \cdot z[x]$
end for
return (V, z)

- Their X is our s_t
- Their Y is our s_{t+1}
- δ is the standard TD error (1-step)
- z is called the *eligibility trace*
- Every step we update at all states (TD(0) only updates V at the current state s_t)

- This code is the improved version with replacing traces; the original version has the red term

Equivalence between backward and forward view

- Will show in a simplified case
- An infinite trajectory, initial state s_1 only appears once, all updates are postponed til the end and "patched" together
- calculate the update for $V(s_1)$ according to the two views
- Forward view: (learning rate α omitted in all updates)
  - $(1-\lambda)\cdot(r_1+\gamma V(s_2)-V(s_1))$
  - $(1-\lambda)\lambda\cdot(r_1+\gamma r_2+\gamma^2 V(s_3)-V(s_1))$
  - $(1-\lambda)\lambda^2\cdot(r_1+\gamma r_2+\gamma^2 r_3+\gamma^3 V(s_4)-V(s_1))$, and so on

$(1-\lambda)\cdot\gamma.$

- Backward view:
  - $1\cdot(r_1+\gamma V(s_2)-V(s_1))$
  - $\lambda\gamma\cdot(r_2+\gamma V(s_3)-V(s_2))$
  - $\lambda^2\gamma^2\cdot(r_3+\gamma V(s_4)-V(s_3))$, and so on

$\gamma-\lambda\gamma$
$=(1-\lambda)\gamma$

$\delta \leftarrow R+\gamma\cdot V[Y]-V[X]$
for all $x\in\mathcal{X}$ **do**
    $z[x]\leftarrow\gamma\cdot\lambda\cdot z[x]$
    if $X=x$ **then**
        $z[x]\leftarrow 1$
    end if
    $V[x]\leftarrow V[x]+\alpha\cdot\delta\cdot z[x]$