Reinforcement Learning

Dynamic Programming (part 2) and Monte Carlo Methods

Michael Herrmann, David Abel

Based on slides by Stefano V. Albrecht

31 January 2025

THE UNIVERSITY of EDINBURGH
informatics

Lecture Outline

- Value Iteration
- Dynamic programming (part 2)
- DP examples

- Monte Carlo policy evaluation
- Monte Carlo control with...
    - Exploring starts
    - Soft policies
    - Off-policy learning
- Importance sampling

Policy Iteration and Value Iteration

Policy Iteration

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad a \leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   $\quad$ If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

Value Iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

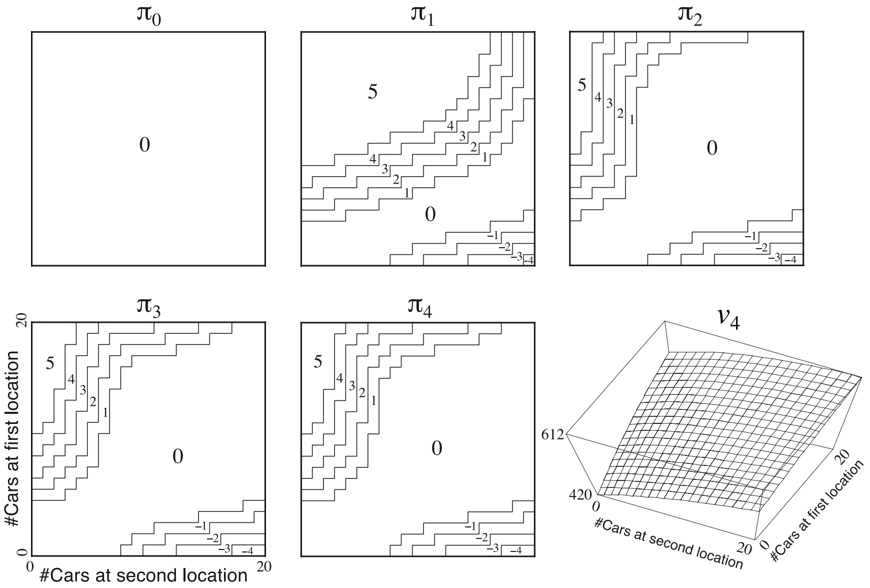
Repeat
$\quad \Delta \leftarrow 0$
$\quad$ For each $s \in \mathcal{S}$:
$\quad\quad v \leftarrow V(s)$
$\quad\quad V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
$\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
$\quad \pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$

Example: Jack's Car Rental

- Two car rental locations
- Cars are requested and returned randomly based on a distribution (see book)
- States: (n_1, n_2) where n_i is number of cars at location i (max 20 each)
- Actions: number of cars moved from one location to other (max 5)
  (positive is from location 1 to 2, negative is from 2 to 1)
- Rewards:
  +\$10 per rented car in time step
  −\$2 per moved car in time step
- $\gamma = 0.9$

Example: Jack's Car Rental



Value Iteration

Iterative policy evaluation may take many sweeps $v_k \rightarrow v_{k+1}$ to converge

Do we have to wait until convergence before policy improvement?

$k = 3$

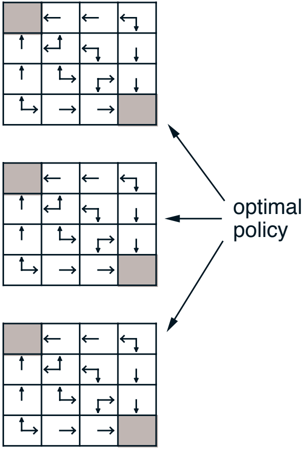
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Value Iteration

Iterative policy evaluation uses Bellman equation as operator:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s')\right] \quad \text{for all } s \in \mathcal{S}$$

Value iteration uses *Bellman optimality equation* as operator:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s')\right] \quad \text{for all } s \in \mathcal{S}$$

- Combines one sweep of iterative policy evaluation and policy improvement
- Sequence converges to optimal policy
  (can show that Bellman optimality operator is γ-contraction)

Value Iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
    $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Asynchronous Dynamic Programming

DP methods so far perform exhaustive *sweeps*:

Policy evaluation and improvement for all $s \in \mathcal{S} \;\Rightarrow\;$ prohibitive if state space large!

Asynchronous DP methods evaluate and improve policy on subset of states:

- Gives flexibility to choose best states to update

  $\Rightarrow$ e.g. random states, recently visited states (real-time DP)

- Can perform updates *in parallel* on multiple processors

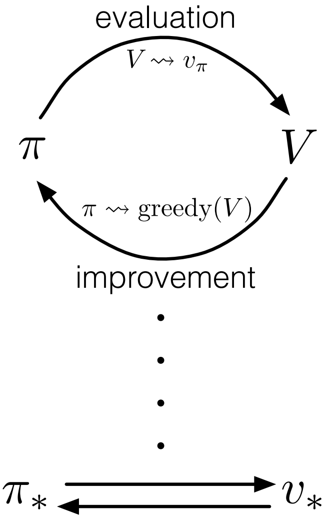
- Still guaranteed to converge to optimal policy if all states in $\mathcal{S}$ are updated infinitely many times in the limit

Conclusion on Dynamic Programming

DP methods iterate through policy evaluation and improvement until convergence to optimal value function v_* and policy π_*

- Policy evaluation via repeated application of Bellman operator
- Requires complete knowledge of MDP model: $p(s', r|s, a)$

Can we compute optimal policy without knowledge of complete model?



Monte Carlo Policy Evaluation

Monte Carlo (MC) methods learn value function based on experience

- Experience: entire episodes $E^i = < S_0^i, A_0^i, R_1^i, S_1^i, A_1^i, R_2^i, ..., S_{T_i}^i >$

MC does not require complete model $p(s', r|s, a)$, only requires sampled episodes

Two ways to obtain episodes:

- **Real experience:** generate episodes directly from “real world”
- **Simulated experience:** use simulation model $\hat{p}$ to sample episodes
  — $\hat{p}(s, a)$ returns a pair (s', r) with probability $p(s', r|s, a)$

Monte Carlo Policy Evaluation

Monte Carlo (MC) Policy Evaluation:

- Estimate value function by averaging sample returns:

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[\sum_{k=t}^{T-1} \gamma^{k-t} R_{k+1} | S_t = s\right] \approx \frac{1}{|\mathcal{E}(s)|} \sum_{t_i \in \mathcal{E}(s)} \sum_{k=t_i}^{T_i - 1} \gamma^{k-t_i} R^i_{k+1}$$

where for each past episode $E^i = < S^i_0, A^i_0, R^i_1, S^i_1, A^i_1, R^i_2, ..., S^i_{T_i} >$:

— **First-visit MC:** $\mathcal{E}(s)$ contains *first* time t_i for which $S^i_{t_i} = s$ in E^i

— **Every-visit MC:** $\mathcal{E}(s)$ contains *all* times t_i for which $S^i_{t_i} = s$ in E^i

- Both methods converge to $v_\pi(s)$ as $|\mathcal{E}(s)| \to \infty$

First-Visit Monte Carlo Policy Evaluation

See Tutorial 5

Initialize:

- $\pi \leftarrow$ policy to be evaluated
- $V \leftarrow$ an arbitrary state-value function
- $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

- Generate an episode using π
- For each state s appearing in the episode:
  - $G \leftarrow$ return following the first occurrence of s
  - Append G to $Returns(s)$
  - $V(s) \leftarrow \text{average}(Returns(s))$

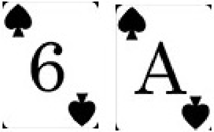
Example: Blackjack

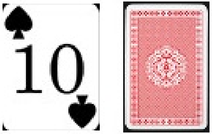
Initial state:

Player

Dealer

Ace worth 1 or 11





Hidden card

First, player samples cards from deck (hit) until stop (stick)

Then, dealer samples cards from deck (hit) until sum > 16 (stick)

Player loses (-1 reward) if bust (card sum > 21)
Player wins (+1 reward) if Dealer bust or Player sum > Dealer sum

Example: Blackjack

Player policy π:

stick if player sum is 20
or 21, else hit

Estimate of v_π using MC ...

States s (3-tuple):

– Player sum (12–21)
– Dealer card (ace–10)
– Usable ace?

Example: Blackjack

Player policy π:

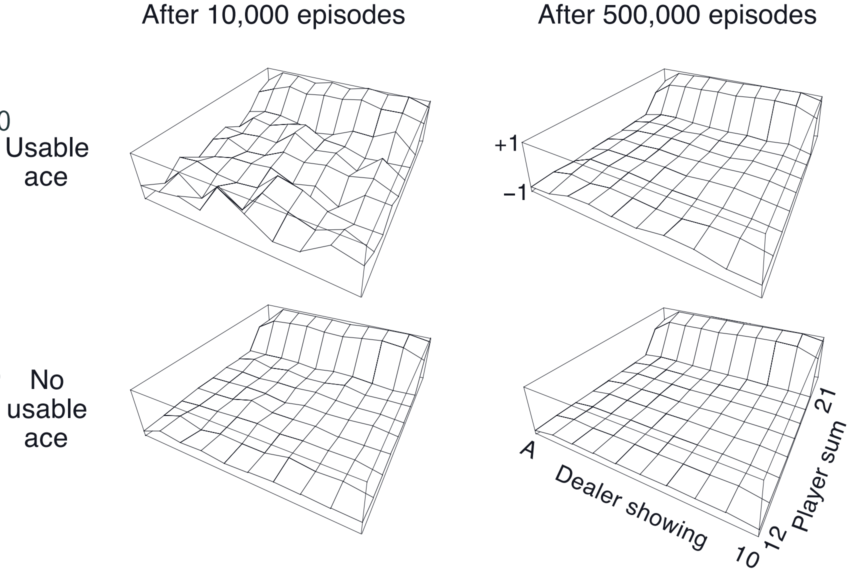
stick if player sum is 20 or 21, else hit

States s (3-tuple):

– Player sum (12–21)

– Dealer card (ace–10)

– Usable ace?



States in Blackjack

Couldn't we just define states as $S_t = \{\text{Player cards, Dealer card}\}$?

- Tricky: states would have variable length (player cards)
- If we fix maximum number of player cards to 4, then there are $10^5 = 100,000$ possible states! (ignoring face cards and ordering)

States in Blackjack

Couldn't we just define states as $S_t = \{\text{Player cards, Dealer card}\}$?

- Tricky: states would have variable length (player cards)
- If we fix maximum number of player cards to 4, then there are $10^5 = 100,000$ possible states! (ignoring face cards and ordering)

Blackjack example uses engineered state features:

- Fixed length: $S_t = (\text{Player sum, Dealer card, Usable ace?})$
- Player sum limited to range 12–21 because decision below 12 is trivial (always hit)
- Number of states: $10 * 10 * 2 = 200 \rightarrow$ much smaller problem!
- Still has all relevant information

Blackjack and Dynamic Programming

Can we solve Blackjack MDP with DP methods?

- Yes, in principle, because we know complete MDP
- But computing $p(s', r|s, a)$ can be complicated!
  E.g. what is probability of $+1$ reward as function of Dealer's showing card?

Blackjack and Dynamic Programming

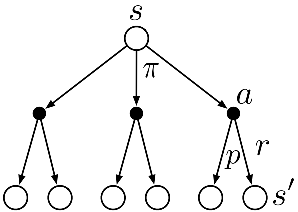
Can we solve Blackjack MDP with DP methods?

- Yes, in principle, because we know complete MDP
- But computing $p(s', r|s, a)$ can be complicated!
  E.g. what is probability of $+1$ reward as function of Dealer's showing card?
- On other hand, easy to code a simulation model:
  — Use Dealer rule to sample cards until stick/bust, then compute reward
  — Reward outcome is distributed by $p(s', r|s, a)$
- MC can evaluate policy without knowledge of probabilities $p(s', r|s, a)$

Monte Carlo Estimation of Action Values

MC methods can learn v_π without knowledge of model $p(s', r|s, a)$

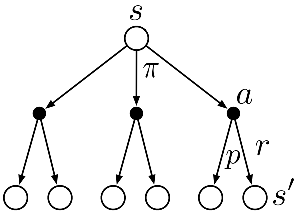
$\Rightarrow$ But improving policy π from v_π requires model *(why?)*



Monte Carlo Estimation of Action Values

MC methods can learn v_π without knowledge of model $p(s', r|s, a)$

$\Rightarrow$ But improving policy π from v_π requires model *(why?)*



Must estimate action values:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

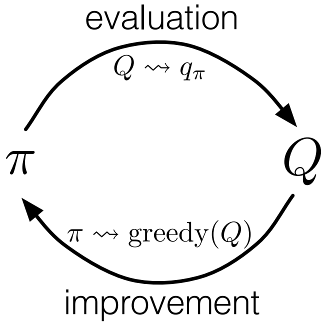
- Improve policy without model: $\pi'(s) = \arg\max_a q_\pi(s, a)$
- Use same MC methods to learn q_π, but visits are to (s, a)-pairs
- Converges to q_π if every (s, a)-pair visited infinitely many times in limit

  E.g. exploring starts: every (s, a)-pair has non-zero probability of being starting pair of episode

Monte Carlo Control

- **MC policy evaluation:**
  Estimate q_π using MC method

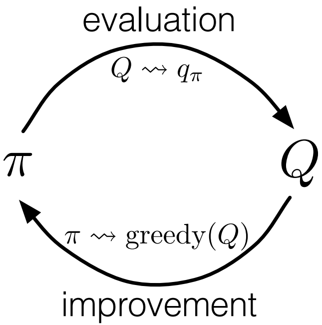
- **Policy improvement:**
  Improve π by making greedy wrt q_π



Monte Carlo Control with Exploring Starts

Greedy policy meets conditions for policy improvement theorem:

$$
\begin{aligned}
q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s, a)) \\
&= \max_a q_{\pi_k}(s, a) \\
&\geq q_{\pi_k}(s, \pi_k(s)) \quad (\textit{why?}) \\
&= v_{\pi_k}(s)
\end{aligned}
$$



Assumes exploring starts and *infinite* MC iterations

- In practice, we update only to a given performance threshold
- Or alternate between evaluation and improvement per episode

Monte Carlo Control with Exploring Starts

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$\pi(s) \leftarrow$ arbitrary

$Returns(s, a) \leftarrow$ empty list

Repeat forever:

Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability > 0

Generate an episode starting from S_0, A_0, following π

For each pair s, a appearing in the episode:

$G \leftarrow$ return following the first occurrence of s, a

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

For each s in the episode:

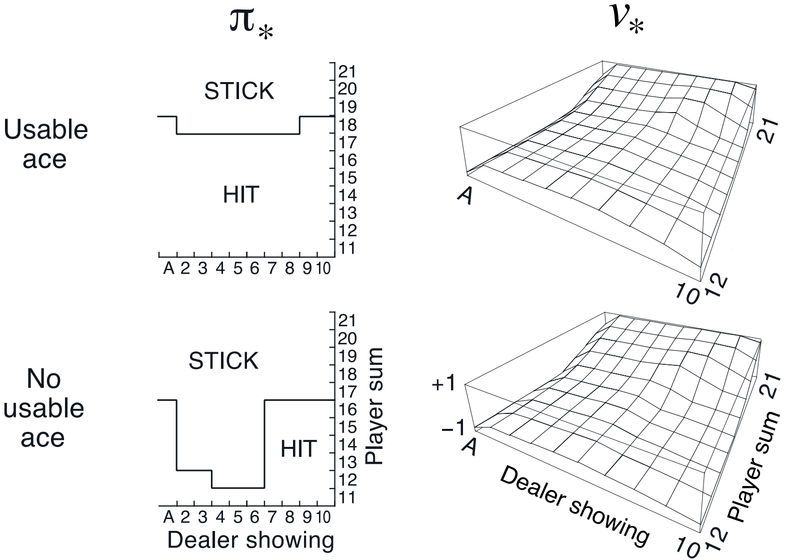
$\pi(s) \leftarrow \arg\max_a Q(s, a)$

Policy π:

stick if player sum is 20 or 21, else hit

Exploring starts:

sample initial states uniformly randomly



Monte Carlo Control with Soft Policies

Convergence to q_π requires that all (s, a)-pairs are visited infinitely many times

- Exploring starts guarantee this, but impractical *(why?)*

Monte Carlo Control with Soft Policies

Convergence to q_π requires that all (s, a)-pairs are visited infinitely many times

- Exploring starts guarantee this, but impractical *(why?)*

Other approach: use soft policy such that $\pi(a|s) > 0$ for all s, a

- e.g. ϵ-soft policy: $\pi(a|s) \geq \epsilon/|\mathcal{A}|$ for $\epsilon > 0$
- **Policy improvement:** make policy ϵ-greedy wrt q_π

$$\pi'(a|s) \doteq \begin{cases} \epsilon/|\mathcal{A}| + (1-\epsilon) & \text{if} \quad a = \arg\max_{a'} q_\pi(s, a') \\ \epsilon/|\mathcal{A}| & \text{else} \end{cases}$$

Monte Carlo Control with Soft Policies

ϵ-greedy policy meets conditions for policy improvement theorem:

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s)\, q_\pi(s,a) \\
&= \frac{\epsilon}{|\mathcal{A}|} \sum_a q_\pi(s,a) + (1-\epsilon) \max_a q_\pi(s,a) \\
&\geq \frac{\epsilon}{|\mathcal{A}|} \sum_a q_\pi(s,a) + (1-\epsilon) \sum_a \frac{\pi(a|s) - \epsilon/|\mathcal{A}|}{1-\epsilon} q_\pi(s,a) \quad \textit{(why?)} \\
&= \frac{\epsilon}{|\mathcal{A}|} \sum_a q_\pi(s,a) - \frac{\epsilon}{|\mathcal{A}|} \sum_a q_\pi(s,a) + \sum_a \pi(a|s)\, q_\pi(s,a) \\
&= v_\pi(s)
\end{aligned}
$$

- Thus, π' better or equal to π, but both are still ϵ-soft
- $q_\pi(s, \pi'(s)) = v_\pi(s)$ only when π' and π both optimal ϵ-soft policies

Monte Carlo Control with Soft Policies

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$Returns(s, a) \leftarrow$ empty list

$\pi(a|s) \leftarrow$ an arbitrary ε-soft policy

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$G \leftarrow$ return following the first occurrence of s, a

Append G to $Returns(s, a)$

$Q(s, a) \leftarrow$ average$(Returns(s, a))$

(c) For each s in the episode:

$A^* \leftarrow \arg\max_a Q(s, a)$

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

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

Off-Policy Methods

Like exploring starts, soft policies ensure all (s, a) are visited infinitely many times

- But policies restricted to be soft

  $\Rightarrow$ Optimal policy is usually deterministic!

- Could slowly reduce ϵ, but not clear how fast

Off-Policy Methods

Like exploring starts, soft policies ensure all (s, a) are visited infinitely many times

- But policies restricted to be soft

  $\Rightarrow$ Optimal policy is usually deterministic!

- Could slowly reduce ϵ, but not clear how fast

Other approach: off-policy learning

- Learn q_π based on experience generated with *behaviour policy* $\mu \neq \pi$
- Requires "coverage": if $\pi(a|s) > 0$ then $\mu(a|s) > 0$, for all s, a

  — e.g. use soft policy μ

- π can be deterministic $\rightarrow$ usually the greedy policy

Discussion: On-Policy vs Off-Policy Methods

On-policy:

Learn q_π with experience generated using policy π

Off-policy:

Learn q_π with experience generated using policy $\mu \neq \pi$

Importance Sampling Ratio

We have episodes generated from μ

$\Rightarrow$ Expected return at t is $\mathbb{E}_\mu[G_t|S_t = s] = v_\mu(s)$

Importance Sampling Ratio

We have episodes generated from μ

$\Rightarrow$ Expected return at t is $\mathbb{E}_\mu[G_t|S_t = s] = v_\mu(s)$

Fix expectation with sampling importance ratio:

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

- $\mathbb{E}_\mu[\rho_{t:T}\, G_t|S_t = s] = v_\pi(s)$

Importance Sampling Ratio

$$\mathbb{E}_{\mu}[\rho_{t:T}\, G_t | S_t = s] = \sum_{E:S_t=s} \left[\prod_{k=t}^{T-1} \mu(A_k|S_k)\, p(S_{k+1}, R_{k+1}|S_k, A_k) \right] \rho_{t:T}\, G_t$$

$$= \sum_{E:S_t=s} \left[\prod_{k=t}^{T-1} \mu(A_k|S_k)\, p(S_{k+1}, R_{k+1}|S_k, A_k) \right] \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}\, G_t$$

$$= \sum_{E:S_t=s} \left[\prod_{k=t}^{T-1} \pi(A_k|S_k)\, p(S_{k+1}, R_{k+1}|S_k, A_k) \right] G_t$$

$$= v_{\pi}(s)$$

Evaluating Policies with Importance Sampling

Denote episodes $E^i = < S^i_0, A^i_0, R^i_1, S^i_1, A^i_1, R^i_2, ..., S^i_{T_i} >$

Define $\mathcal{E}(s)/\mathcal{E}(s,a)$ as before for first-visit or every-visit MC

Estimate v_π / q_π as

$$v_\pi(s) \;\approx\; \eta^{-1} \sum_{t_i \in \mathcal{E}(s)} \rho_{t_i:T_i}\, G^i_{t_i}$$

$$q_\pi(s,a) \;\approx\; \eta^{-1} \sum_{t_i \in \mathcal{E}(s,a)} \rho_{t_i+1:T_i}\, G^i_{t_i} \qquad \textit{(why } t_i + 1\textit{?)}$$

- **Ordinary** importance sampling: $\eta = |\mathcal{E}(s,a)|$
- **Weighted** importance sampling: $\eta = \sum_{t_i \in \mathcal{E}(s)} \rho_{t_i:T_i}$ resp. $\eta = \sum_{t_i \in \mathcal{E}(s,a)} \rho_{t_i+1:T_i}$

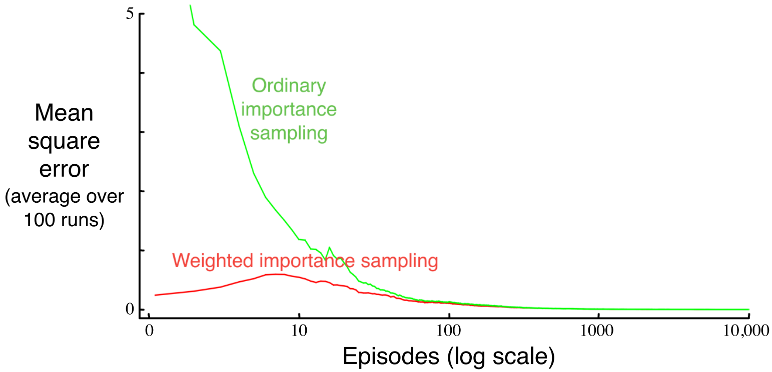
Off-Policy Value Estimation in Blackjack Example

π : stick if player sum is 20 or 21, else hit

μ : uniformly random

s : player sum 13
dealer showing 2
usable ace

True value:
$v_\pi(s) \approx -0.27726$



Reading

Required:

- RL book, Chapter 5 (5.1–5.7)

Optional:

- *Sequential Monte Carlo Methods in Practice*
  Arnaud Doucet, Nando de Freitas, Neil Gordon (editors)
  University library has copies