Monte Carlo for prediction and control

Herke van Hoof

Last lecture

Dynamic programming

- Value iteration

  One round of value function updates using current V estimate

  Update policy (often implicitly)

- Policy iteration

  Rounds of value function updates till convergence
  (policy evaluation)

  Update policy

Policy iteration and value iteration

Dynamic programming requires knowing the transition probabilities!

- In RL, we typically assume we do not have know them up front…
- We thus need to **learn** something about the environment
- We could learn the transition probabilities
- Can be more effective to learn the value function directly
- And we can even learn a policy directly, without value function

Big picture

Finding optimal policies

Known MDP dynamics

$$p\left(s', r | s, a\right)$$

Dynamic programming

Policy iteration

Value iteration

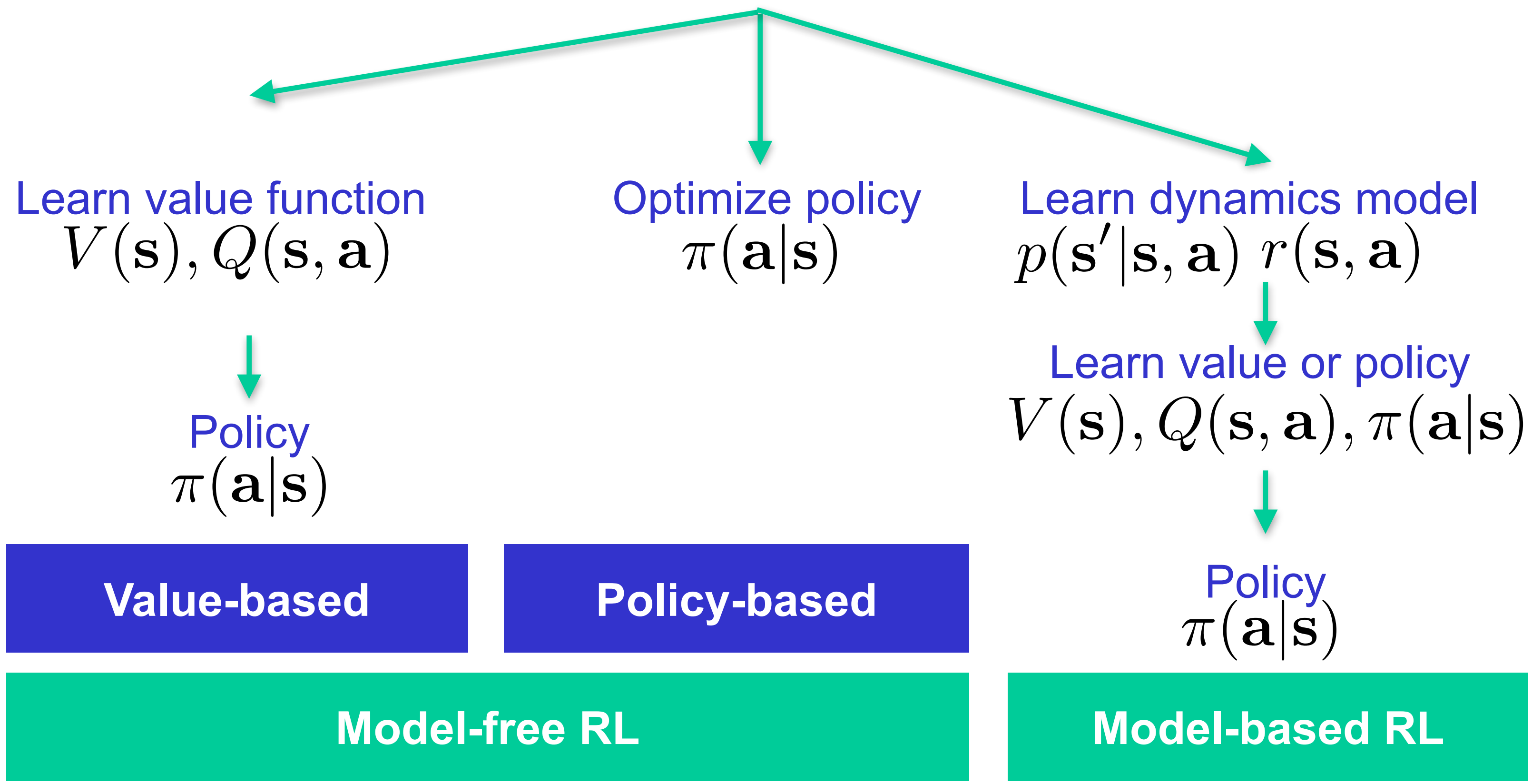
Only **data** from MDP

$$D = \left\{\left(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\right)\right\}_{i=1\ldots N}$$

Reinforcement learning

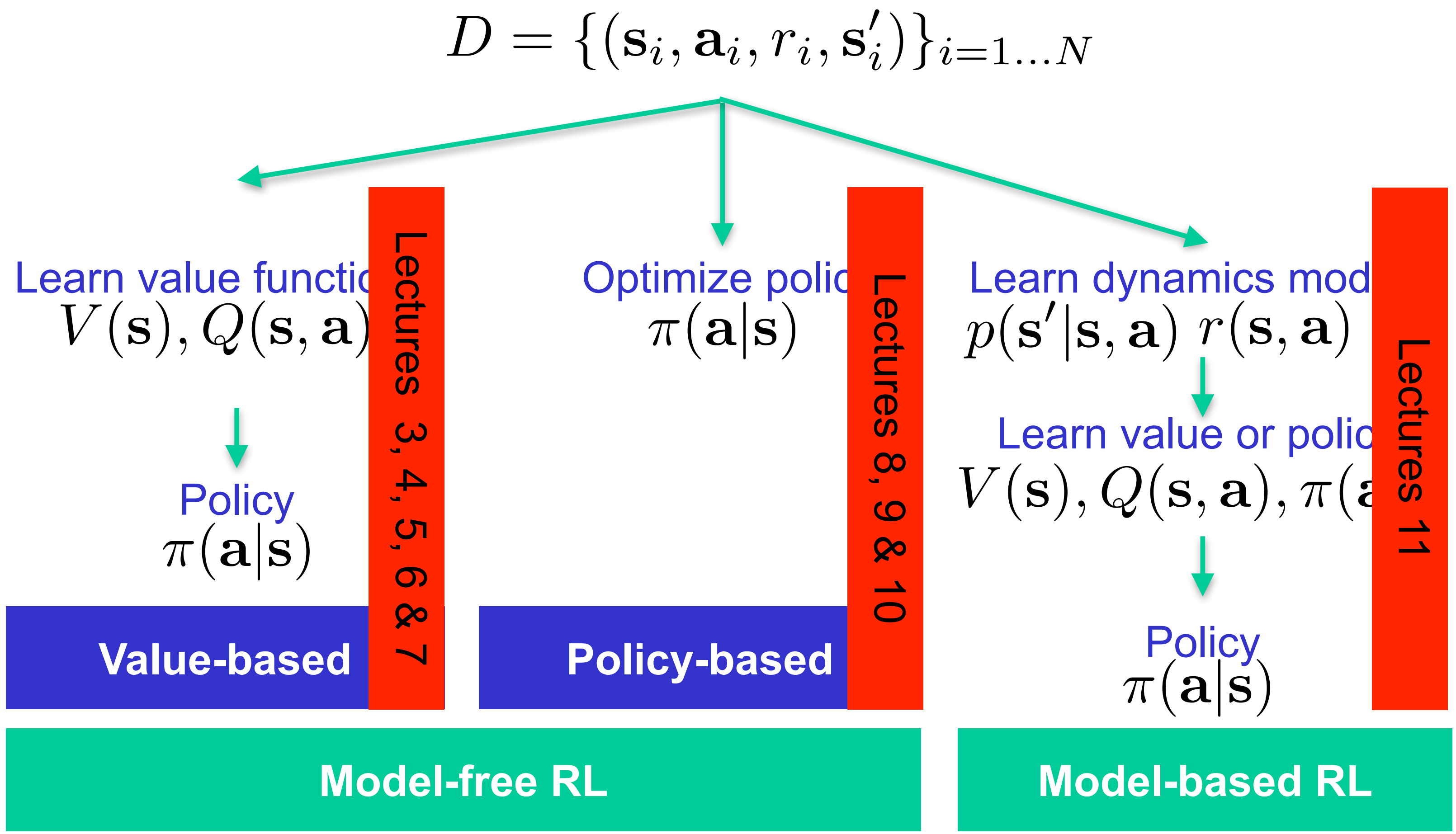
Big picture: How to learn policies

$$D = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_i')\}_{i=1\ldots N}$$



Thanks to Jan Peters

Big picture: How to learn policies



Thanks to Jan Peters

Episodes

Experience can be represented as one long *trajectory*

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Some tasks have a natural *terminal state* at which a trajectory of experience can be split into *episodes*

(e.g., game won or lost, exit of maze found)

$$S_0^{(1)}, A_0^{(1)}, R_1^{(1)}, S_1^{(1)}, A_1^{(1)}, \ldots, R_{T^{(1)}}^{(1)}, S_{T^{(1)}}^{(1)}$$
$$S_0^{(2)}, A_0^{(2)}, R_1^{(2)}, S_1^{(2)}, A_1^{(2)}, \ldots, R_{T^{(2)}}^{(2)}, S_{T^{(2)}}^{(2)}$$

In *episodic* tasks, each every interaction sequence eventually terminates

Monte Carlo prediction

We now want to *learn* value function

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t|S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s\right]$$

The expected (average) return under the policy can be approximated by simply trying the policy multiple times!

(Only if task is episodic)

First-visit Monte-Carlo: estimate value as average of return from the first visits (in each episode) to this state

MC prediction

Generate whole trajectory before any update

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow \text{average}(Returns(S_t))$

Figure: Sutton&Barto; RL:AI

MC prediction

Generate whole trajectory before any update

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$: ← Loop backwards in time

    $G \leftarrow \gamma G + R_{t+1}$

    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

      Append G to $Returns(S_t)$

      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

Figure: Sutton&Barto; RL:AI

MC prediction

Generate whole trajectory before any update

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Loop backwards in time

Here: only use the first visit
(Can also do every-visit MC)

Figure: Sutton&Barto; RL:AI

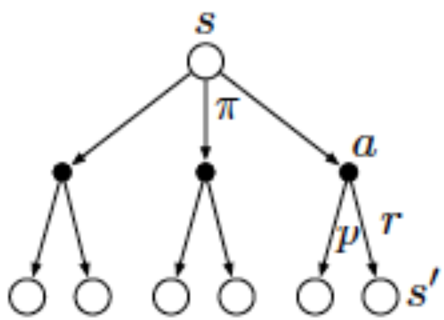
Monte Carlo and DP

Dynamic programming

Width: all next states

Depth: Looks one step ahead

Only possible if transitions known exactly



Backup diagram for v_π

Monte-Carlo

Width: a single possible future

Depth: until the end of episode

Only possible if task is episodic!

Figures: Sutton&Barto; RL:AI

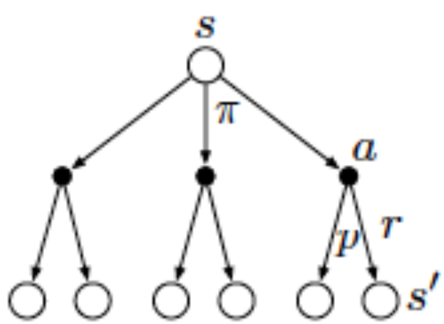
Monte Carlo and DP

Dynamic programming

Width: all next states

Depth: Looks one step ahead

Only possible if transitions known exactly



Backup diagram for v_π

Monte-Carlo

Width: a single possible future

Depth: until the end of episode

Only possible if task is episodic!

In what situations do we not know transition distribution, but are able to obtain experience trajectories?

Figures: Sutton&Barto; RL:AI

Monte Carlo for control

Can we now select actions with the learned value function v?

Monte Carlo for control

Can we now select actions with the learned value function v?

To do that, we again need to know the *dynamics function:*
Which actions lead to states with high v?

Instead, learn a state-action function q.
Very similar to learning v: average return from (s,a) pair visits

From some state s, for which actions will we learn a meaningful q function?

Monte Carlo for control

Guarantee every state-action pair continues to be visited:

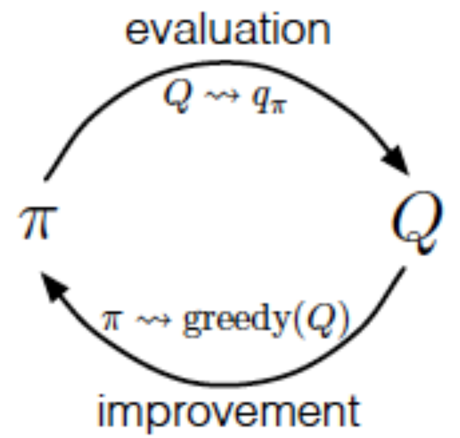
- 'Exploring starts': Start from random (s,a) pair.

Learned Q doesn't equal q_π with finite experience, but: move towards q_π

Generalized policy iteration suggests:

- If PE step moves Q closer to q_π
- And PU step moves π closer to greedy(Q)
- And all (s,a) pairs continue to be visited

We expect to eventually find optimal policy



$$\pi(s) \doteq \arg\max_a q(s,a)$$

Monte Carlo for control

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

  $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0

  Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

      Append G to $Returns(S_t, A_t)$

      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Figure: Sutton&Barto; RL:AI

Monte Carlo for control

‘Exploring starts’ requires starting from random (s,a) pair.

- Not always possible!

Other way to ensure we’ll visit all state-action pairs?

Monte Carlo for control

‘Exploring starts’ requires starting from random (s,a) pair.

- Not always possible!

Other way to ensure we’ll visit all state-action pairs?

- Like in the bandit case, use ‘soft’ policy that takes each action with non-zero probability

Getting rid of exploring starts

In policy improvement step, greedy update means exploration is lost…

Two possibilities:

- Change policy update to move “towards” greedy policy, but keep exploring
  We use data from the policy we are updating: **on-policy**
- Use two policies: non-greedy behaviour policy and greedy target policy
  Now, we are using data from a different policy then the one we are updating: **off-policy**

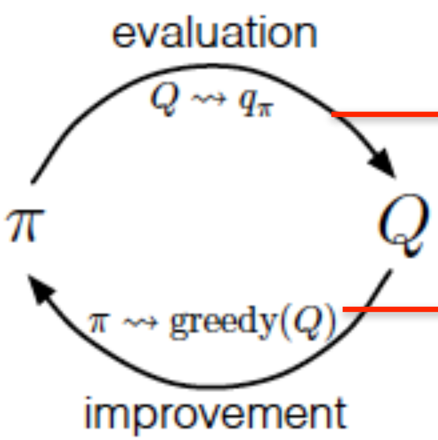
Both possibilities require a different approach. Distinction on-policy and off-policy important for many methods!

On-policy MC control

Ensure any action is taken with non-0 probability

Example: ε-greedy from first lecture

Again follow the GPI recipe:



Move ‘towards’ q_π

Move ‘towards’ greedy(Q) by making policy ε-greedy

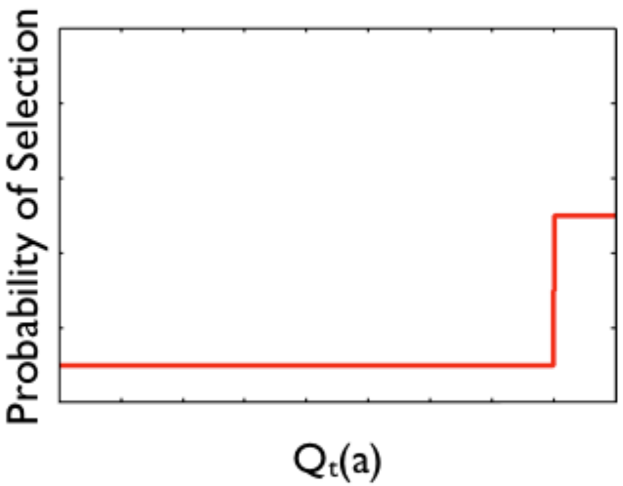


Figure: Sutton&Barto, RL:AI

GPI with epsilon-greedy policies

$$\pi_1 \rightarrow V_{\pi_1} \rightarrow \pi_2 \rightarrow V_{\pi_2} \rightarrow \pi_3 \rightarrow \ldots$$

ϵ-greedy under V_{π_1} ϵ-greedy under V_{π_2}

GPI with epsilon-greedy policies

$$\pi_1 \rightarrow V_{\pi_1} \rightarrow \pi_2 \rightarrow V_{\pi_2} \rightarrow \pi_3 \rightarrow \ldots$$

ϵ-greedy under V_{π_1} ϵ-greedy under V_{π_2}

Are 'improved' policies really always better?

GPI with epsilon-greedy policies

$$\pi_1 \rightarrow V_{\pi_1} \rightarrow \pi_2 \rightarrow V_{\pi_2} \rightarrow \pi_3 \rightarrow \ldots$$

ϵ-greedy under V_{π_1} ϵ-greedy under V_{π_2}

Are 'improved' policies really always better?

Useful concept:

ε-soft: policy takes any action with probability $> \varepsilon / |A(s)|$

From the first PI improvement step, all policies are "ϵ-soft"

Assume that π_1 is "ϵ-soft" as well

On-policy MC control

Is ε-greedy policy π' wrt q_π better then current ε-soft policy π?
We know this is the case if: (by PIT)

$$\sum_a \pi'(a|s) q_\pi(s,a) \overset{?}{\geq} v_{\pi(s)}$$

On-policy MC control

Is ε-greedy policy π' wrt q_π better then current ε-soft policy π?
We know this is the case if: (by PIT)

$$\sum_a \pi'(a|s) q_\pi(s,a) \overset{?}{\geq} v_{\pi(s)}$$

$$\sum_a \pi'(a|s) q_\pi(s,a) = \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \underline{\max_a q_\pi(s,a)}$$

$$v_{\pi(s)} = \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + \sum_a \pi(a|s) q_\pi(s,a)$$

$$= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \underline{\sum_a \frac{\boxed{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}}{1-\varepsilon} q_\pi(s,a)}$$

π needs to be ε-soft

convex combination ≤ max

On-policy MC control

We now know:

- if we start with any policy that takes any action with probability at least ε/|A| (ε-soft) *e.g. old ε-greedy policy*
- The new ε-greedy policy will be at least as good wrt the used q function. The policy improvement theorem then implies that:
  $\pi' \geq \pi \, (\text{i.e.}, v_{\pi'}(s) \geq v_\pi(s), \text{ for all } s \in \mathcal{S})$

Furthermore, it can be proven that the policy does not improve only if it is already optimal among ε-soft policies

So, with ε-greedy policies, the policy improvement step indeed either leads to policy improvement or the policy is already the optimal ε-soft policy for current q.

Thus, GPI will converge to the optimal ε-soft policy

On-policy MC control with ϵ-greedy

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Make π ε-greedy

On-policy MC control with ϵ-greedy

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\quad \pi \leftarrow$ an arbitrary ε-soft policy

$\quad Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$\quad Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

$\quad$ Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$\quad G \leftarrow 0$

$\quad$ Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$\quad\quad G \leftarrow \gamma G + R_{t+1}$

$\quad\quad$ Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

$\quad\quad\quad$ Append G to $Returns(S_t, A_t)$

$\quad\quad\quad Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\quad\quad\quad A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

$\quad\quad\quad$ For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Make π ε-greedy

Need to store all returns?

Incremental implementation

Incremental implementation briefly discussed in lecture 1

General equation:
$$\hat{V}(S_t) \leftarrow \hat{V}(S_t) + \frac{1}{k_s + 1}\Big(G_t - \hat{V}(S_t)\Big)$$

Popular alternative: fixed learning rate / learning rate schedule
$$\hat{V}(S_t) \leftarrow \hat{V}(S_t) + \alpha\Big(G_t - \hat{V}(S_t)\Big)$$

Incremental implementation

Incremental implementation briefly discussed in lecture 1

General equation: $$\hat{V}(S_t) \leftarrow \hat{V}(S_t) + \frac{1}{k_s + 1}\Big(G_t - \hat{V}(S_t)\Big)$$

Popular alternative: fixed learning rate / learning rate schedule

$$\hat{V}(S_t) \leftarrow \hat{V}(S_t) + \alpha\Big(G_t - \hat{V}(S_t)\Big)$$

With fixed learning rate, there is more weight on recent transitions; old information gradually gets forgotten.

An (incremental) average never forgets: more efficient, but problematic if environment changes.

Convergence proofs often require certain decreasing schedules

Off-policy learning

We ‘only’ obtain the best ε-soft policies. Can we do better?

Use two policies:

- non-greedy behaviour policy b
- greedy target policy π

Data from a different policy than updated one: **off-policy**

Off-policy learning

First consider predicting v_π using data from b

Coverages assumption:

$$\pi(a|s) > 0 \rightarrow b(a|s) > 0$$

(always satisfied if b is ε-soft)

Off-policy learning

Consider trajectories

$$\tau_t = S_t, A_t, \ldots, S_T$$

We know the normal Monte Carlo approximation

$$\begin{aligned} v_{\boxed{b}}(s) = \mathbb{E}_b[G(\tau_t)|S_t = s, A_t \sim \boxed{b}] &= \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim \boxed{b})G(\tau_t) \\ &\approx \sum_{i=1}^{n} \frac{1}{n} G(\tau^i) \end{aligned}$$

where τ^i are sampled using b starting at state s

Off-policy learning

$$v_{\pi}(s) = \mathbb{E}[G(\tau_t)|S_t = s, A_t \sim \pi] = \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim \pi)G(\tau_t)$$

How to approximate this expectations using samples from π?

How about samples from b?

Importance sampling

Let's try importance weights

$$v_{\pi}(s) = \mathbb{E}[G(\tau_t)|S_t = s, A_t \sim \pi] = \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim \pi)G(\tau_t)$$

Importance sampling

Let's try importance weights

$$v_\pi(s) = \mathbb{E}[G(\tau_t)|S_t = s, A_t \sim \pi] = \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim \pi)G(\tau_t)$$

$$= \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim \pi)\frac{p(\tau_t|S_t = s, A_t \sim b)}{p(\tau_t|S_t = s, A_t \sim b)}G(\tau_t)$$

$$= \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim b)\frac{p(\tau_t|S_t = s, A_t \sim \pi)}{p(\tau_t|S_t = s, A_t \sim b)}G(\tau_t)$$

$$\approx \sum_{i=1}^{n} \frac{1}{n}\underbrace{\boxed{\frac{p(\tau_t|S_t = s, A_t \sim \pi)}{p(\tau_t|S_t = s, A_t \sim b)}}}_{=\rho_{t:T-1}}G(\tau^i)$$

Approximating an expectation based on b, so sample τ^i from b!

Importance sampling

Let's look at the importance weights

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right)}{\prod_{k=t}^{T-1} b\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right)}$$

Importance sampling

Let's look at the importance weights

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi\left(A_k | S_k\right) p\left(S_{k+1} | S_k, A_k\right)}{\prod_{k=t}^{T-1} b\left(A_k | S_k\right) p\left(S_{k+1} | S_k, A_k\right)} = \prod_{k=t}^{T-1} \frac{\pi\left(A_k | S_k\right)}{b\left(A_k | S_k\right)}$$

Luckily, the weights don't depend on the transition dynamics!

Use this to re-weight trajectories following visits to s (first- or every-visit)

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}$$

set of time steps with visits to s

Importance sampling

Weighted importance sampling is an alternative

ordinary i.s.

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}$$

weighted i.s.

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

Importance sampling

Weighted importance sampling is an alternative

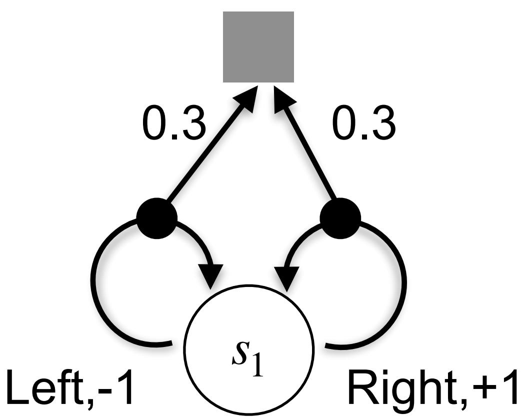
	Extreme case: one trajectory	ρ=10 or ρ=0.1
ordinary i.s. $V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\lvert \mathcal{T}(s) \rvert}$	ρ/1 can be >1 or <1 ⇒good on average	Estimate very high or very low! High variance!
weighted i.s. $V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$		

Importance sampling

Weighted importance sampling is an alternative

	Extreme case: one trajectory	ρ=10 or ρ=0.1
ordinary i.s. $V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\lvert\mathcal{T}(s)\rvert}$	ρ/1 can be >1 or <1 ⇒good on average	Estimate very high or very low! High variance!
weighted i.s. $V(s) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}}$	ρ / ρ =1, always ⇒biased	Estimates close together Low variance!

Example



$$b(a) = \begin{cases} 0.2 & a = \text{left} \\ 0.8 & a = \text{right} \end{cases}$$

$$\pi(a) = \begin{cases} 0 & a = \text{left} \\ 1 & a = \text{right} \end{cases}$$

$$\tau = (s_1, \text{left}, -1, s_1, \text{right}, +1, \text{terminal})$$

$$\rho_{1:1} = ?$$

$$\rho_{0:1} = ?$$

Off-policy Monte Carlo control

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
total weight $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
evaluation $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)}[G - Q(S_t, A_t)]$
improvement $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode) stop! (why?)
weight $W \leftarrow W \frac{1}{b(A_t|S_t)}$

$$\rho = \prod_t \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

Figure: Sutton&Barto; RL:AI

Importance sampling

First-visit with ordinary importance sampling is unbiased (expected value equal to true value function)

Every-visit MC or using weighted importance sampling biased

- But weighted i.s. has much lower variance, so typically lower errors
- Weighed i.s. typically preferred
- Every-visit MC easier to implement
- Bias falls asymptotically to 0 as number of samples increases

Incremental implementation are possible

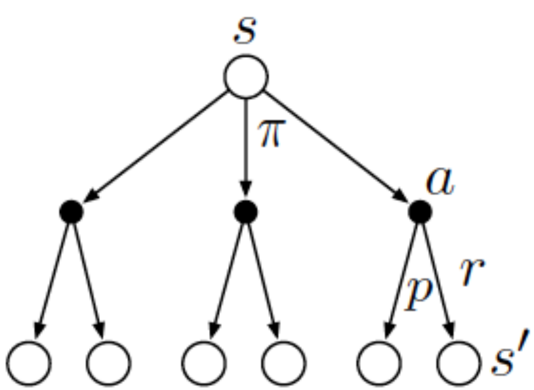
- Bit more complicated for weighted i.s., see book

Off-policy learning

On-policy	Off-policy
Often simpler	Often more complex
Specific case	More general (we can have $b=\pi$)
Often converges faster	Often large variance or slow convergence
Only data gathered with current policy	Can reuse data, use data from other source
Generally needs non-greedy policy	Allows greedy target policy

So far

Dynamic programming



Need successor distribution

Uses structure of value function

Monte Carlo

Needs samples only

Unbiased updates possible

Ignores structure

High variance, especially with long updates

Limits of Monte Carlo

Constant-α MC learns by updating in direction of return G, moving roughly in direction of the true value (expected return):

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Target

However:

- Only know G when episode is over. What if we have a continuing task?

What you should know

How do Monte-Carlo methods learn value functions and what are their properties?

Why do we need ‘exploring starts’, non-greedy policies, or offline learning, and what are the advantages of each?

What are ordinary and weighted importance sampling and when are they used?

Thanks for your attention!

Feedback?

h.c.vanhoof@uva.nl