Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Monte Carlo Learning

Spring 2020, CMU 10-403

Katerina Fragkiadaki

Used Materials

- **Disclaimer:** Much of the material and slides for this lecture were borrowed from Russ who in turn borrowed some materials from Rich Sutton's class and David Silver's class on Reinforcement Learning.

Summary so far

- So far, to estimate value functions we have been using dynamic programming with *known* rewards and dynamics functions:

$$v_{[k+1]}(s) = \sum_a \pi(a \mid s) \left(r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) v_{[k]}(s') \right), \forall s$$

$$v_{[k+1]}(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) v_{[k]}(s') \right), \forall s$$

Q: Was our agent interacting with the world? Was our agent exploring?

A: 1) No. 2) No, if you know everything, there is nothing to explore.

Coming up

- So far, to estimate value functions we have been using dynamic programming with *known* rewards and dynamics functions:

$$v_{\pi,[k+1]}(s) = \sum_{a} \pi(a \mid s)\left(r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) v_{\pi,[k]}(s') \right), \forall s$$

$$v_{[k+1]}(s) = \max_{a \in \mathscr{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathscr{S}} p(s' \mid s,a) v_{[k]}(s') \right), \forall s$$

- Next: estimate value functions and policies from interaction experience, without known rewards or dynamics.
- How? By sampling all the way. Instead of probabilities distributions to compute expectations, we will use empirical expectations by averaging sampled returns.

Monte Carlo (MC) Methods

- Monte Carlo methods are learning methods
  - Experience → values, policy
- Monte Carlo methods learn from complete sampled trajectories and their returns.
  - Only defined for episodic tasks .
  - All episodes must terminate.
- Monte Carlo uses the simplest possible idea: value = mean return

Monte-Carlo Policy Evaluation

- Goal: learn $v_\pi(s)$ from episodes of experience under policy π:

$$S_1, A_1, R_2, ..., S_k \sim \pi$$

- Remember that the return is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

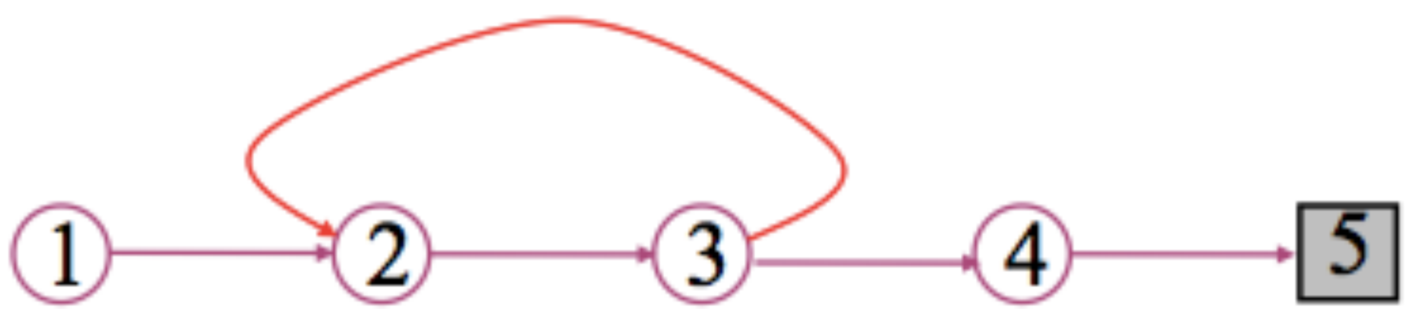
- Remember that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

- Monte-Carlo policy evaluation **uses empirical mean return instead of expected return**

Monte-Carlo Policy Evaluation

- Goal: learn $v_\pi(s)$ from episodes of experience under policy π:
- Idea: Average returns observed after visits to s:



- Every-Visit MC: average returns for every time s is visited in an episode
- First-visit MC: average returns only for first time s is visited in an episode
- Both converge asymptotically based on the law of large numbers

First-Visit MC Policy Evaluation

- To evaluate state s
- The first time-step t that state s is visited in an episode,
  - Increment counter: $N(s) \leftarrow N(s) + 1$
  - Increment total return: $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Law of large numbers

Every-Visit MC Policy Evaluation

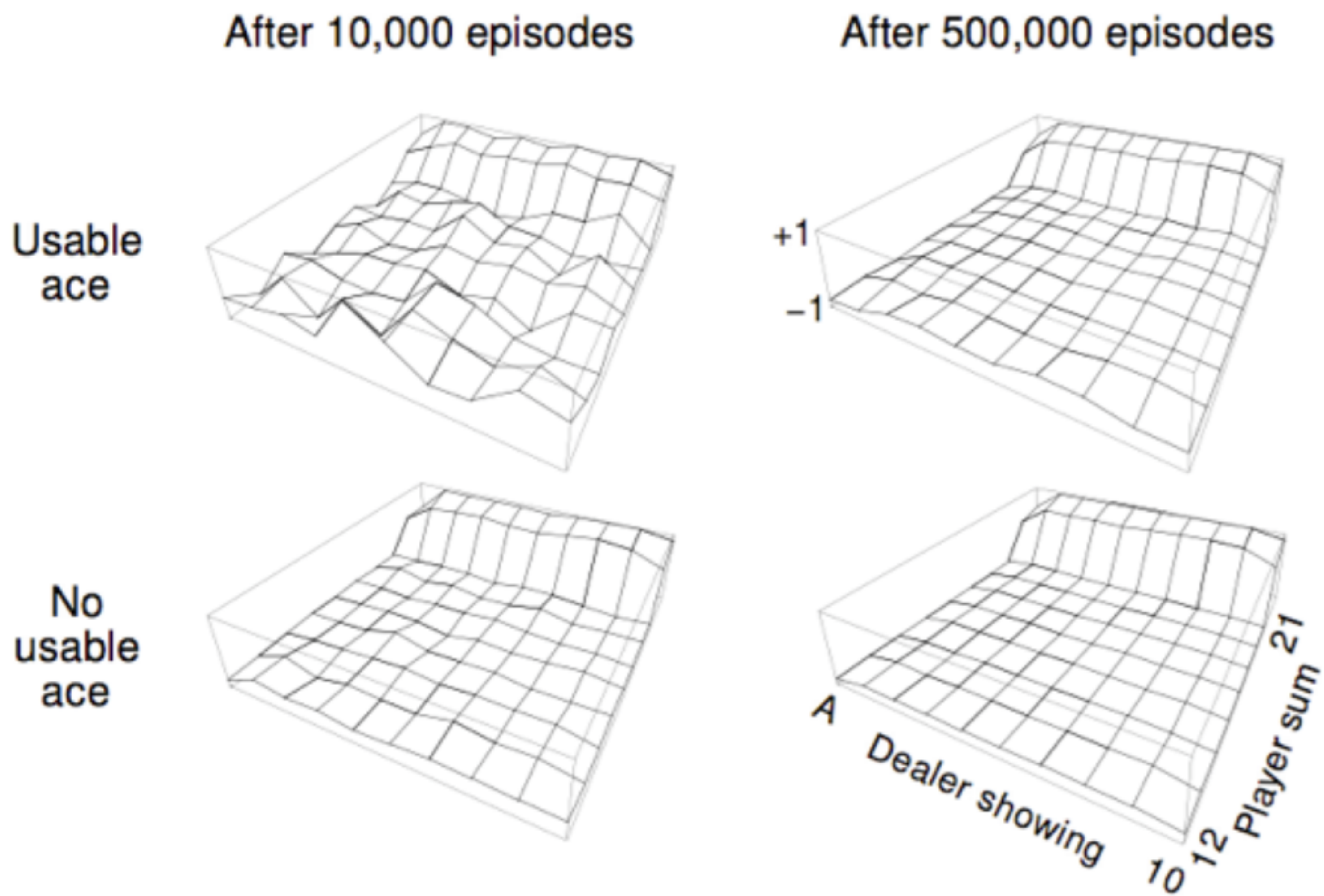
- To evaluate state s
- Every time-step t that state s is visited in an episode,
  - Increment counter: $N(s) \leftarrow N(s) + 1$
  - Increment total return: $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers $V(s) \to v_\pi(s)$ as $N(s) \to \infty$

Blackjack Example

- Objective: Have your card sum be greater than the dealer's without exceeding 21.
- States (200 of them):
    - current sum (12-21)
    - dealer's showing card (ace-10)
    - do I have a useable ace?

- Reward: +1 for winning, 0 for a draw, -1 for losing
- Actions: stick (stop receiving cards), hit (receive another card)
- Policy: Stick if my sum is 20 or 21, else hit
- No discounting ($\gamma=1$)

Learned Blackjack State-Value Functions



Incremental Mean

- The mean μ_k of a sequence $x_1 \ldots x_k$ can be computed incrementally:

$$\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\
&= \frac{1}{k}\left(x_k + (k-1)\mu_{k-1}\right) \\
&= \mu_{k-1} + \frac{1}{k}\left(x_k - \mu_{k-1}\right)
\end{aligned}$$

Incremental Monte Carlo Updates

- Update V(s) incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state S_t with return G_t :

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

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

Monte Carlo Prediction

- Update V(s) incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state St with return Gt

$$N(S_t) \leftarrow N(S_t) + 1$$

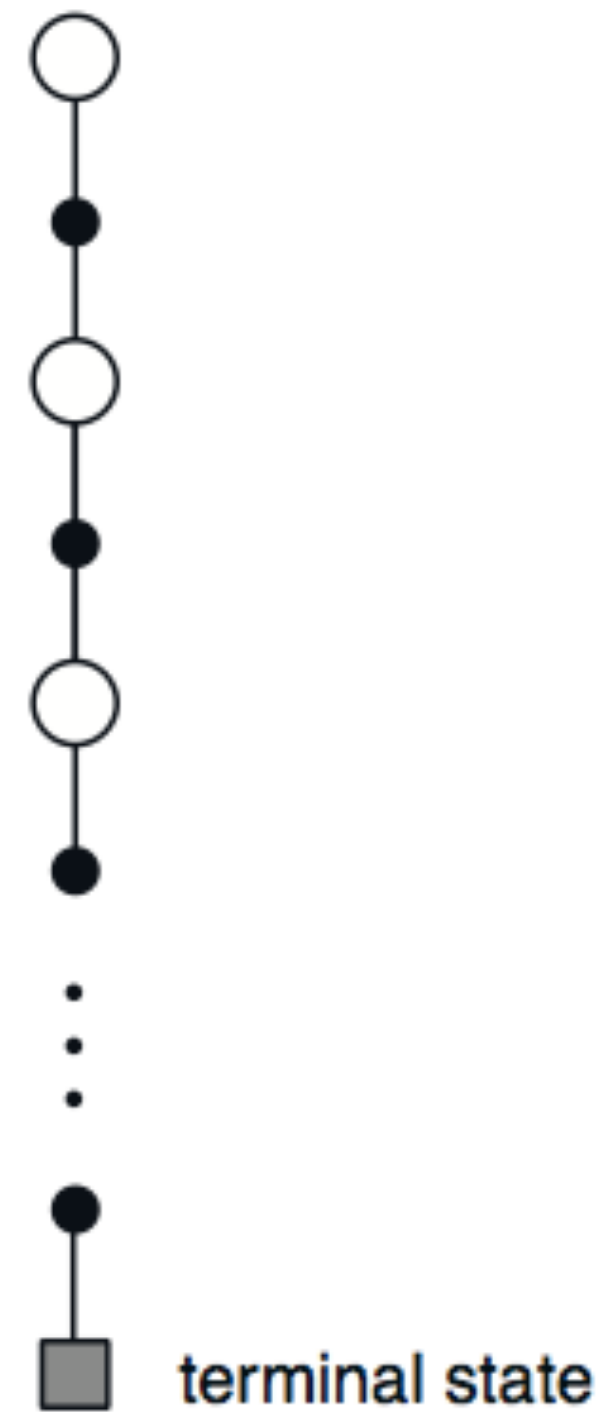
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}\left(G_t - V(S_t)\right)$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

Backup Diagram for Monte Carlo

- Entire rest of episode included
- **Only one choice considered at each state (unlike DP)**
- Does not bootstrap from successor state's values (unlike DP), i.e., the value estimates of later states are not used to inform the values of nearby states.
- Value is estimated by mean return.

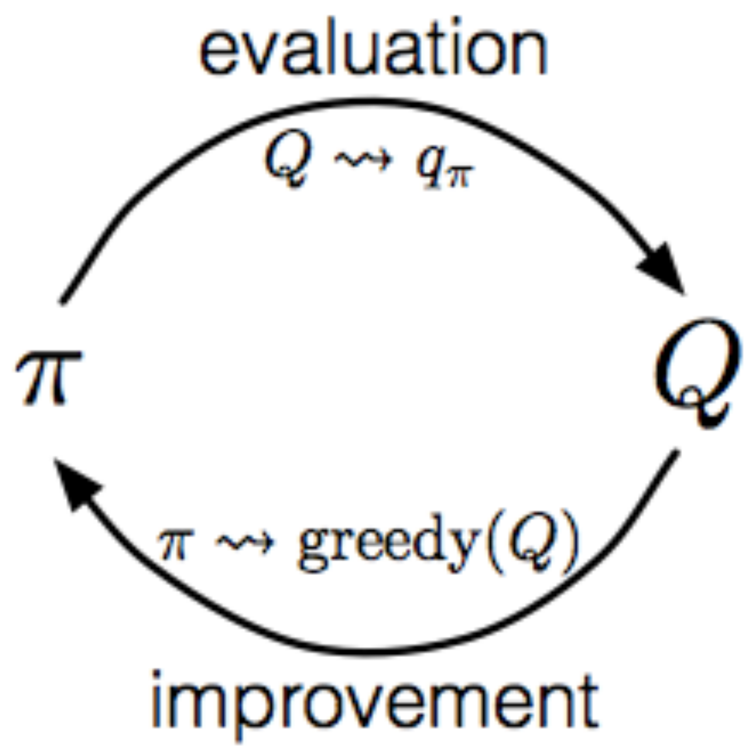


Summary so far

- Unknown dynamics: estimate value functions and optimal policies using Monte Carlo
    - **Monte Carlo Prediction**: estimate the value function of a given policy by deploying it, collect episodes and average their returns.
    - Next: **Monte Carlo control**: find optimal policies by interaction

Monte-Carlo Control

$$\pi_0 \xrightarrow{\mathrm{E}} q_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} q_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} q_*$$



- MC policy iteration step: Policy evaluation using MC methods followed by policy improvement

- Policy improvement step: greedify with respect to value (or action-value) function

Greedy Policy

- For any action-value function q, the corresponding greedy policy is the one that:
  - For each s, deterministically chooses an action with maximal action-value:

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

- Policy improvement then can be done by constructing each π_{k+1} as the greedy policy with respect to $q_{\pi,k}$.

MC Estimation of Action Values (Q)

- Monte Carlo (MC) is most useful when a model is not available
  - We want to learn $q*(s, a)$ because then we can get an optimal policy without knowing dynamics.
- $q_\pi(s, a)$ - average return starting from state s and action a following π

$$
\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big].
\end{aligned}
$$

- Converges asymptotically if every state-action pair is visited.
  - Q: Is this possible if we are using a deterministic policy?

The Exploration problem

- If we always follow the deterministic policy to collect experience, we will never have the opportunity to see and evaluate (estimate q) of alternative actions…
- **ALL learning methods face a dilemma:** they seek to learn action values conditioned on subsequent optimal behaviour but they need to act suboptimally in order to explore all actions (to discover the optimal actions). The exploration-exploitation dilemma.
- Q: Does a learning algorithm know when the optimal policy has been reached to stop exploring?

The Exploration problem

- If we always follow the deterministic policy to collect experience, we will never have the opportunity to see and evaluate (estimate q) of alternative actions…

- **ALL learning methods face a dilemma**: they seek to learn action values conditioned on subsequent optimal behaviour but they need to act suboptimally in order to explore all actions (to discover the optimal actions). The exploration-exploitation dilemma.

- Solutions:

  1. exploring starts: Every state-action pair has a non-zero probability of being the starting pair

  2. Give up on deterministic policies and only search over ϵ-soft policies

  3. Off-policy: use a different policy to collect experience than the one you care to evaluate

Monte Carlo Exploring Starts

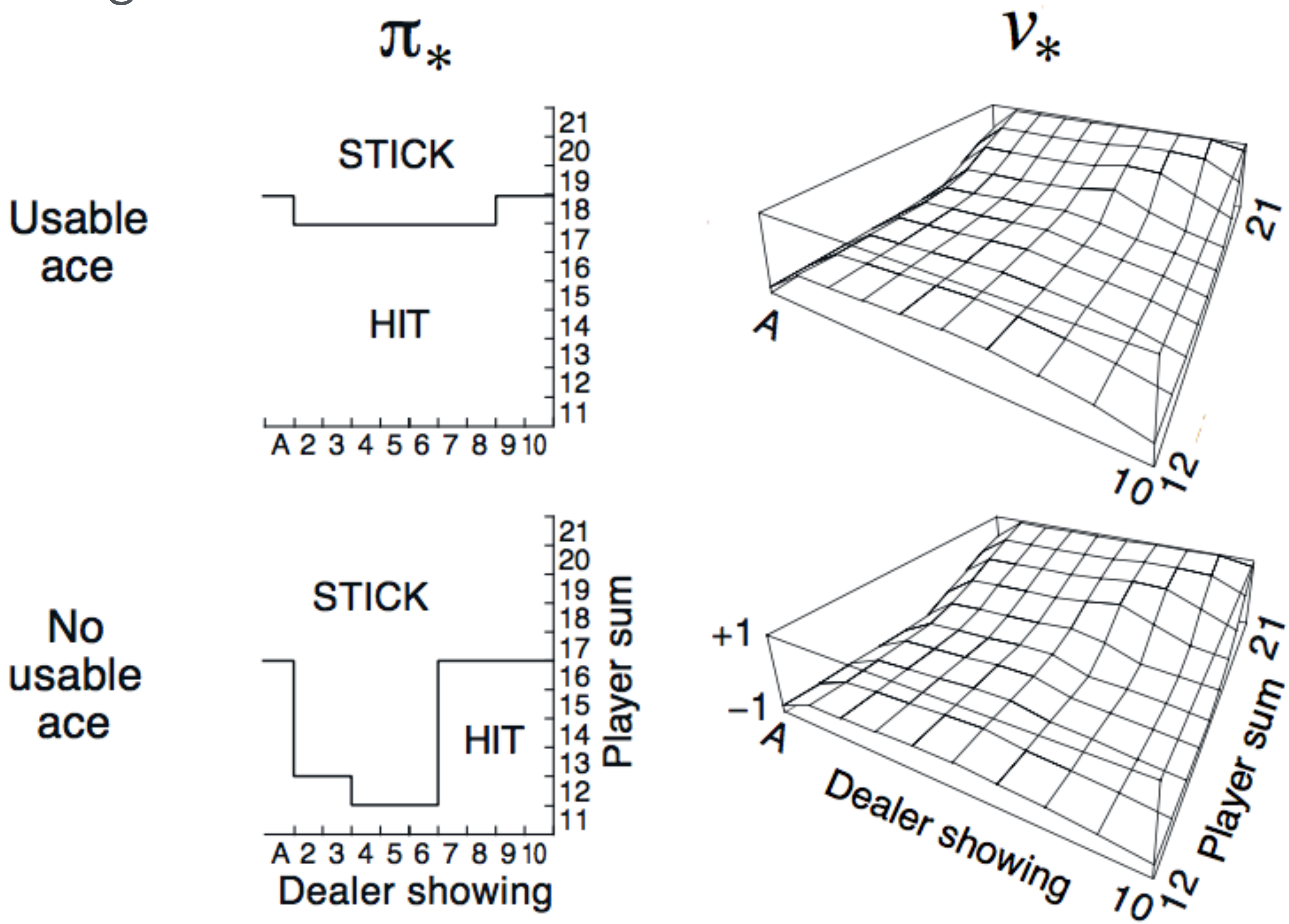
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s, a) \leftarrow$ arbitrary
    $\pi(s) \leftarrow$ arbitrary
    $Returns(s, a) \leftarrow$ empty list

Fixed point is optimal policy π^*

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

Blackjack example continued

- With exploring starts



Convergence of MC Control

- Greedified policy meets the conditions for policy improvement:

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

- And thus must be ≥ π_k.

- This assumes exploring starts and infinite number of episodes for MC policy evaluation

On-policy Monte Carlo Control

- On-policy: learn about policy currently executing
- How do we get rid of exploring starts?
  - The policy must be eternally soft: $\pi(a \mid s) > 0$ for all s and a.
- For example, for ε-soft policy, probability of an action, π(a|s),

$$= \underset{\text{non-max}}{\frac{\epsilon}{|\mathcal{A}(s)|}} \quad \text{or} \quad \underset{\text{max (greedy)}}{1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}}$$

- Similar to GPI: move policy towards greedy policy
- Converges to the best ε-soft policy.

$\epsilon -$ soft Policies

- They keep choosing suboptimal actions even when the best one has been discovered.
- The second best action is as bad as the worst action.
- However, we will stick with them till we figure out better exploration methods later in the course.

On-policy Monte Carlo Control

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

- Learn the value of the target policy π from experience due to behavior policy μ.
- For example, π is the greedy policy (and ultimately the optimal policy) while μ is exploratory (e.g., ε-soft) policy
- In general, we only require coverage, i.e., that μ generates behavior that covers, or includes, π:

$$\mu(a|s) > 0 \quad \text{for every } s,a \text{ at which} \quad \pi(a|s) > 0$$

- Q: can I average returns as before to obtain the value function of π ?

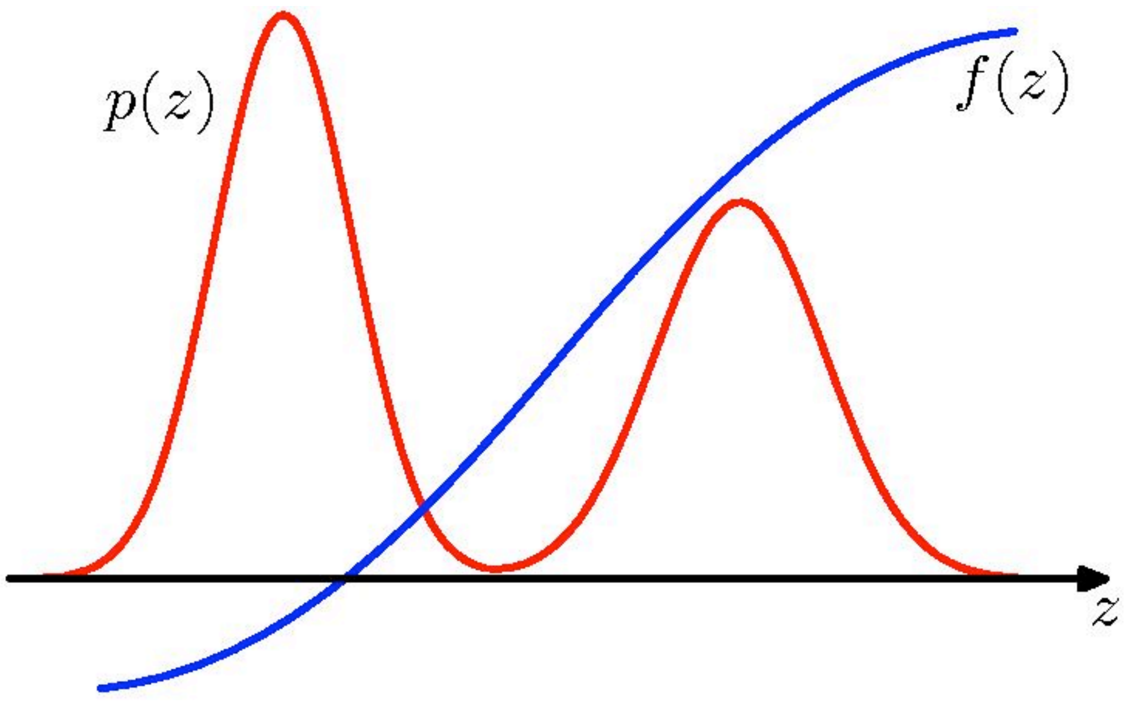
Off-policy methods

- Learn the value of the target policy π from experience due to behavior policy μ.
- For example, π is the greedy policy (and ultimately the optimal policy) while μ is exploratory (e.g., ε-soft) policy
- In general, we only require coverage, i.e., that μ generates behavior that covers, or includes, π:

$$\mu(a|s) > 0 \quad \text{for every } s,a \text{ at which} \quad \pi(a|s) > 0$$

- Idea: Importance Sampling:
    - Weight each return by the ratio of the probabilities of the trajectory under the two policies.

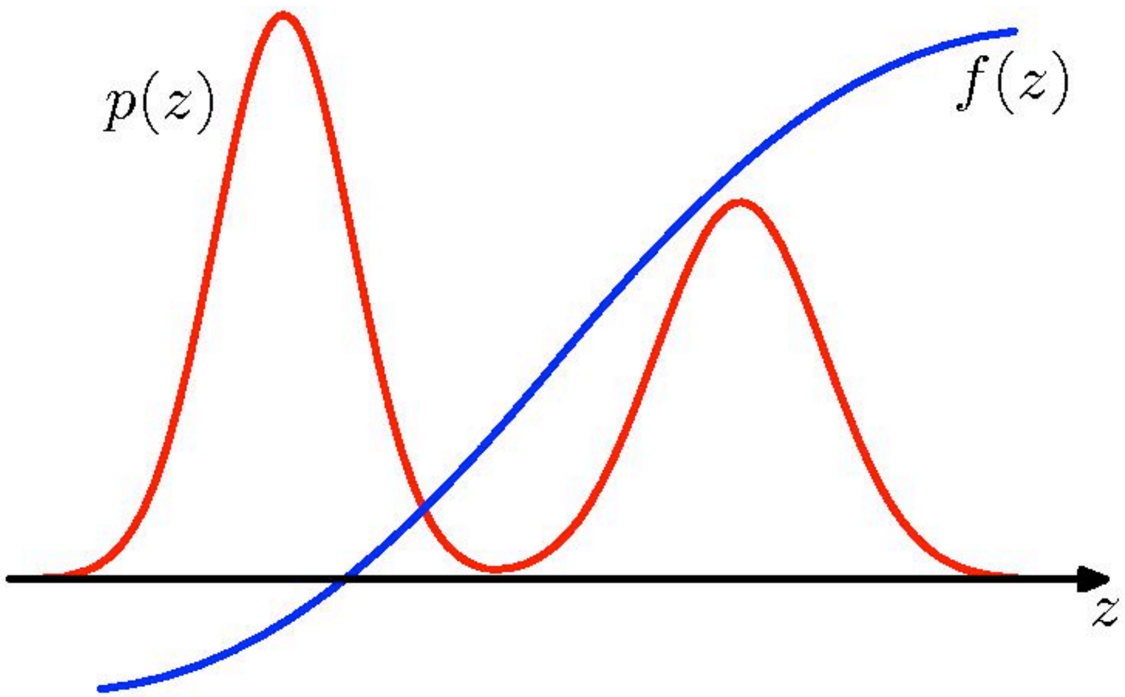
Expectations



$$\mathbb{E}[f] = \int f(z)p(z)dz$$

Estimating Expectations

- General Idea: Draw independent samples $\{z^1, \ldots, z^n\}$ from distribution p(z) to approximate expectation:

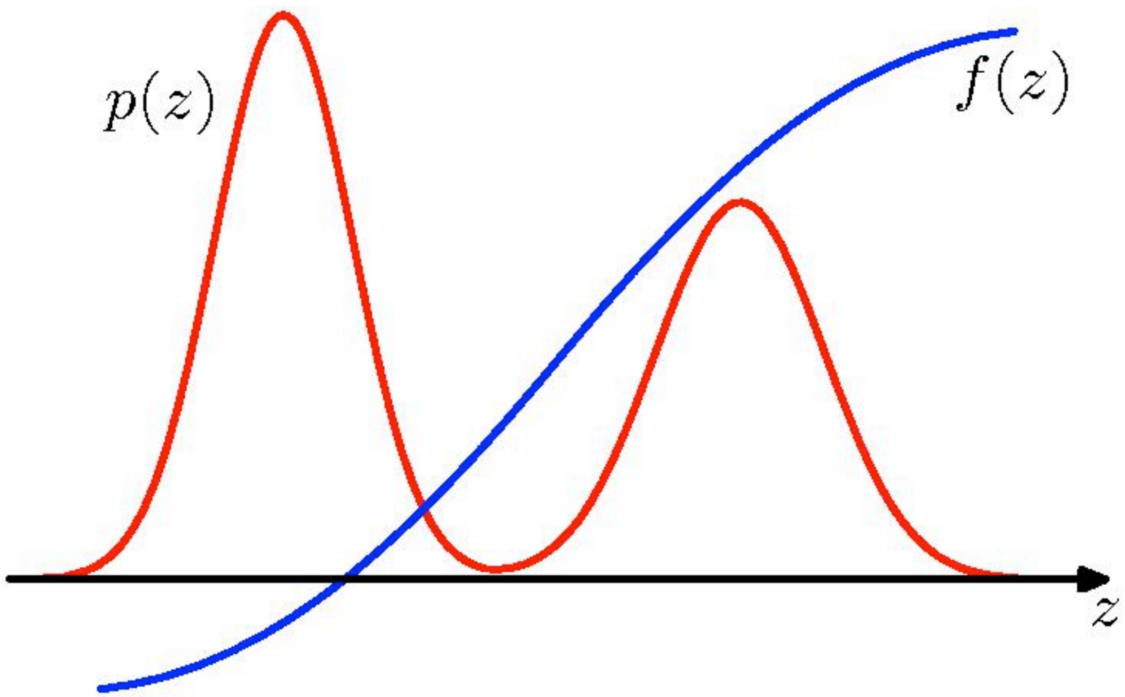


$$\mathbb{E}[f] = \int f(z)p(z)dz \approx$$

$$\frac{1}{N}\sum_{n=1}^{N} f(z^n) = \hat{f}.$$

Estimating Expectations

- General Idea: Draw independent samples $\{z^1, \ldots, z^n\}$ from distribution p(z) to approximate expectation:



$$\mathbb{E}[f] = \int f(z)p(z)dz \approx$$

$$\frac{1}{N}\sum_{n=1}^{N} f(z^n) = \hat{f}.$$

Note that:

$$\mathbb{E}[f] = \mathbb{E}[\hat{f}].$$

so the estimator has correct mean (unbiased).

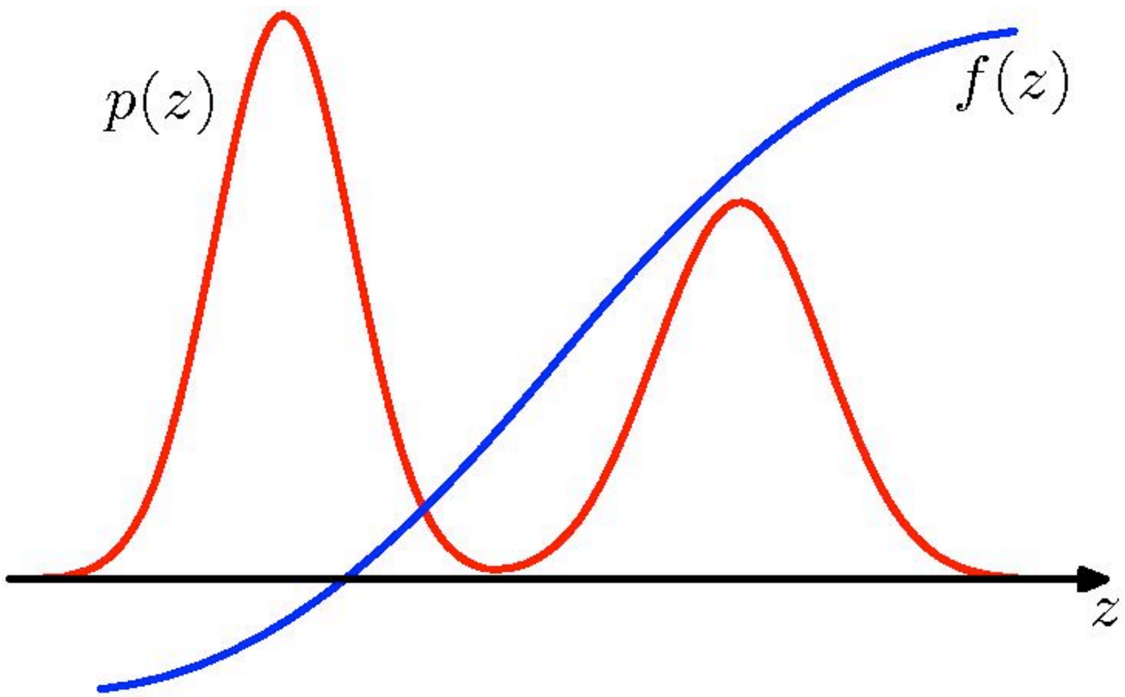
- The variance:

$$\text{var}[\hat{f}] = \frac{1}{N}\mathbb{E}\left[(f - \mathbb{E}[f])^2\right].$$

- Variance decreases as 1/N.

Estimating Expectations

- General Idea: Draw independent samples $\{z^1, \ldots, z^n\}$ from distribution p(z) to approximate expectation:



$$\mathbb{E}[f] = \int f(z)p(z)dz \approx \frac{1}{N}\sum_{n=1}^{N} f(z^n) = \hat{f}.$$

Note that:

$$\mathbb{E}[f] = \mathbb{E}[\hat{f}].$$

so the estimator has correct mean (unbiased).

- The variance:

$$\mathrm{var}[\hat{f}] = \frac{1}{N}\mathbb{E}\left[(f - \mathbb{E}[f])^2\right].$$

- Variance decreases as 1/N.

- **Remark**: The accuracy of the estimator does not depend on dimensionality of z.

Importance Sampling

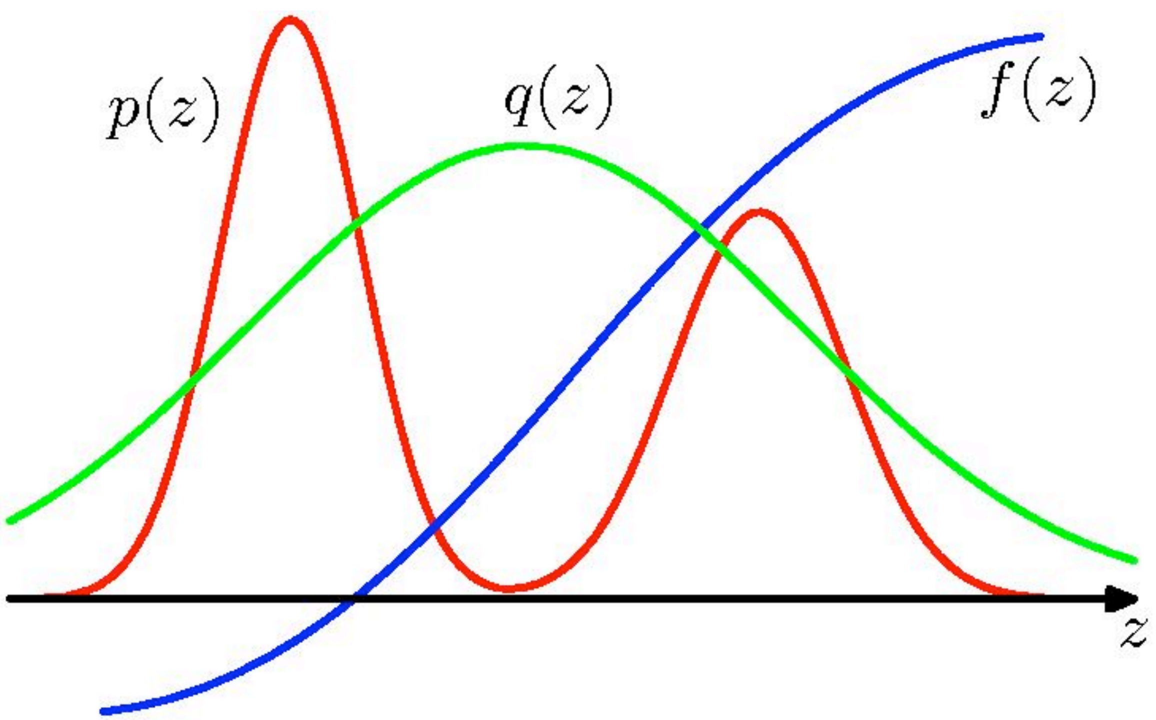
- Suppose we have an easy-to-sample proposal distribution q(z), such that

$$q(z) > 0 \;\; \text{if} \;\; p(z) > 0.$$

$$\mathbb{E}[f] = \int f(z)p(z)dz$$

$$= \int f(z)\frac{p(z)}{q(z)}q(z)dz$$

$$\approx \frac{1}{N}\sum_n \frac{p(z^n)}{q(z^n)} f(z^n), \;\; z^n \sim q(z).$$



- The quantities

$$w^n = p(z^n)/q(z^n)$$

are known as importance weights.

- This is useful when we can evaluate the probability p but is hard to sample from it

Importance Sampling Ratio

- Probability of the rest of the trajectory, after S_t, under policy π:

$$
\begin{aligned}
\Pr\{A_t, S_{t+1}, A_{t+1}, \ldots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\
&= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\
&= \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),
\end{aligned}
$$

- Importance Sampling: Each return is weighted by the relative probability of the trajectory under the target and behavior policies

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

- This is called the Importance Sampling Ratio

Importance Sampling

- Ordinary importance sampling forms estimate

First time of termination following time t

return after t up through T(t)

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}.$$

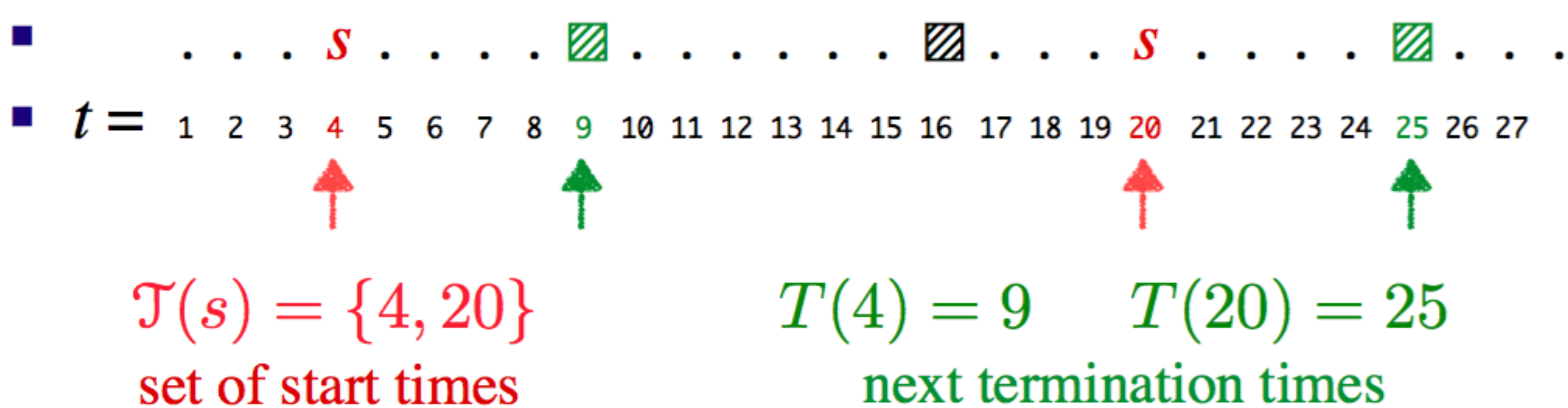
Every time: the set of all time steps in which state s is visited

Importance Sampling

- Ordinary importance sampling forms estimate

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}.$$

- New notation: time steps increase across episode boundaries:



Importance Sampling Ratio

- All importance sampling ratios have expected value 1:

$$\mathbb{E}_{A_k \sim \mu}\left[\frac{\pi(A_k|S_k)}{\mu(A_k|S_k)}\right] = \sum_a \mu(a|S_k)\frac{\pi(a|S_k)}{\mu(a|S_k)} = \sum_a \pi(a|S_k) = 1.$$

- Note: Importance Sampling can have high (or infinite) variance.

Importance Sampling

- Two ways of averaging weighted returns:
  - Ordinary importance sampling forms estimate:

$$V(s) \doteq \frac{\sum_{t \in \mathfrak{T}(s)} \rho_t^{T(t)} G_t}{|\mathfrak{T}(s)|}.$$

  - Weighted importance sampling forms estimate:

$$V(s) \doteq \frac{\sum_{t \in \mathfrak{T}(s)} \rho_t^{T(t)} G_t}{\sum_{t \in \mathfrak{T}(s)} \rho_t^{T(t)}}$$

So far

- MC has several advantages over DP:
  - Can learn directly from interaction with environment
  - No need for full models

- MC methods provide an alternate policy evaluation process
- One issue to watch for: maintaining sufficient exploration

- Looked at distinction between on-policy and off-policy methods
- Looked at importance sampling for off-policy learning
- Looked at distinction between ordinary and weighted IS

Coming up next

- MC methods are different than Dynamic Programming in that they:
  1. use experience in place of known dynamics and reward functions
  2. do not bootstrap
- Next lecture we will see temporal difference learning which
  3. use experience in place of known dynamics and reward functions
  4. bootstrap!