Introduction to Reinforcement Learning

Shimon Whiteson
Dept. of Computer Science
University of Oxford

(based on material from
Rich Sutton and Andrew Barto)

September 2, 2019

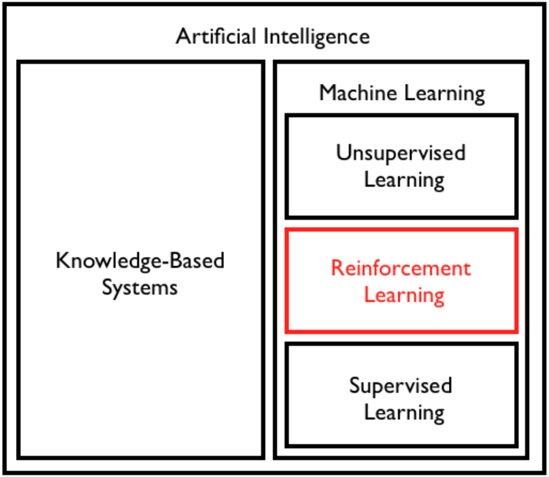
Course Outline

- **Monday: Tabular Methods**
  - MDPs & value functions
  - Tabular planning and model-free RL
- **Tuesday: Function Approximation & Deep RL**
  - Semi-gradient evaluation & control
  - Linear function approximation & LSTD
  - Deep RL
- **Wednesday: Policy Gradients & Model-Based RL**
  - Policy gradients, actor-critic, natural gradients
  - Tabular & deep model-based methods
  - Smart exploration with models

Reinforcement learning

How can an intelligent *agent* learn from experience how to make decisions that maximise its *utility* in the face of *uncertainty*?

Artificial intelligence



Reinforcement learning

- In *reinforcement learning* an agent tries to solve a control problem by directly interacting with an unfamiliar environment
- The agent must learn by trial and error, trying out actions to learn about their consequences
- Applicable to robot control, game playing, system optimisation, ad serving, and information retrieval
- Part of machine learning, inspired by behavioural psychology, related to operations research, control theory, classical planning, and aspects of neuroscience

Reinforcement learning vs. supervised learning

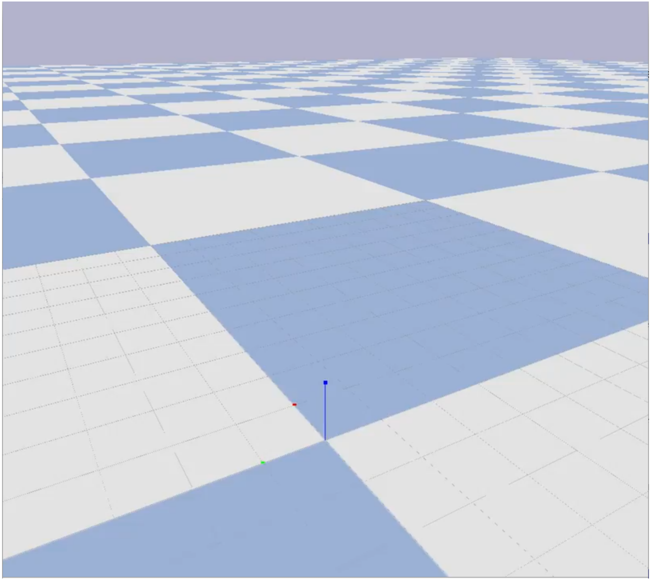
- No examples of correct or incorrect behaviour; instead only *rewards* for actions tried
- The agent is *active* in the learning process: it has partial control over what data it will obtain for learning
- The agent must learn *on-line*: it must maximise performance during learning, not afterwards

Sutton's reward hypothesis

"All of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward)."

Source: `http://rlai.cs.ualberta.ca/RLAI/rewardhypothesis.html`

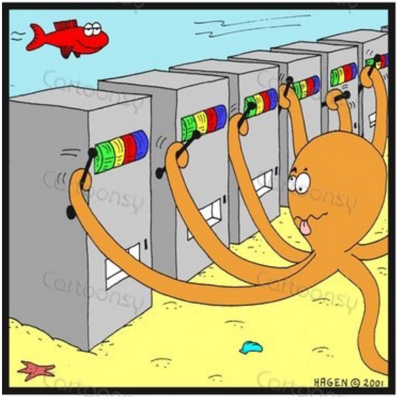
Reward design problem



Source: David Ha, https://goo.gl/W61QgR

K-armed bandit problem

- Sit before a slot machine (bandit) with many arms
- Each arm has an unknown stochastic payoff
- Goal is to maximise cumulative payoff over some period



Compulsive gambling

Formalizing the K-armed bandit problem

- There are K *actions* available at each timestep (also called *plays* or *pulls*)
- After the t-th action, the agent receives reward $r_i \sim R_{a_t}$
- In a *finite-horizon* problem, the agent tries to maximise its total reward over T actions: $\sum_{t=1}^{T} r_t$
- In an *infinite-horizon* problem, the agent tries to maximise its *discounted* total reward: $\sum_{t=0}^{\infty} \gamma^t r_t$ where $\gamma \in [0, 1)$
- $1 - \gamma$ can be interpreted as the probability of the game ending after each step

Exploration and exploitation

- The agent's ability to get reward in the future depends on what it knows about the arms. Thus, it must *explore* the arms in order to learn about them and improve its chances of getting future reward

- But the agent must also use what it already knows in order to maximise its total reward; Thus it must *exploit* by pulling the arms it expects to give the largest rewards

Balancing exploration and exploitation

- The main challenge in a k-armed bandit is how to balance the competing needs of exploration and exploitation
- If the horizon is finite, exploration should decrease as the horizon gets closer
- If the horizon is infinite but $\gamma < 1$, exploration should decrease as the agent's uncertainty about expected rewards goes down
- If the horizon is infinite and $\gamma = 1$, there is an *infinitely delayed splurge*.

Action-value methods

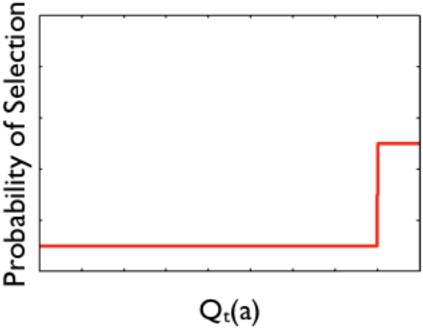
- Based on observed rewards, maintain estimates of the expected value of each arm: $Q_t(a) \simeq E[r_t|a_t]$
- Estimates are based on the sample average; If action a has been chosen k_a times, yielding rewards $r_1, r_2, \ldots, r_{k_a}$, then:

$$Q_t(a) = \frac{\sum_{i=1}^{k_a} r_i}{k_a}$$

- Exploiting means taking the *greedy* action: $a^* = \arg\max_a Q_t(a)$
- Exploring means taking any other action

ϵ-greedy exploration

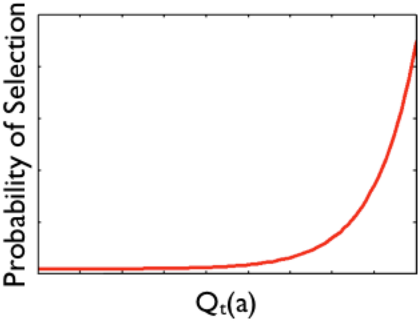
In *ϵ-greedy* exploration, the agent selects a random action with probability ϵ, and the greedy action otherwise



Softmax exploration

In *softmax* exploration, the agent chooses actions according to a *Boltzmann* distribution

$$p(a) = \frac{\exp(Q(a)/\tau)}{\sum_{a'} \exp(Q(a')/\tau)}$$

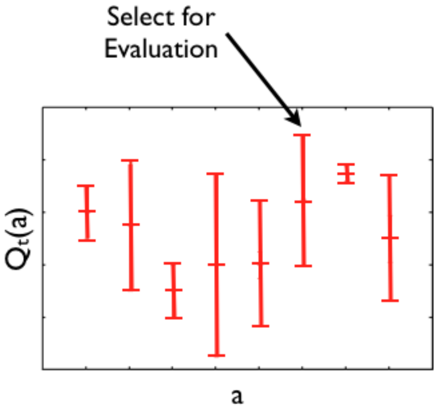


Optimism in the face of uncertainty

- Neither ϵ-greedy nor softmax considers *uncertainty* in action-value estimates
- Goal of exploration is to reduce uncertainty
- So focus exploration on most uncertain actions
- Principle of *optimism in the face of uncertainty*

Upper confidence bound

- Compute *confidence interval* for each arm
- Select arm with largest *upper confidence bound*
- Formally: $a_t = \arg\max_i u_t(a_i)$ where $u_t(a_i) = Q_t(a_i) + c_t(a_i)$
  - $Q_t(a_i)$: action-value estimate
  - $c_t(a_i)$: optimism bonus
- Defining optimism bonus:
  - $c_t(a_i) = \sqrt{\frac{\alpha \ln t}{N_t(a_i)}}$
  - Decreases with # pulls of a_i
  - Increases with t

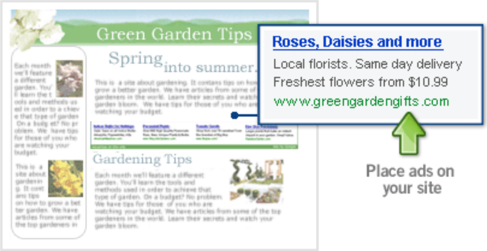


Contextual bandit problem

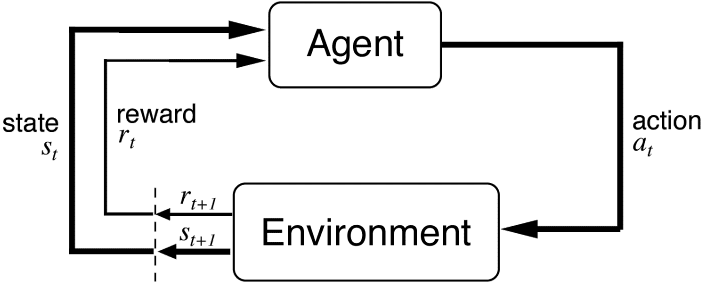
- Also called *associative search*
- At each play, agent receives a *state signal*, also called an *observation* or *side-information*
- Expected payoffs depend on that observation
- Suppose there are many bandits, each a different color; after each play, you are randomly transported to another bandit
- In principle, can be treated as multiple simultaneous bandit problems and estimate $Q(s, a) = \mathbb{E}[R|s, a]$

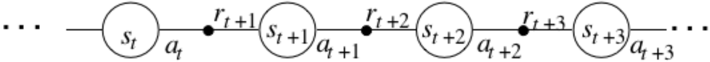
Ad placement

- Web page = state
- Actions = ads
- Environment = user
- Reward = pay per click



The full reinforcement learning problem





The credit-assignment problem

Suppose an agent takes a long sequence of actions, at the end of which it receives a large positive reward?

How can it determine to what degree each action in that sequence deserves *credit* for the resulting reward?

Richard Bellman

- Father of decision-theoretic planning
- Formalized Markov decision processes, derived Bellman equation, invented dynamic programming
- "A towering figure among the contributors to modern control theory and systems analysis" -IEEE
- "The Bellman equation is one of the five most important ideas in artificial intelligence" -Bram Bakker

Markov decision processes

- The Markov decision process (MDP) is the classic formal model of a sequential decision problem
- Assume a fully-observable, stationary, and possibly stochastic environment
- A finite MDP consists of:
  - Discrete time $t = 0, 1, 2, \ldots$
  - A discrete set of states $s \in S$
  - A discrete set of actions $a \in A(s)$ for each s
  - A *transition function* $P^a_{ss'} = p(s'|s, a)$: probability of transitioning to state s' when taking action a at state s
  - A *reward function* $R^a_{ss'} = \mathbb{E}[r|s, a, s']$: expected reward when taking action a at state s and transitioning to s'
  - A planning horizon H or discount factor γ

MDP example: recycling robot

$S = \{\texttt{high}, \texttt{low}\}$

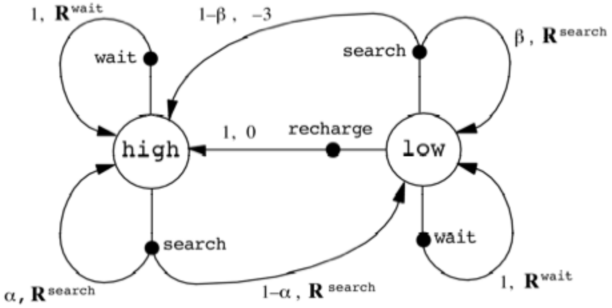
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$A(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$\mathbf{R}^{\texttt{search}} =$ expected no. of cans while searching

$\mathbf{R}^{\texttt{wait}} =$ expected no. of cans while waiting

$\mathbf{R}^{\texttt{search}} > \mathbf{R}^{\texttt{wait}}$



The Markov property

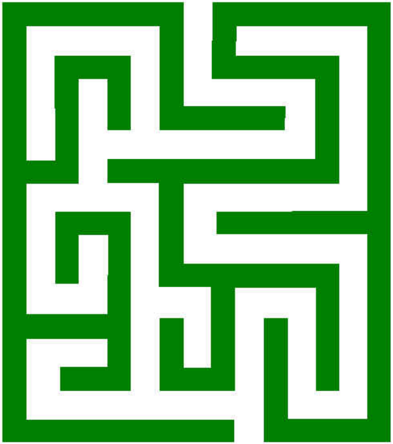
- For all $s_t, a_t, s_{t+1}, r_{t+1}$:

$$p(s_{t+1}, r_{t+1}|s_t, a_t) = p(s_{t+1}, r_{t+1}|s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0)$$

- The current state is a sufficient statistic for the agent's history
- Conditioning actions on that history cannot possibly help
- Can restrict search to *reactive policies*:
  - Stochastic reactive policy: $\pi(s, a) = p(a|s)$
  - Deterministic reactive policy: $\pi(s) = a$
  - In every MDP there exists at least one optimal deterministic reactive policy

Is it Markov? (1)

- A robot in a maze
- State: wall/no wall on all 4 sides
- Actions: move up, down, left, right, unless a wall is in the way



Is it Markov? (2)

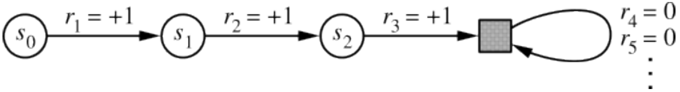
- A game of chess
- State: board position
- Actions: legal moves
- Opponent has a fixed reactive policy

Return

- The goal of the agent is to maximize the expected *return*, a sum over the rewards received.
- In an infinite-horizon task, the return is defined as:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- In a finite-horizon task, this becomes a finite summation
- In an infinite-horizon task that is *episodic* instead of *continuing*, we represent episode termination as transition to an *absorbing state* with self-transitions and zero reward.



Value functions

- *Value functions* are the primary tool for reasoning about future reward
- The *state-value function* of a policy π is:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\Big[R_t | s_t = s\Big] = \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\Big]$$

- The *action-value* of a policy π is:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\Big[R_t | s_t = s, a_t = a\Big] = \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\Big]$$

Bellman equation

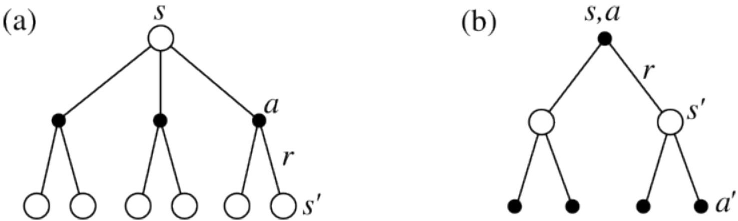
- The definition of V^π can be rewritten recursively by making use of the transition model, yielding the *Bellman equation*:

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

- This is a set of linear equations, one for each state, the solution of which defines the value of π
- A similar recursive definition holds for Q-values:

$$Q^\pi(s, a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \sum_{a'} \pi(s', a') Q^\pi(s', a') \right]$$

Backup diagrams



$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \Big[R^a_{ss'} + \gamma V^\pi(s') \Big]$$

$$Q^\pi(s,a) = \sum_{s'} P^a_{ss'} \Big[R^a_{ss'} + \gamma \sum_{a'} \pi(s',a') Q^\pi(s',a') \Big]$$

Optimal value functions

- Value functions define a partial ordering over policies:

$$\pi \succ \pi' \Rightarrow V^{\pi}(s) \geq V^{\pi'}(s), \forall s \in S$$

- There can be multiple optimal policies but they all share the same *optimal state-value function*:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in S$$

- They also share the same *optimal action-value function*:

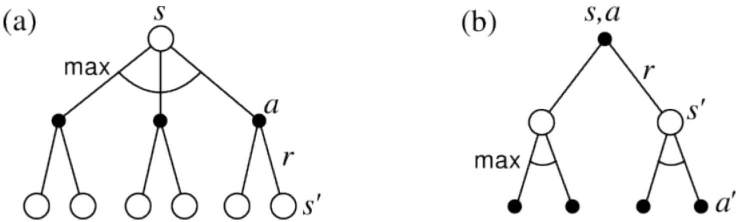
$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a), \forall s \in S, a \in A$$

Bellman optimality equations

- *Bellman optimality equations* express this recursively:

$$V^* = \max_{a \in A} \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s') \right]$$

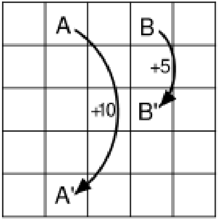
$$Q^*(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a \ \in A} Q^*(s', a') \right]$$



Why optimal value functions are useful

An optimal policy is *greedy* with respect to V^* or Q^*:

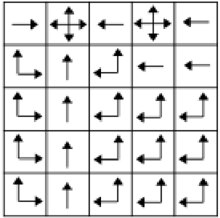
$$\pi^*(s) \in \arg\max_a Q^*(s,a) = \arg\max_a \left[R^a_{ss'} + \gamma \sum_{s'} P^a_{ss'} V^*(s') \right]$$



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*

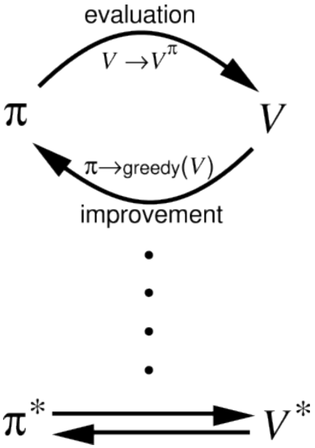


c) π^*

MDP planning

- MDPs give us a formal model of sequential decision making
- Given the optimal value function, computing an optimal policy is straightforward
- How can we find V^* or Q^*?
- Algorithms for *MDP planning* compute the optimal value function given a complete *model* of the MDP
- Given a model, V^* is usually sufficient

Dynamic programming approach



Policy evaluation (1)

- Rather than estimating value of each state independently, use Bellman equation to exploit the relationship between states
- Initial value function V_0 is chosen arbitrarily
- Policy evaluation update rule:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right]$$

- Apply to every state in each *sweep* of the state space
- Repeat over many sweeps
- Converges to the fixed point $V_k = V^\pi$

Policy evaluation (2)

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

Policy improvement (1)

- Policy evaluation yields V^π, the true value of π
- Use this to incrementally improve the policy by considering whether for some state s there is a better action $a \neq \pi(s)$
- Is choosing a in s and then using π better than using π, i.e.,

$$Q^\pi(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right] \geq V^\pi(s)?$$

- If so, then the *policy improvement theorem* tells us that changing π to take a in s will increase its value:

$$\forall s \in S, Q^\pi(s, \pi'(s)) \geq V^\pi(s) \Rightarrow \forall s \in S, V^{\pi'}(s) \geq V^\pi(s)$$

- In our case, $\pi = \pi'$ except that $\pi'(s) = a \neq \pi(s)$

Policy improvement (2)

- Applying this principle at all states yields the *greedy* policy with respect to V^π:

$$\pi'(s) \leftarrow \arg\max_a Q^\pi(s,a) = \arg\max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

- If $\pi = \pi'$, then $V^\pi = V^{\pi'}$ and for all $s \in S$:

$$V^{\pi'} = \max_{a \in A} \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^{\pi'}(s') \right]$$

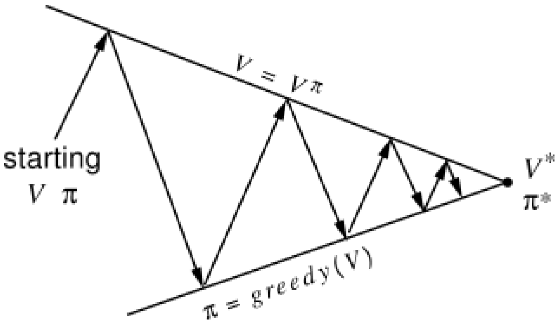
- This is equivalent to the Bellman optimality equation, implying that $V^\pi = V^{\pi'} = V^*$ and $\pi = \pi' = \pi^*$

Policy iteration (1)

$$\pi_0 \xrightarrow{\mathrm{E}} V^{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} V^{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi^* \xrightarrow{\mathrm{E}} V^*,$$

- Policy improvement makes result of policy evaluation obsolete
- Return to policy evaluation to compute $V^{\pi'}$
- Converges to the fixed point $V^\pi = V^*$

Policy iteration (2)



“Counterexample”

- Two actions (left and right) and two timesteps
- LL yields return of 5
- LR yields return of 0
- RL yields return of 0
- RR yields return of 10
- Policy of LL is a local maximum?

Value iteration

- We do not have to wait for policy evaluation to complete before doing policy improvement
- In extreme case, two steps are integrated in one update rule:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right]$$

- Turns Bellman optimality equation into an update rule
- This can also be written:

$$V_{k+1}(s) \leftarrow \max_a Q_{k+1}(s, a),$$
$$Q_{k+1}(s, a) \leftarrow \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right]$$

Monte-Carlo methods

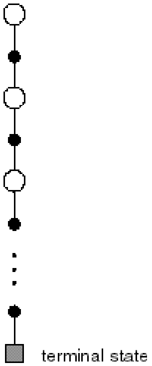
- Monte-Carlo (MC) methods are statistical techniques for estimating properties of complex systems via random sampling
- MC provides one way to perform reinforcement learning: finding optimal policies without a priori models of MDP
- MC for RL learns from complete sample returns in episodic tasks: uses value functions but not Bellman equations

Monte-Carlo policy evaluation

- Learn V^π without a model of the MDP
- Use π for many episodes
- For each state s, average observed returns after visiting s
- *Every-visit MC*: average returns for all visits to s in an episode
- *First-visit MC*: average returns only for first visit to s in an episode
- Both converge asymptotically

Monte-Carlo backup diagram

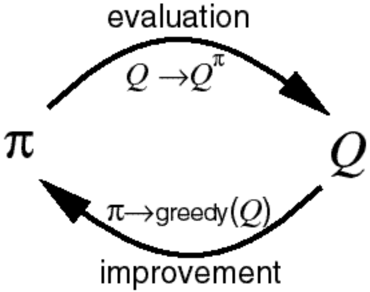
- Unlike dynamic programming, only one choice at each state
- Unlike dynamic programming, entire episode included: MC does not bootstrap
- Computational and sample costs to estimate $V^{\pi}(s)$ for one s are independent of $|S|$



Monte-Carlo estimation of Q-values

- MC methods most useful when no model is available
- π^* cannot be derived from V^* without a model so learn Q^*
- Can learn Q^π by averaging returns obtained when following π after taking action a in state s
- Converges asymptotically if every (s, a) visited infinitely often
- Requires explicit exploration of actions not favored by π

Monte-Carlo control



- Policy evaluation step: use MC methods
- Policy improvement step: $\pi(s) \leftarrow \arg\max_a Q(s, a)$

On-policy Monte-Carlo control

- Use ϵ-greedy soft policies:
  - Non-greedy actions: $\frac{\epsilon}{|A(s)|}$
  - Greedy action: $1 - \epsilon + \frac{\epsilon}{|A(s)|}$
- Replace greedification with soft greedifaction
- Policy improvement theorem guarantees any ϵ-greedy policy wrt to Q^π is an improvement over any ϵ-soft policy π
- Converges to the best ϵ-soft policy

Off-policy Monte-Carlo control (1)

- Evaluate an *estimation policy* using samples gathered from a *behaviour policy* if behaviour policy is sufficiently exploratory
- Useful if behaviour policy cannot be changed
- Also allows estimating a deterministic policy while still exploring with behaviour policy
- Use *importance sampling* to weight returns from behaviour policy by their probabilities under estimation policy

Importance sampling

- Interested in $\mathbb{E}_d[f(x)]$ where d is a *target distribution* over x
- Samples $f(x_1), f(x_2), \ldots, f(x_n)$ from *source distribution* d'
- Distributions d and d' are known but f is unknown
- Importance sampling is based on the following observation:

$$\mathbb{E}_d[f(x)] = \sum_x f(x)d(x) = \sum_x f(x)\frac{d(x)}{d'(x)}d'(x) = \mathbb{E}_{d'}[f(x)\frac{d(x)}{d'(x)}]$$

- This leads to the *importance sampling estimator*:

$$\mathbb{E}_d[f(x)] \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)\frac{d(x)}{d'(x)}$$

- In our case: target distribution comes from estimation policy; source distribution from behaviour policy

Off-policy Monte-Carlo control (2)

- Given n_s returns $R_i(s)$ from state s with probability $p_i(s)$ and $p'_i(s)$ of being generated by π and π':

$$V^\pi(s) \approx \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)} R_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'_i(s)}}$$

- $p_i(s)$ and $p'_i(s)$ are unknown but:

$$\frac{p_i(s)}{p'_i(s)} = \frac{\prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) P^{a_k}_{s_k s_{k+1}}}{\prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) P^{a_k}_{s_k s_{k+1}}} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}$$

Temporal-difference methods

- DP exploits Bellman equation but requires model
- MC doesn't require model but doesn't exploit Bellman equation
- TD methods can get the best of both worlds: exploit Bellman equation without requiring a model
- Core algorithms of model-free RL

TD(0)

- *Constant-α-MC* is a simple every-visit MC for nonstationary environments:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$$

- TD(0) just uses a different *update target*:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

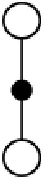
- TD(0) is a *bootstrapping method* because it bases updates on existing estimates, like DP

TD(0)

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    $a \leftarrow$ action given by π for s
    Take action a; observe reward, r, and next state, s'
    $V(s) \leftarrow V(s) + \alpha\big[r + \gamma V(s') - V(s)\big]$
    $s \leftarrow s'$
  until s is terminal

TD(0) backup diagram

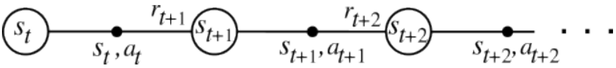
- Sampling: unlike DP but like MC, only one choice at each state
- Bootstrapping: like DP but unlike MC, use estimate from next state



Advantages of TD prediction methods

- TD methods require only experience, not a model
- TD, but not MC, methods can be fully incremental
- Learn before final outcome: less memory and peak computation
- Learn without the final outcome: from incomplete sequences
- Both MC and TD converge but TD tends to be faster

Sarsa: on-policy TD estimation of Q-values



- To learn π^* with TD, we need to learn Q^* instead of V^*
- Sarsa updates Q by bootstrapping off next (s, a):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Sarsa: on-policy TD control

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g., ε-greedy)
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    until s is terminal

Expected sarsa

- Take expectation wrt actions:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \sum_a \pi(s_{t+1}, a) Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- Action probabilities known from policy but more computation needed
- Reduces variance in updates

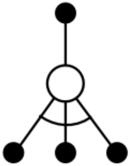
Q-learning: off-policy TD control

Make TD off-policy: bootstrap with best action, not actual action:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

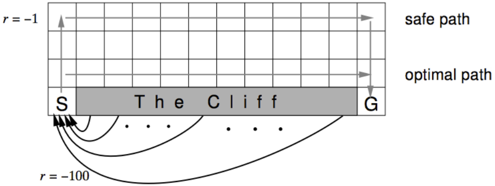
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$;
    until s is terminal

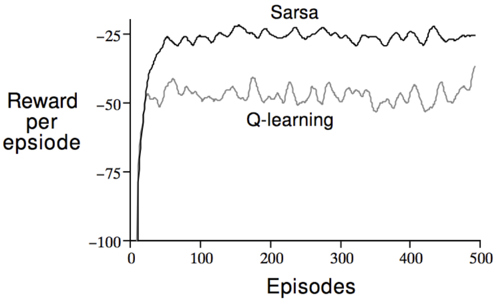
Q-learning backup diagram



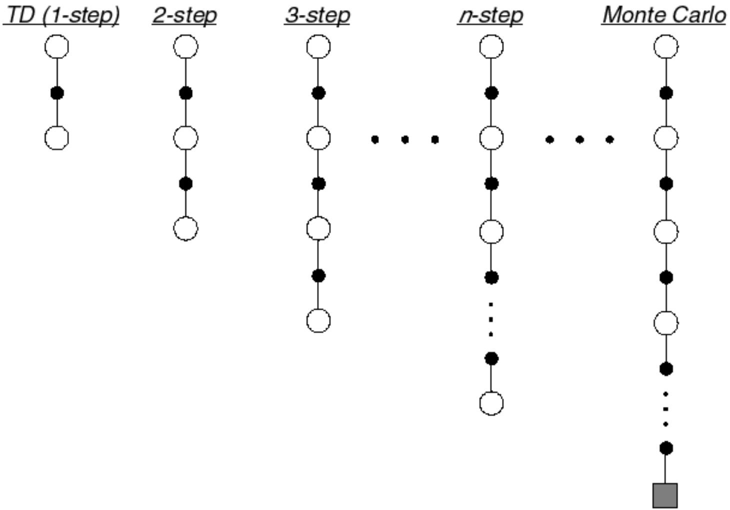
What is Sarsa's backup diagram?

Example: cliff walking

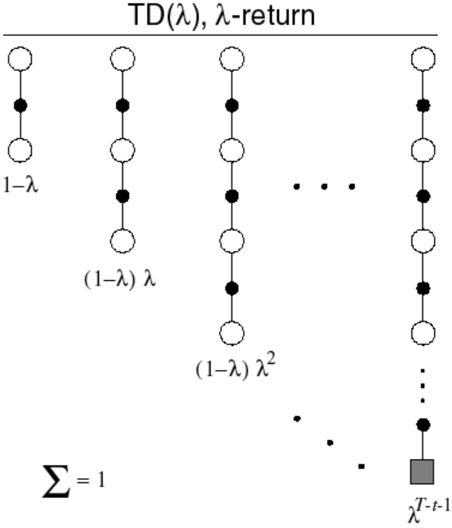




n-step updates



Eligibility traces: TD(λ)



Unified view