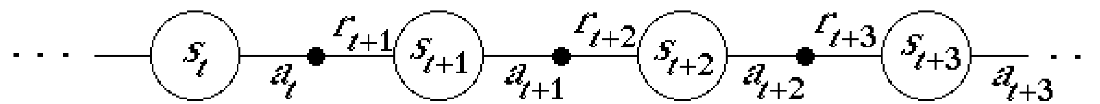
COMP 652: Machine Learning

Lecture 18

Today

- Policy value
- Value functions
- Monte Carlo methods for estimating policy value / value functions
- Policy iteration

Recall: Markov Decision Process (MDP)



An MDP is defined by:

- Set of *states* S
- Set of *actions* $A(s)$ available in each state s
- *Rewards*:

$$r_{ss'}^a = E\left\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\right\}$$

- *Transition probabilities*

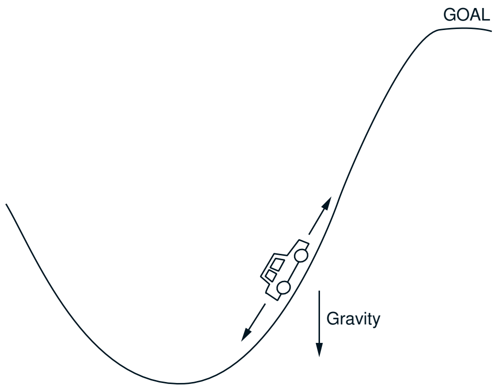
$$p_{ss'}^a = P\left(s_{t+1} = s' | s_t = s, a_t = a\right)$$

Recall: Agent's Learning Task

- Assuming no prior knowledge of $r^a_{ss'}$, $p^a_{ss'}$ learn/select a *policy* (strategy, way of behaving) $\pi : S \times A \rightarrow [0,1]$, where $\pi(s,a)$ is probability the agent chooses action a when environment is in state s.

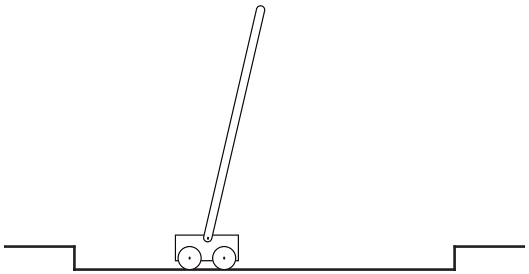
- π should maximize some notion of return (long-term reward):
  - $R = r_1 + r_2 + \ldots + r_T$, where task ends on step T.
  - $R = r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots$, where $\gamma < 1$ is a discount rate.
  - $R = \lim_{T\rightarrow\infty} \frac{1}{T}(r_1 + r_2 + \ldots + r_T)$

Example: Mountain-Car



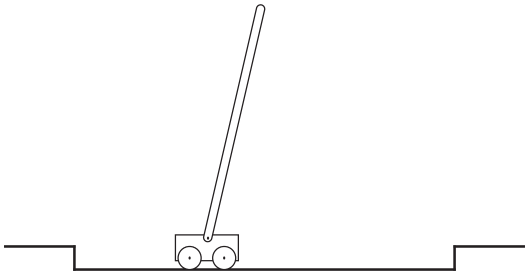
- States: position and velocity
- Actions: accelerate forward, accelerate backward, coast
  (It is assumed the engine is weak!)
- Two reward formulations:
  - reward $= -1$ for every time step, until car reaches the top
  - reward $= 1$ at the top, 0 otherwise $\gamma < 1$
- In both cases, the return is maximized by minimizing the number of steps to the top of the hill

Example: Pole Balancing



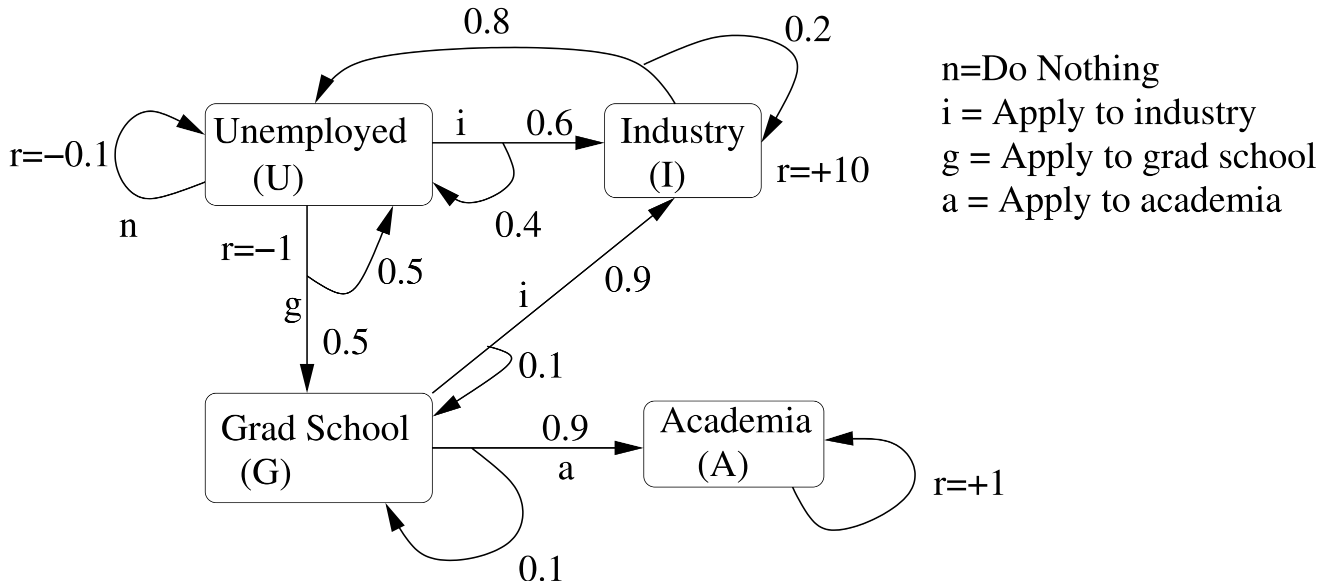
- States: Position of cart, angle and angular velocity of pole.
- Actions: Turn wheels left, turn wheels right, stay still
- Goal: prevent pole falling beyond a given angle, or cart hitting the end of the track
- Rewards?

Example: Pole Balancing



- States: Position of cart, angle and angular velocity of pole.
- Actions: Turn wheels left, turn wheels right, stay still
- Goal: prevent pole falling beyond a given angle, or cart hitting the end of the track
- Rewards:
  - Episodic task formulation: reward $= +1$ for each step before failure
    $\Rightarrow$ return $=$ number of steps before failure
  - Continuing task formulation: reward $=$ -1 upon failure, 0 otherwise, $\gamma < 1$
    $\Rightarrow$ return $= -\gamma^k$ if there are k steps before failure

Graduate school example



⇒ In any of these cases, what is the best policy? How can we learn it from experience (assuming we have no model of the world)?

Finding a good policy

- The problem seems difficult to solve even for toy examples
- Since we do not have expert-labeled examples, ideas for supervised learning do not apply immediately.
- One way to address the problem is to use *search for a good policy*, in the space of all possible policies
- To do this, we need a *measure of the quality* of a policy

Some simplifying assumptions

- Suppose the task is episodic – either always ending at a present time T, or ending whenever some terminal state is reached
- Suppose the environment always starts in the same state
- Suppose we have the ability to attempt the task as many times as we want (starting the task, and choosing actions until termination, observe states and rewards along the way)
- Suppose there is a finite set of policies $\pi_1, \ldots, \pi_m$ from which we want to find the one that maximizes expected return

$\Rightarrow$ How can we solve this problem?

(Aside: Where might there be a finite set of policies of interest?)

One solution

- Treat as an m-armed bandit problem!
- On each attempt of the problem, we choose one policy π to follow ($\equiv$ which arm to pull)
- Follow the policy until the task ends
- The return generated is one sample of the expected return of that policy ($\equiv$ the arm's expected reward)

Example: Walking a hall

+10					S					-5

- 11-state MDP.
- Start state in the middle.
- Terminal states at either end, with rewards shown. All other rewards are zero.
- Two actions: Left, Right. However, with 20% chance, the action takes you in the opposite direction.
- Returns are undiscounted: $R = r_1 + r_2 + \ldots + r_T$.
- Consider three policies:
  - Left: Always go left
  - Right: Always go right
  - Rand: Go left with probability 0.5, otherwise right
- Trying each policy 10,000 times yields: (??)

Example: Walking a hall

+10					S					-5

- 11-state MDP.
- Start state in the middle.
- Terminal states at either end, with rewards shown. All other rewards are zero.
- Two actions: Left, Right. However, with 20% chance, the action takes you in the opposite direction.
- Returns are undiscounted: $R = r_1 + r_2 + \ldots + r_T$.
- Consider three policies:
  - Left: Always go left
  - Right: Always go right
  - Rand: Go left with probability 0.5, otherwise right
- Trying each policy 10,000 times yields:

$$E_{Left}\{R\} \approx 9.991 \qquad E_{Rand}\{R\} \approx 2.4460 \qquad E_{Right}\{R\} \approx -4.991$$

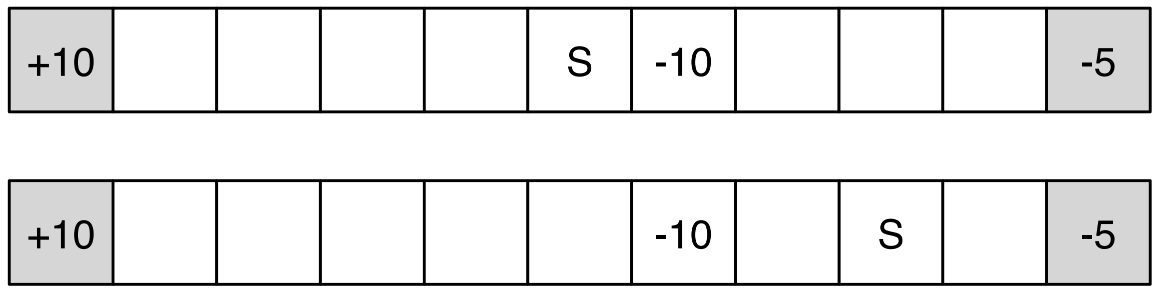
Comments?

Comments?

- How do we come up with policies to try?
- Can we try all policies? (If S and $A(s)$ finite, the number of possible deterministic policies is finite—but probably very large!)
- What if the start state were different?
- Should we change policy during a trajectory?

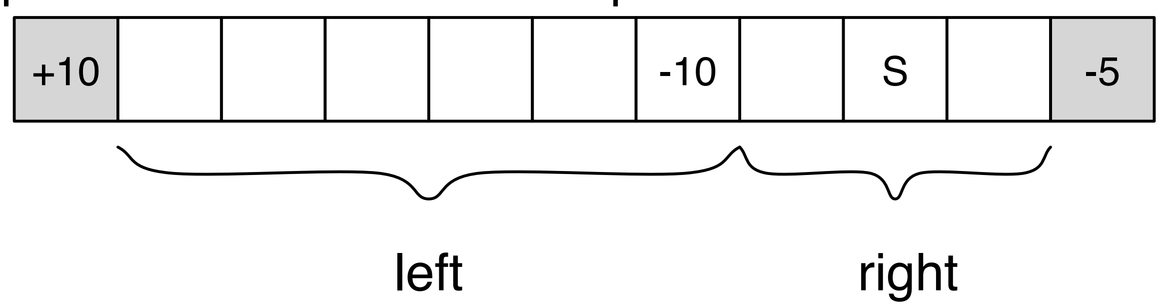
Examples and comments

- Optimal policy can depend on the start state:



- More generally, which policy is best to follow may change during performance of the task (e.g., if agent gets to an unexpected state)

- Of course, policies are allowed to depend on the state!



State Value Function

- The **<u>value of a state s</u>** under policy π is the expected return when starting from s and choosing actions according to π:

$$V^{\pi}(s) = E_{\pi}\{R_0 \mid s_0 = s\} = E_{\pi}\left\{\sum_{k=1}^{\infty} \gamma^{k-1} r_k \mid s_0 = s\right\}$$

- This is for $\gamma \in (0, 1]$. For $\gamma = 1$, we must additionally assume termination with probability one under any π.
- If the state space is finite, the collection of values of all states, V^{π}, can be represented as a vector of size equal to the number of states.
- This vector is called the **<u>state-value function</u>**

State-action value function

- Analogously, the **value of taking action a in state s** under policy π is:

$$Q^\pi(s,a) = E_\pi\left\{\sum_{k=1}^{\infty} \gamma^{k-1} r_k \mid s_0 = s, a_0 = a\right\}$$

- It is the expected return if we start by taking action a, and follow π thereafter.
- Q^π can be represented as a matrix of size $|S| \times |A|$; this is called the **action-value function** (or **state-action-value function**)

Policies and value functions

- Value functions define a *partial order over policies*:

$$\pi_1 \geq \pi_2 \text{ if and only if } V^{\pi_1}(s) \geq V^{\pi_2}(s) \forall s \in S$$

- So a policy is "better" than another policy if and only if it generates at least the same amount of return *at all states*
- If π_1 has higher value than π_2 at some states and lower value at other, the two policies are not comparable.
- Computing the value of a policy will be helpful in searching for it.

How to estimate these functions?

- For a given policy π, how can we estimate the state-value function $V^\pi(s)$?
- How can we estimate the action-value function $Q^\pi(s, a)$?

Monte Carlo Method for $V^\pi(s)$ estimation

- Suppose we have an episodic task
- The agent creates multiple trajectories following policy π
- Compute $V^\pi(s)$ by *averaging the observed returns* after s on the trajectories in which s was visited.
- Two main approaches:
  - **Every-visit:** average returns for every time a state is visited in an episode
  - **First-visit:** average returns only for the first time a state is visited in an episode

Example

+10					S					-5

- Monte Carlo $V^{\pi}(s)$ estimation
- 100,000 trajectories each, for the
- Left, Rand, and Right policies
- Returns from first visits only used in estimates

s	1	2	3	4	5	6	7	8	9	10	11
$V^{Left}(s)$	0	9.9997	9.9995	9.9985	9.9962	9.9837	9.9341	9.7357	8.9465	5.9125	0
# Samps	99891	99893	99894	99901	99916	100000	24816	6186	1552	400	109

s	1	2	3	4	5	6	7	8	9	10	11
$V^{Rand}(s)$	0	8.5034	6.9851	5.5001	3.9941	2.5008	0.9956	-0.5129	-1.9963	-3.5005	0
# Samps	50005	55547	62584	71435	83396	100000	83284	71334	62513	55548	49995

s	1	2	3	4	5	6	7	8	9	10	11
$V^{Right}(s)$	0	-0.9125	-3.9465	-4.7357	-4.9341	-4.9836	-4.9962	-4.9985	-4.9995	-4.9997	0
# Samps	109	400	1552	6186	24816	100000	99916	99901	99894	99893	99891

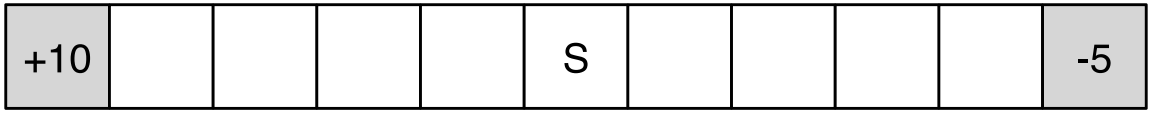
How to estimate these functions?

- □ For a given policy π, how can we estimate the state-value function $V^\pi(s)$?
- $\Rightarrow$ How can we estimate the action-value function $Q^\pi(s, a)$?

Monte Carlo $Q^\pi(s,a)$ estimation

- Again, assume episodic task
- Need to generate samples of return following π, from every (s,a) pair
- Can simply follow policy π, as in Monte Carlo V^π
- Use first-visit or every-visit observed after each distinct (s,a) pair to estimate returns.
- However, if π never chooses some actions, there values will not be learned. Ways to accommodate that include:
  - If have simulator, can start in any state s and choose initial action a randomly, then follow π afterwards
    This is called *Monte Carlo [with] exploring starts*
  - Occassionaly ignore π (say once per trajectory, or with some small probability) and choose a different a

Example



- Monte Carlo $Q^{\pi}(s,a)$ estimation
- 100,000 "exploring starts" trajectories – initial (s,a) uniformly random; π followed aftewards
- Only return from first state-action pair used to estimate Q^{π}
  Thus, approx 100,000/18 or 5555 updates per (s,a) pair.

s	2	3	4	5	6	7	8	9	10
$Q^{Left}(s, Left)$	10.00	10.00	10.00	9.99	9.98	9.95	9.73	9.06	6.21
$Q^{Left}(s, Right)$	10.00	10.00	10.00	9.99	9.97	9.80	9.26	6.91	-2.25

s	2	3	4	5	6	7	8	9	10
$Q^{Rand}(s, Left)$	9.40	7.80	6.44	4.88	3.50	1.77	0.44	-1.08	-2.69
$Q^{Rand}(s, Right)$	7.63	6.13	4.71	3.08	1.46	0.26	-1.46	-2.80	-4.42

s	2	3	4	5	6	7	8	9	10
$Q^{Right}(s, Left)$	7.24	-1.94	-4.22	-4.82	-4.96	-4.99	-4.99	-5.00	-5.00
$Q^{Right}(s, Right)$	-1.16	-4.08	-4.75	-4.96	-4.99	-4.99	-5.00	-5.00	-5.00

Searching for / determining a good policy

How can we use $V^\pi(s)$ or $Q^\pi(s, a)$ to aid our search for a good (or better yet, optimal) policy?

Searching for / determining a good policy

How can we use $V^{\pi}(s)$ or $Q^{\pi}(s,a)$ to aid our search for a good (or better yet, optimal) policy?

- If $V^{\pi_1}(s) \geq V^{\pi_2}(s)$ for all s, then $\pi_1 \geq \pi_2$, and π_2 can be discarded. (This can sometimes be useful. For example, in the previous example it tells is the Left policy is better than Rand or RIght.)

Searching for / determining a good policy

How can we use $V^{\pi}(s)$ or $Q^{\pi}(s,a)$ to aid our search for a good (or better yet, optimal) policy?

- □ If $V^{\pi_1}(s) \geq V^{\pi_2}(s)$ for all s, then $\pi_1 \geq \pi_2$, and π_2 can be discarded. (This can sometimes be useful. For example, in the previous example it tells is the Left policy is better than Rand or RIght.)

- □ If $Q^{\pi}(s,a) > Q^{\pi}(s,\pi(s))$, then we can change $\pi(s)$ to a, and the resulting policy is better!
(Proof omitted.)

Policy iteration

- Start with any initial policy π
- Repeat the following, until π stops changing:
  1. Policy evaluation: Determine $Q^{\pi}(s, a)$
  2. Policy improvement: $\pi(s) = \arg\max_{a \in A(s)} Q^{\pi}(s, a)$ for all s
     (Also called "greedifying")

Example

+10					S		-3			-5

- 100,000 Monte Carlo exploring starts for $Q^{\pi}(s,a)$ estimation

s	2	3	4	5	6	7	8	9	10
π	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$
$Q^{\pi}(s, Left)$	6.3394	-5.8731	-8.8477	-9.7242	-9.8884	-9.8891	-9.2257	-8.9869	-5.946
$Q^{\pi}(s, Right)$	-4.865	-8.7399	-9.6257	-9.8612	-9.878	-9.9618	-6.9695	-6.1738	-5.2483
π	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longrightarrow$	$\longleftarrow$	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$
$Q^{\pi}(s, Left)$	9.9854	9.8815	9.4107	7.8371	5.9691	-0.80517	-2.0726	-7.1307	-5.5093
$Q^{\pi}(s, Right)$	9.8914	9.5221	8.0489	2.7912	0.94291	-6.1247	-4.7915	-5.5325	-5.1099
π	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longrightarrow$	$\longrightarrow$
$Q^{\pi}(s, Left)$	9.9978	9.9995	9.9765	9.8761	9.5511	8.2702	5.9845	1.425	-3.5054
$Q^{\pi}(s, Right)$	9.9876	9.9881	9.9144	9.654	8.6529	4.3082	-0.81565	-3.1705	-4.6068
π	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$
$Q^{\pi}(s, Left)$	9.9978	9.9995	9.9797	9.911	9.6749	8.7433	7.7998	4.3559	2.4591
$Q^{\pi}(s, Right)$	9.9903	9.9897	9.9362	9.7511	9.0253	5.7833	5.2101	2.8932	-3.15
π	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$

Questions

- Does policy iteration always terminate?
- If so, how after how many iterations?
- Does it find a locally optimal deterministic policy?
- Does it find a globally optimal deterministic policy?
- Does it find a globally optimal policy? (I.e., allowing stochastic policies as well?)

Answers

- Yes, it always terminates.
  - It generates a sequence of deterministic policies π_i such that $V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s)$.
  - There are finitely many deterministic policies, so it must terminate.
  - Often the number of iterations is much smaller than the total number of possible policies.
- It finds globally optimal policies.
  - It turns out that in any suboptimal policy, there is always a single action change that improves the policy.
  - It turns out that stochastic policies cannot outperform deterministic ones – for this version of the problem.
- $\Rightarrow$ Policy iteraction, with Monte Carlo action-value estimation, is thus a complete solution to the reinforcement learning problem!
  (There are many other solutions, however.)

Drawbacks of MCES Policy Iteration?

Drawbacks of MCES Policy Iteration

- How many samples of the return for each (s, a) pair do we need? Infinitely many?
- We're not trying to optimize performance during learning
- Isn't a lot of information being ignored?
- What if any of our various simplifying assumptions are violated: can start in any state, task terminates under any policy, state set is finite and reasonably small, ...

Representing value functions using function approximation

- If the state and action spaces are finite, $Q^\pi(s,a)$ can be represented explicitly as a table
- If the state space (or action space) is infinite (or just inconveniently large):
  - Define a feature mapping $\Phi : S \mapsto \Re^n$
  - Use your favorite function approximator that can represent real-values functions: linear, polynomial, nearest neighbor, neural network, regression tree, SVM, etc.
- This helps represent Q^π. But how do we represent π istelf?