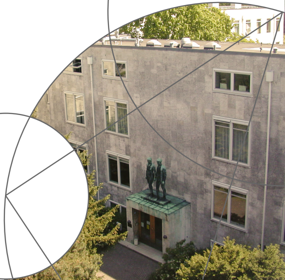
Reinforcement Learning

Advanced Topics in Machine Learning

Christian Igel
Department of Computer Science



igel@diku.dk

Reinforcement learning

> Reinforcement learning is a branch of machine learning concerned with using experience gained through interacting with the world and evaluative feedback to improve a system's ability to make behavioural decisions.
>
> M. L. Littman, *Nature*, 2015

Outline

1. Markov Decision Processes
2. Dynamic Programming
3. Monte Carlo Algorithms

Outline

1. Markov Decision Processes
2. Dynamic Programming
3. Monte Carlo Algorithms

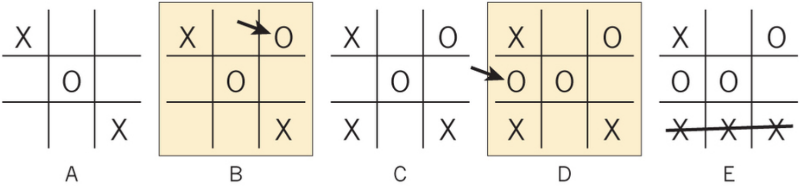
Types of feedback

- Exhaustive vs. sampled feedback: A learner given exhaustive feedback is exposed to all possible situations, given sampled feedback only to a subset
- Supervised vs. evaluative feedback: Supervised feedback provides examples with known optimal decisions, evaluative feedback gives an assessment to the effectiveness of the learners decisions
- One-shot vs. sequential feedback: In the one-shot scenario all feedback is provided directly after a decision, while in sequential feedback there may be longer-term impacts that are evaluated over a sequence of decisions

In reinforcement learning (RL), we try to tackle evaluative, sequential, sampled feedback.

M. L. Littman, *Nature*, 2015

Nought and crosses



M. L. Littman, *Nature*, 2015

The agent-environment-interface

- Agent and environment interact at discrete time steps $t = 0, 1, \ldots$
- At time step t, the agent
  - observes current state $s_t \in S$,
  - produces action $a_t \in A(s_t)$,
  - gets resulting reward $r_{t+1} \in \mathbb{R}$, and
  - transitions to next state $s_{t+1} \in S$.

The agent follows a policy

A policy π_t at step t maps states to action probabilities:

$$\pi_t : S \times A \to [0, 1]$$

$\pi(s, a)$ is the probability that $a_t = a$ if $s_t = s$.

For a deterministic policy we view π_t as a function $\pi_t : S \to A$.

Reinforcement learning methods specify how the agent changes its policy as a result of experience.
Roughly, the agents goal is to get as much reward as it can over the long run.

Getting the degree of abstraction right

- Time steps need not refer to fixed intervals of real time.
- Actions can be low level (e.g., voltages to motors), or high level (e.g., accept a job offer), "mental" (e.g., shift in focus of attention), etc.
- States can low-level "sensations ", or they can be abstract, symbolic, based on memory, or subjective (e.g., the state of being " surprised" or "lost ").
- An RL agent is not like a whole animal or robot, which consist of many RL agents as well as other components.
- The environment is not necessarily unknown to the agent, only incompletely controllable.
- Reward computation is in the agent's environment because the agent cannot change it arbitrarily.

Goals, rewards, and returns

- Is a scalar reward signal an adequate notion of a goal? Maybe not, but it is surprisingly flexible.
- A goal should specify what we want to achieve, not how we want to achieve it.
- A goal must be outside the agent's direct control – thus outside the agent.
- The agent must be able to measure success
    - explicitly,
    - frequently during its lifespan.

Rewards and returns

Suppose the sequence of rewards after step t is:

$$R_t = r_{t+1}, r_{t+2}, r_{t+3}, \dots$$

What do we want to maximize?

In general, we want to maximize the expected return $E\{R_t\}$ for each step t.

Episodic and continuing tasks

Episodic tasks: Interaction breaks naturally into episodes, e.g., plays in a game, trips through a maze. We have

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T \quad ,$$

where T is a final step at which a terminal state is reached ending an episode.

Continuing tasks: No natural episodes, we consider discounted return:

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

where γ, $0 \leq \gamma \leq 1$, is the discount rate.

shortsighted $0 \longleftarrow \gamma \longrightarrow 1$ farsighted

A unified notation

- In episodic tasks, we number the time steps of each episode starting from zero.
- We usually do not have distinguish between episodes, so we write s_t instead of $s_{t,j}$ for the state at step t of episode j.
- Think of each episode as ending in an absorbing state that always produces reward of zero.

We can cover all cases by writing

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

where γ can be 1 only if a zero reward absorbing state is always reached.

The Markov property

A "the state" at step t collects whatever information is available to the agent at step t about its environment.

The state can include immediate "sensations" highly processed sensations, and structures built up over time from sequences of sensations.

Ideally, a state should summarize past sensations so as to retain all "essential" information, i.e., it should have the Markov property:

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0\} = \Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}$$

for all s', r and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0$.

Markov decision process

- If a RL task has the Markov Property, it is basically a Markov Decision Process (MDP).
- If state and action sets are finite, it is a finite MDP.
- To define a finite MDP, you need to give:
  - State and action sets
  - One-step dynamics defined by transition probabilities:

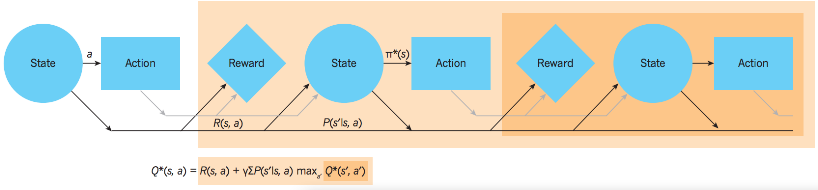
$$P^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \quad \forall s', s \in S\,, a \in A(s)$$

  - Expected rewards:

$$R^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \quad \forall s', s \in S\,, a \in A(s)$$

  - (distribution over start states, γ)

MDP schema



M. L. Littman, *Nature*, 2015

State-value functions

The value of a state is the expected return starting from that state; depends on the agent's policy.

State-value function for a policy π:

$$V^{\pi}(s) = E_{\pi}\left\{R_t | s_t = s\right\} = E_{\pi}\left\{\sum_{k}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\}$$

Action-value functions

The value of taking an action in a state under policy π is the expected return starting from that state, taking that action, and thereafter following π.

Action-value function for a policy π:

$$Q^\pi(s,a) = E_\pi\left\{R_t \,\middle|\, s_t = s, a_t = a\right\}$$
$$= E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a\right\}$$

Bellman equation for a policy

$$
\begin{aligned}
R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots \\
&= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \dots \right) \\
&= r_{t+1} + \gamma R_{t+1}
\end{aligned}
$$

implies

$$
\begin{aligned}
V^\pi(s) &= E_\pi \{ R_t \,|\, s_t = s \} = E_\pi \{ r_{t+1} + \gamma R_{t+1} \,|\, s_t = s \} \\
&= E_\pi \{ r_{t+1} + \gamma E_\pi \{ R_{t+1} \} \,|\, s_t = s \} \\
&= E_\pi \{ r_{t+1} + \gamma V^\pi(s_{t+1}) \,|\, s_t = s \} \\
&= E_\pi \left\{ E_\pi \{ r_{t+1} + \gamma V^\pi(s_{t+1}) \,|\, s_t = s, a_t = a \} \,\middle|\, s_t = s \right\} \\
&= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]
\end{aligned}
$$

Value function for π is solution of this set of equations.

Optimal value function

- For finite MDPs, policies can be partially ordered:

$$\pi \geq \pi' \text{ if and only if } V^{\pi}(s) \geq V^{\pi'}(s) \text{ for all } s \in S$$

- There is always at least one (and possibly many) optimal policy π^* that is better than or equal to all the others.
- Optimal policies share the same optimal state-value function:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all } s \in S$$

- Optimal policies share the same optimal action-value function:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \text{for all } s \in S \text{ and } a \in A(s)$$

This is the expected return for taking a in s and thereafter following an optimal policy.

Bellman optimality equations I

The value of a state under an optimal policy must equal the expected return for the best action for that state:

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^*(s,a) \\
&= \max_{a \in A(s)} E_\pi \left\{ r_{t+1} + \gamma V^*(s_{t+1}) \,\middle|\, s_t = s \right\} \\
&= \max_{a \in A(s)} \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^*(s') \right]
\end{aligned}
$$

V^* is the unique solution of this system of nonlinear equations.

Bellman optimality equations II

Bellman optimality equation for Q^*:

$$\begin{aligned} Q^*(s,a) &= E_\pi \left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,\middle|\, s_t = s, a_t = a \right\} \\ &= \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right] \end{aligned}$$

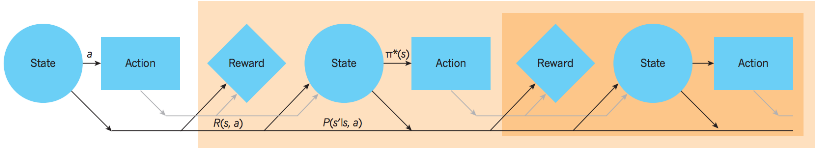
Q^* is the unique solution of this system of nonlinear equations.

Any policy that is greedy with respect to V^* is an optimal policy. Therefore, given V^*, one-step-ahead search produces the long-term optimal actions.

Given Q^*, the agent does not even have to do a one-step-ahead search:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} Q^*(s,a)$$

MDP schema revisited



$Q^*(s,a) = R(s,a) + \gamma \sum P(s'|s,a) \max_{a'} Q^*(s',a')$

M. L. Littman, *Nature*, 2015

Outline

1. Markov Decision Processes
2. Dynamic Programming
3. Monte Carlo Algorithms

Solving the Bellman optimality equation

- Finding an optimal policy by solving the Bellman optimality equation requires the following:
    - Accurate knowledge of environment dynamics;
    - Enough space and time to do the computation
    - Markov property
- How much space and time do we need?
    - Polynomial in number of states using dynamic programming (DP) methods,
    - but, number of states is often huge (e.g., backgammon has 10^{20} states).
- We usually have to settle for approximations.
- Many RL methods can be understood as approximately solving the Bellman optimality equation.

Policy evaluation

For a given policy π, compute the state-value function V^π.
Recall state-value function:

$$V^\pi(s) = E_\pi\left\{R_t \,\middle|\, s_t = s\right\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s\right\}$$

Recall Bellman equation for V^π:

$$\begin{aligned} V^\pi(s) &= \sum_a \pi(s,a) E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a\right\} \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^\pi(s')\right] \end{aligned}$$

This amounts to a system of $|S|$ linear equations.

Bellman equation in vector form

Assume $S = (s_1, \dots, s_k)$ we define $\boldsymbol{v}^\pi \in \mathbb{R}^k$ with

$$v_i^\pi = V^\pi(s_i)$$

and the expected immediate reward vector $\boldsymbol{r}^\pi \in \mathbb{R}^k$

$$r_i^\pi = \sum_a \pi(s_i, a) \sum_{j=1}^{k} P_{s_i s_j}^a R_{s_i s_j}^a$$

and the transition matrix $\boldsymbol{T}^\pi \in \mathbb{R}^{k \times k}$ with

$$T_{ij}^\pi = \sum_a \pi(s_i, a) P_{s_i s_j}^a$$

and write the Bellman equation in vector form:

$$\boldsymbol{v}^\pi = \boldsymbol{r}^\pi + \gamma \boldsymbol{T}^\pi \boldsymbol{v}^\pi$$

Solving Bellman equation in vector form

Bellman equation:

$$\boldsymbol{v}^{\pi} = \boldsymbol{r}^{\pi} + \gamma \boldsymbol{T}^{\pi} \boldsymbol{v}^{\pi}$$

Knowing the full finite MDP, we can solve the Bellman equation directly:

$$\begin{aligned} \boldsymbol{v}^{\pi} &= \boldsymbol{r}^{\pi} + \gamma \boldsymbol{T}^{\pi} \boldsymbol{v}^{\pi} \\ (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi}) \boldsymbol{v}^{\pi} &= \boldsymbol{r}^{\pi} \\ \boldsymbol{v}^{\pi} &= (\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi})^{-1} \boldsymbol{r}^{\pi} \end{aligned}$$

Iterative policy evaluation

$$V_0 \longrightarrow V_1 \longrightarrow \ldots \longrightarrow V_k \longrightarrow V_{k+1} \longrightarrow \ldots\ldots \longrightarrow V^\pi$$

A sweep consists of applying a backup operation to each state.

Full policy evaluation backup:

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

Iterative policy evaluation algorithm

Algorithm 1: Dynamic programming

Input: policy π to be evaluated, arbitrary V, threshold $\theta > 0$

repeat
- $\Delta \leftarrow 0$
- **foreach** $s \in S$ **do**
  - $v \leftarrow V(s)$
  - $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V(s')\right]$
  - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

Output: $V \approx V^\pi$

Policy improvement theorem: Idea

- Suppose we have computed V^π for a deterministic policy π.
- For a given state s, would it be better to do an action $a \neq \pi(s)$?
- The value of doing a in s is:

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\left\{r_{t+1} + \gamma V^\pi(s_{t+1}) \,\middle|\, s_t = s, a_t = a\right\} \\ &= \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^\pi(s')\right] \end{aligned}$$

- It is better to switch to action a for state s if and only if:

$$Q^\pi(s,a) > V^\pi(s)$$

Policy improvement theorem

Given two policies $\pi, \pi' : S \to A$. If for *all* states $s \in S$

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s)$$

and for *some* states $s \in S$

$$Q^\pi(s, \pi'(s)) > V^\pi(s)$$

then for *all* states $s \in S$

$$V^{\pi'}(s) \geq V^\pi(s)$$

and for *some* states $s \in S$

$$V^{\pi'}(s) > V^\pi(s) \quad .$$

Policy improvement theorem proof I

$$
\begin{aligned}
V^\pi(s) &\le Q^\pi(s, \pi'(s)) \\
&= E_{\pi'}\left\{r_{t+1} + \gamma V^\pi(s_{t+1}) \,\middle|\, s_t = s\right\} \\
&\le E_{\pi'}\left\{r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) \,\middle|\, s_t = s\right\} \\
&= E_{\pi'}\left\{r_{t+1} + \gamma E_{\pi'}\left\{r_{t+2} + \gamma V^\pi(s_{t+2})\right\} \,\middle|\, s_t = s\right\} \\
&= E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) \,\middle|\, s_t = s\right\} \\
&\le E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 Q^\pi(s_{t+2}, \pi'(s_{t+2})) \,\middle|\, s_t = s\right\} \\
&= E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 E_{\pi'}\left\{r_{t+3} + \gamma V^\pi(s_{t+3})\right\} \,\middle|\, s_t = s\right\} \\
&= E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^\pi(s_{t+3}) \,\middle|\, s_t = s\right\} \\
&\ldots \\
&\le E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots \,\middle|\, s_t = s\right\} \\
&= V^{\pi'}(s) \qquad \text{(for all } s \in S\text{)}
\end{aligned}
$$

Policy improvement theorem proof II

Change policy accordingly for all states to get a new policy π' that is greedy with respect to $V^{\pi'}$:

$$\begin{aligned}\pi'(s) &= \operatorname*{argmax}_a Q^\pi(s,a) \\ &= \operatorname*{argmax}_a \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^\pi(s')\right]\end{aligned}$$

Then $\forall s \in S : V^{\pi'}(s) \geq V^\pi(s)$.

$V^{\pi'} = V^\pi$ implies
$\forall s \in S : V^\pi(s) = \max_a \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^\pi(s')\right]$.
That's the Bellman optimality equation and $V^\pi = V^{\pi'} = V^*$.

$\square$

Policy iteration: Idea

Alternate:

- Policy evaluation
- Policy improvement

$$\pi_0 \xrightarrow[\text{evaluate}]{} V^{\pi_0} \xrightarrow[\text{improve}]{} \pi_1 \xrightarrow[\text{evaluate}]{} V^{\pi_1} \xrightarrow[\text{improve}]{} \dots$$

$$\dots \xrightarrow[\text{improve}]{} \pi^* \xrightarrow[\text{evaluate}]{} V^{\pi^*} \xrightarrow[\text{improve}]{} \pi^*$$

Policy iteration algorithm

Algorithm 2: Policy iteration

Input: policy π and value function V, threshold $\theta > 0$

repeat

  repeat `// policy evaluation`

    $\Delta \leftarrow 0$

    foreach $s \in S$ **do**

      $v \leftarrow V(s)$

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

      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

  until $\Delta < \theta$

  $f_{\text{stable}} \leftarrow$ True

  foreach $s \in S$ **do** `// policy improvement`

    $a \leftarrow \pi(s)$

    $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$

    if $a \neq \pi(s)$ **then** $f_{\text{stable}} \leftarrow$ False

until $f_{\text{stable}} =$ True

Output: $\pi \approx \pi^*, V \approx V^\pi$

Value iteration

Recall the *full policy evaluation backup*:

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

Here is the *full value iteration backup*:

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

Value iteration algorithm

Algorithm 3: Value iteration

Input: arbitrary value function V, threshold $\theta > 0$

repeat

  $\Delta \leftarrow 0$

  foreach $s \in S$ **do**

    $v \leftarrow V(s)$

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

    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

foreach $s \in S$ **do** $\pi(s) \leftarrow \mathrm{argmax}_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V(s')\right]$

Output: $\pi \approx \pi^*, V \approx V^\pi$

Q-value iteration

Full policy evaluation backup, $\forall S \in S, a \in A$:

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

Full value iteration backup, $\forall S \in S, a \in A$:

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

Convergence of value iteration I

Assumption: $\gamma < 1$

$$\Delta_k = \|Q^*(s,a) - Q_k(s,a)\|_\infty = \max_{s,a} |Q^*(s,a) - Q_k(s,a)|$$

$$\begin{aligned} Q_{k+1}(s,a) &= \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q_k(s',a') \right] \\ &\leq \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q^*(s',a') \right] \\ &= Q^*(s,a) \end{aligned}$$

Convergence of value iteration II

$$
\begin{aligned}
Q_{k+1}(s,a) &= \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q_k(s',a') \right] \\
&\geq \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} \left(Q^*(s',a') - \Delta_k \right) \right] \\
&= \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \max_{a'} Q^*(s',a') \right] - \gamma \Delta_k \\
&= Q^*(s,a) - \gamma \Delta_k \\
&\Rightarrow \Delta_{k+1} \leq \gamma \Delta_k
\end{aligned}
$$

Efficiency of DP

- To find an ϵ-optimal policy using value iteration is polynomial in the number of states ...
- But, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called "the curse of dimensionality").
- In practice, classical DP can be applied to problems with a few millions of states.
- Asynchronous DP can be applied to larger problems, and appropriate for parallel computation.
- It is surprisingly easy to come up with MDPs for which DP methods are not practical.

Outline

1. Markov Decision Processes
2. Dynamic Programming
3. Monte Carlo Algorithms

Monte Carlo algorithms

- Monte Carlo (MC) methods learn from complete sample returns

$\rightarrow$ Only defined for episodic tasks

- Monte Carlo methods learn directly from experience
  - On-line: No model necessary and still attains optimality
  - Simulated: No need for a full model

Monte Carlo policy evaluation

- Goal: learn $V^\pi(s)$
- Given: some number of episodes under π which contain s
- 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

MC policy evaluation: Algorithm

Algorithm 4: First-visit Monte Carlo policy evaluation

Input: policy π to be evaluated and arbitrary value function V

foreach $s \in S$ **do**
- $R(s) \leftarrow 0$ `// accumulated returns`
- $c(s) \leftarrow 0$ `// counter`

repeat
- generate an episode using π
- **foreach** *state* s *appearing in the episode* **do**
  - $R(s) \leftarrow R(s) +$ return following first occurrence of s
  - $c(s) \leftarrow c(s) + 1$
  - $V(s) \leftarrow R(s)/c(s)$

until *until some stopping criterion is met*

Output: $V \approx V^{\pi}$

MC vs. DP

- Entire episode included
- Only one choice at each state (unlike DP)
- MC does not bootstrap
- Time required to estimate one state does not depend on the total number of states

More on MC

- Monte Carlo is most useful when a model is not available
- We want to learn Q^*
- $Q^\pi(s, a)$ – average return starting from state s and action a following π
- Also converges asymptotically if every state-action pair is visited
- Exploring starts: Every state-action pair has a non-zero probability of being the starting pair

Monte Carlo control

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

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

Convergence of Monte Carlo control

- Policy improvement theorem tells us:

$$\begin{aligned} Q^{\pi_k}(s, \pi_{k+1}(s)) &= Q^{\pi_k}(s, \operatorname{argmax}_a Q^{\pi_k}(s, a)) \\ &= \max_a Q^{\pi_k}(s, a) \\ &\geq Q^{\pi_k}(s, \pi_k(s)) \\ &= V^{\pi_k}(s) \end{aligned}$$

- This assumes exploring starts and infinite number of episodes for MC policy evaluation. To solve the latter, one update only to a given level of performance
- Alternate between evaluation and improvement per episode.

Monte Carlo control: Algorithm

Algorithm 5: Monte Carlo control with exploring starts

Input: arbitrary policy π and state-value function Q

foreach $(s, a) \in S \times A$ **do**
    $R(s, a) \leftarrow 0$ `// accumulated returns`
    $c(s, a) \leftarrow 0$ `// counter`

repeat
    generate an episode using exploring starts and π
    foreach *pair* s, a *appearing in the episode* **do**
        $R(s, a) \leftarrow R(s, a)+$ return following first occurrence of s, a
        $c(s, a) \leftarrow c(s, a) + 1$
        $Q(s, a) \leftarrow R(s, a)/c(s, a)$
    foreach *state* s *appearing in the episode* **do**
        $\pi(s) \leftarrow \operatorname{argmax}_a Q(s, a)$

until *some stopping criterion is met*

Output: $\pi \approx \pi^*, Q \approx Q^*$

On-policy Monte-Carlo control

- On-policy: learn about policy currently executing
- How do we get rid of exploring starts?
  - Need soft policies: $\pi(s,a) > 0$ for all s and a
  - Example: ϵ-soft policy:

$$\underset{\text{non-max}}{\frac{\epsilon}{|A(s)|}} \qquad \underset{\text{greedy}}{1-\epsilon\frac{\epsilon}{|A(s)|}}$$

- Similar to GPI: move policy *towards* greedy policy (e.g., ϵ-soft)
- Converges to best ϵ-soft policy

On-policy Monte-Carlo control

Algorithm 6: On-policy MC-control

Input: arbitrary ϵ-greedy policy π and state-value function Q

foreach $(s, a) \in S \times A$ **do** $R(s, a) \leftarrow 0$; $c(s, a) \leftarrow 0$

repeat

  generate an episode using π

  foreach *pair* s, a *appearing in the episode* **do**

    $R(s, a) \leftarrow R(s, a) +$ return following first occurrence of s, a

    $c(s, a) \leftarrow c(s, a) + 1$; $Q(s, a) \leftarrow R(s, a)/c(s, a)$

  foreach *state* s *appearing in the episode* **do**

    $a^* \leftarrow \operatorname{argmax}_a Q(s, a)$

    foreach *state* $a \in A(s)$ **do**

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

until *some stopping criterion is met*

Output: $\pi \approx \pi^*, Q \approx Q^*$

Off-policy MC control I

- Behavior policy π' generates behavior in environment
- Estimation policy π is policy being learned about
- Challenge: Learning about π while following π'
- Idea: Average returns from behavior policy by their probabilities in the estimation policy

Off-policy MC control II

Goal: Estimate V^π or Q^π based on episodes generated following $\pi \neq \pi'$
Assumption: $\pi(s,a) > 0 \rightarrow \pi'(s,a) > 0$

Consider i-th first visit of state s in episodes generate by π'. Let the time of this visit be $t_i(s)$ and $T_i(s)$ the time of termination of the i-th episode; let $p_i(s)$ and $p'_i(s)$ be the probabilities of this sequence

$$s_{t_i(s)}, a_{t_i(s)}, r_{t_i(s)+1}, s_{t_i(s)+1}, a_{t_i(s)+1}, r_{t_i(s)+2}, \dots, a_{T_i(s)-1}, r_{T_i(s)}, s_{T_i(s)}$$

under π and π', let the observed return be $R_i(s)$.

Off-policy MC control III

Desired MC estimate after observing n_s returns from s is

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

($\rightarrow$ weighted importance sampling)

$$p_i(s) = \prod_{k=t_i(s)}^{T_i(s)-1} \pi(s_k, a_k) P_{s_k s_{k+1}}^{a_k}$$

$$\Rightarrow \frac{p_i(s)}{p'_i(s)} = \frac{\prod_{k=t_i(s)}^{T_i(s)-1} \pi(s_k, a_k) P_{s_k s_{k+1}}^{a_k}}{\prod_{k=t_i(s)}^{T_i(s)-1} \pi'(s_k, a_k) P_{s_k s_{k+1}}^{a_k}} = \prod_{k=t_i(s)}^{T_i(s)-1} \frac{\pi_i(s_k, a_k)}{\pi'_i(s_k, a_k)}$$

Can be estimated without knowing the environment's dynamics!

Off-policy MC control VI

What's the difference if we want to adapt Q^π?

Let $p_i(s,a)$ and $p'_i(s,a)$ be the probabilities of this sequence

$$s_{t_i(s)}, a_{t_i(s)}, r_{t_i(s)+1}, s_{t_i(s)+1}, a_{t_i(s)+1}, r_{t_i(s)+2}, \dots, a_{T_i(s)-1}, r_{T_i(s)}, s_{T_i(s)}$$

under π and π'.

It holds $p_i(s,a) = p_i(s)/\pi(s,a)$, $p'_i(s,a) = p'_i(s)/\pi'(s,a)$ and thus

$$\frac{p_i(s,a)}{p'_i(s,a)} = \prod_{k=t_i(s)+1}^{T_i(s)-1} \frac{\pi_i(s_k,a_k)}{\pi'_i(s_k,a_k)}$$

Off-policy MC control: Algorithm

Algorithm 7: Off-policy MC control

Input: arbitrary policies π (deterministic) and π' (soft) and function Q

foreach $(s, a) \in S \times A$ **do**
    $N(s, a) \leftarrow 0$ `// numerator of` $Q(s, a)$
    $D(s, a) \leftarrow 0$ `// denominator of` $Q(s, a)$

repeat
    generate an episode $s_0, a_0, r_1, s_1, \ldots, a_{T-1}, r_T, s_T$ using π'
    $\tau \leftarrow$ last time in episode at which $a_\tau \neq \pi(s_\tau)$
    foreach *pair* s, a *appearing in the episode later than* τ **do**
        $t \leftarrow$ the time of first occurrence of s, a such that $t > \tau$
        $w = \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k, a_k)}$ `// ` π ` deterministic`
        $N(s, a) \leftarrow N(s, a) + wR_t$
        $D(s, a) \leftarrow D(s, a) + w$
        $Q(s, a) \leftarrow \frac{N(s,a)}{D(s,a)}$
    foreach *state* s **do** $\pi(s) \leftarrow \operatorname{argmax}_a Q(s, a)$
until *until some stopping criterion is met*

Output: deterministic policy π

Off-policy MC control: Implementation

- MC can be implemented incrementally to save memory
- Compute the weighted average of each return

$$V_n = \frac{\sum_{k=1}^{n} w_k R_k}{\sum_{k=1}^{n} w_k}$$

non-incremental

$$\begin{aligned} V_0 &= W_0 = 0 \\ W_{n+1} &= W_n + w_{n+1} \\ V_{n+1} &= V_n + \frac{w_{n+1}}{W_{n+1}} \left[R_{n+1} - V_n \right] \end{aligned}$$

incremental

Summary Monte Carlo methods

- MC has several advantages over DP:
  - Can learn directly from interaction with environment
  - No need for full models
  - No need to learn about all states
  - Less harm by violations of Markov property
- MC methods provide an alternate policy evaluation process
- One issue to watch for: Maintaining sufficient exploration
  - Exploring starts, soft policies
- No bootstrapping (as opposed to DP)

Further reading

Many slides are based on slides for the seminal book:

Sutton, R. S. & Barto, A. G. Reinforcement Learning: An Introduction, MIT Press, 1998

Recent review, from which images were taken:

Littman, M. L.. Reinforcement learning improves behaviour from evaluative feedback. *Nature* **521**: 445–451, 2015

Good more recent advanced book:

Szepesvári, C. Algorithms for Reinforcement Learning, Morgan & Claypool, 2010