Reinforcement Learning and Planning for MDPs: Monte-Carlo Methods

Lecturers: Diederik M. Roijers & Ann Nowé

TA: Denis Steckelmacher

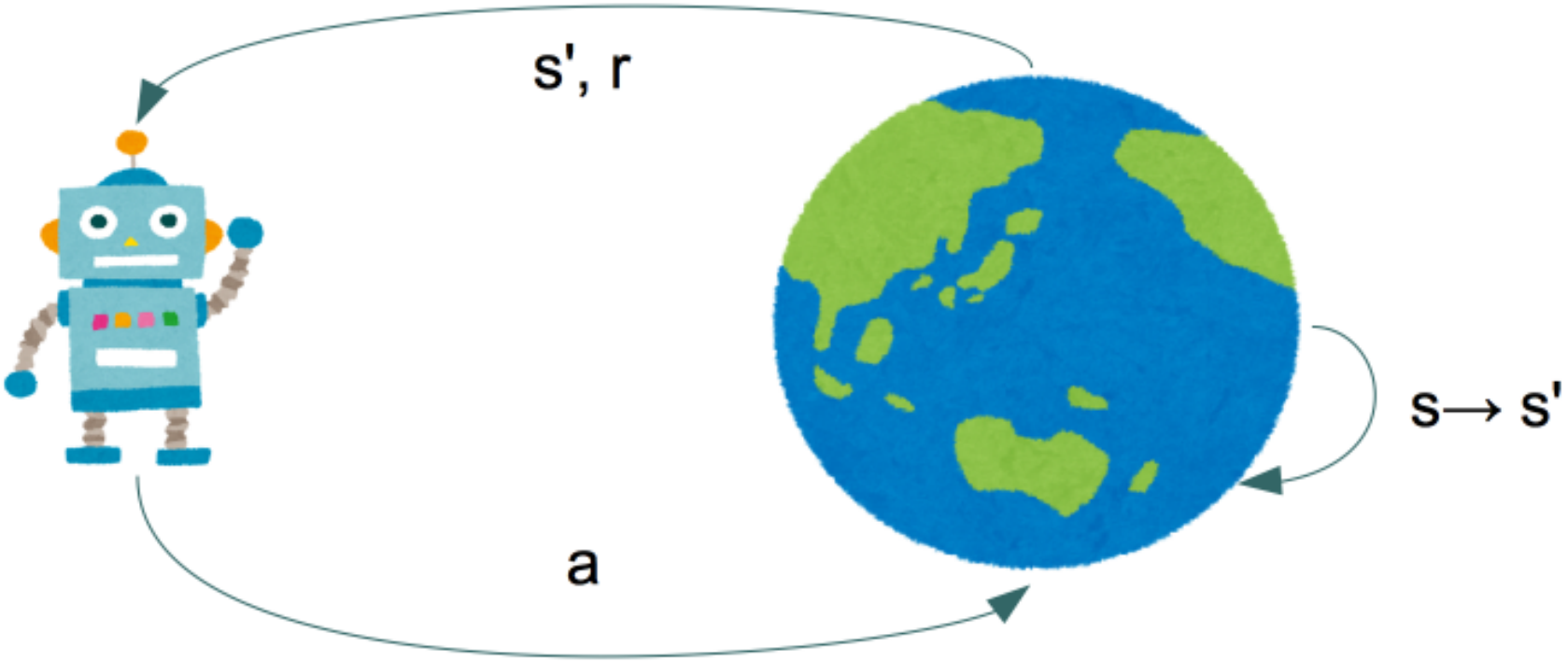
AI laboratory
Vrije Universiteit Brussel

Fall, 2017

Reinforcement Learning in Sequential Settings

- Agent does not have a (complete) model of the environment
- Agent learns about environment through interaction
- Last time: reinforcement learning in multi-armed bandits
  - Stateless
  - Looking for the optimal action / arm

Reinforcement Learning: MDPs



Reinforcement Learning Settings

- Online / Offline
  - Online:
  - Offline:
- On-policy / Off-policy
  - On-policy:
  - Off-policy:

Reinforcement Learning Settings

- Online / Offline
  - Online: we care about the accrued rewards during learning
  - Offline: we only care about the quality of the policy after learning
- On-policy / Off-policy
  - On-policy:
  - Off-policy:

Reinforcement Learning Settings

- Online / Offline
  - Online: we care about the accrued rewards during learning
  - Offline: we only care about the quality of the policy after learning
- On-policy / Off-policy
  - On-policy: we evaluate and/or optimise the value of the policy that we are executing
  - Off-policy: we evaluate and/or optimise the value of a different (possibly better) policy

Reinforcement Learning Choices

- Representation choices
  - explicitly learn a model of the environment?
    (today: no)
  - explicitly represent the policy the agent is learning?
    (today: no)
  - explicitly represent a state-based value function, $V(s)$ and/or $Q(s, a)$?
    (today: yes)

Reinforcement Learning Choices

- Bootstrapping choices
  - Standard Bellman backups (bootstrap from value of next state)

$$V^{\pi}(s_t) = E[r_t + \gamma V^{\pi}(s_{t+1})|\pi]$$

  - Deeper backups (eligibility traces)

$$V^{\pi}(s_t) = E[r_t + \gamma r_{t+1} + ... + \gamma^{n-1} r_{t+n-1} + \gamma^n V^{\pi}(s_{t+n})|\pi]$$

  - *Or just use the entire episode (Monte Carlo methods)*

$$V^{\pi}(s_t) = E[r_t + \gamma r_{t+1} + ... + \gamma^{t_{end}-t} r_{t_{end}}|\pi]$$

Monte-Carlo methods

- Monte-Carlo (MC) methods are statistical techniques for estimating properties of complex systems via random sampling
  - Wide range of applications, across sciences (math, physics, biology, etc.)
  - "Randomization, bootstrap and Monte Carlo methods in biology" by B.F.J. Manly cited over 8000 times
- MC for RL: finding optimal policies without a priori models of MDP by random *roll-outs* and estimating expected returns (i.e., the value)
- MC for RL learns from complete sample returns in episodic tasks: uses value functions but not Bellman equations

Monte-Carlo methods

- Monte-Carlo (MC) methods are statistical techniques for estimating properties of complex systems via random sampling
- MC for RL: finding optimal policies without a priori models of MDP by random *roll-outs* and estimating expected returns (i.e., the value)
  - Model-free RL
- MC for RL learns from complete sample returns in episodic tasks: uses value functions but not Bellman equations
  - No bootstrapping

Monte-Carlo policy evaluation

1. Policy evaluation
2. On-Policy learning
3. Off-Policy learning
4. Planning: Monte-Carlo Tree Search

Monte-Carlo policy evaluation

- Obtain V^π by doing roll-outs using π
- For each state s, average observed returns after visiting s
  - *Every-visit MC*: average returns for every time s is visited in an episode; if a state is visited x times in an episode, there are x data points for estimating $V^\pi(s)$
  - *First-visit MC*: average returns only for the first time s is visited in an episode; if a state is visited $x > 1$ times in an episode, there is one data point for estimating $V^\pi(s)$

First-visit Monte-Carlo policy evaluation

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:

(a) Generate an episode using π

(b) For each state s appearing in the episode:

$R \leftarrow$ return following the first occurrence of s

Append R to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

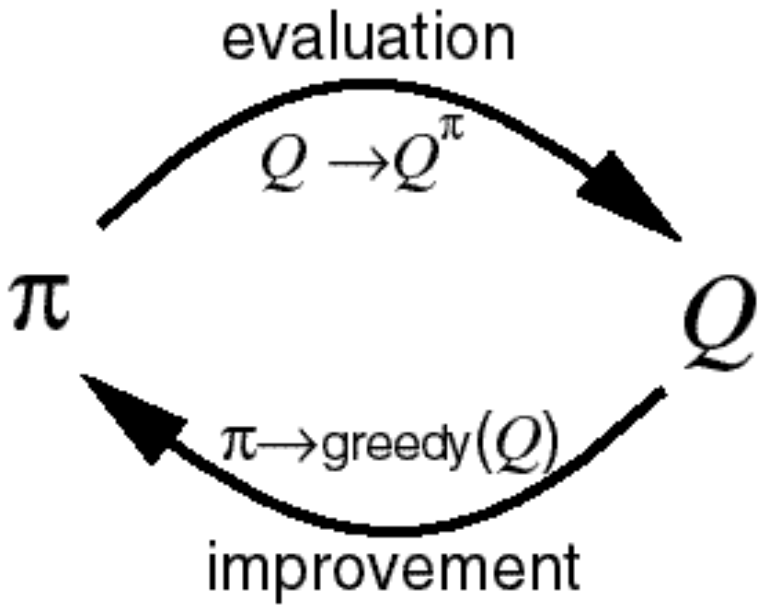
Monte-Carlo policy evaluation

- Unlike dynamic programming policy evaluation, only one subsequent state at each state is taken into account per operation
- Unlike dynamic programming and TD policy evaluation, the entire episode included: MC does not bootstrap
- Computational and sample costs to estimate $V^\pi(s)$ for one s are independent of $|S|$

From evaluation to optimisation: Q-values

- 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)-pair is visited infinitely often
- Requires explicit exploration of actions not possible under π
- Possible solutions:
  - *Exploring starts*: every (s, a) has a non-zero probability of being the starting pair
  - *Soft policies*: $\pi(s, a) > 0$ for all (s, a)

Optimising policies with Monte-Carlo



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

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:
  (a) Generate an episode using exploring starts and π
  (b) For each pair s, a appearing in the episode:
    $R \leftarrow$ return following the first occurrence of s, a
    Append R to $Returns(s,a)$
    $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
  (c) For each s in the episode:
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$

On-policy Monte-Carlo control

- Avoid exploring starts, use ϵ-greedy policies:
  - Non-greedy actions: $\frac{\epsilon}{|A(s)|}$
  - Greedy action: $1 - \epsilon + \frac{\epsilon}{|A(s)|}$

- *Policy improvement theorem*: any ϵ-greedy policy w.r.t. to Q^π is an improvement over any ϵ-greedy policy π

- Converges to the best ϵ-greedy policy

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
$\pi \leftarrow$ an arbitrary ε-soft policy

Repeat forever:
(a) Generate an episode using π
(b) For each pair s, a appearing in the episode:
$R \leftarrow$ return following the first occurrence of s, a
Append R to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
(c) For each s in the episode:
$a^* \leftarrow \arg\max_a Q(s,a)$
For all $a \in \mathcal{A}(s)$:

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

Off-policy Monte-Carlo control

- Evaluate an *estimation policy* using roll-outs from a *behaviour policy*
- Useful if behaviour policy cannot be changed
- Behaviour policy needs to explore sufficiently
- Allows estimating a deterministic policy while still exploring with behaviour policy
- Use *importance sampling* to re-weight roll-out returns from behaviour policy by the probabilities of them occuring under estimation policy

Importance sampling

- General technique from statistics
- Interested in $E_d[f(x)]$ 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

- Distributions d and d' are known but f is unknown
- Importance sampling is based on the following observation:

$$E_d[f(x)] = \sum_x f(x)d(x) = \sum_x f(x)\frac{d(x)}{d'(x)}d'(x) = E_{d'}[f(x)\frac{d(x)}{d'(x)}]$$

- This leads to the *importance sampling estimator*:

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

- In our case: target distribution comes from estimation policy; source distribution from behaviour policy
- We are estimating $V^\pi(s)$ from π' (which is a distribution over actions given a state)

Off-policy Monte-Carlo control

- 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:

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

Off-policy Monte-Carlo control

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

$Q(s,a) \leftarrow$ arbitrary

$N(s,a) \leftarrow 0$; Numerator and

$D(s,a) \leftarrow 0$; Denominator of $Q(s,a)$

$\pi \leftarrow$ an arbitrary deterministic policy

Repeat forever:

(a) Select a policy π' and use it to generate an episode:

$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$

(b) $\tau \leftarrow$ latest time at which $a_\tau \neq \pi(s_\tau)$

(c) For each pair s, a appearing in the episode at time τ or later:

$t \leftarrow$ the time of first occurrence of s, a such that $t \geq \tau$

$w \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k, a_k)}$

$N(s,a) \leftarrow N(s,a) + wR_t$

$D(s,a) \leftarrow D(s,a) + w$

$Q(s,a) \leftarrow \frac{N(s,a)}{D(s,a)}$

(d) For each $s \in \mathcal{S}$:

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

Incremental implementation

Monte-Carlo methods can process each return incrementally:

$$V_{n+1} = V_n + \frac{w_{n+1}}{W_{n+1}}[R_{n+1} - V_n]$$

where:

$$w_{n+1} = \frac{p_{n+1}(s)}{p'_{n+1}(s)}$$

and:

$$W_{n+1} = W_n + w_{n+1}$$

Reinforcement Learning $\rightarrow$ Planning

- Let's take a step back from RL, and have a look at Monte-Carlo methods for *planning*.

Monte Carlo Tree Search

- One of the most successful AI algorithms
- The first reasonably good AI go players (2006)
- AlphaGo beats world top Go players (2016)
- Combination of:
  - Tree Search
  - Monte Carlo roll-outs
  - Exploration heuristics (upper confidence bounds)

S. Gelly, Y. Wang, R. Munos, and O. Teytaud. Modification of UCT with patterns in Monte-Carlo Go. 2006.

Silver, D., et alii (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489.

Monte Carlo Tree Search: other successes

- General Video Game playing
- `http://www.gvgai.net/` — The General Video Game AI Competition
- Partially observable MDP planning (POMCP)
- Multi-agent planning (even partially observable)
- Bayesian Reinforcement learning

Frederik Frydenberg, Kasper R. Andersen, Sebastian Risi, and Julian Togelius. "Investigating MCTS modifications in general video game playing." In Computational Intelligence and Games (CIG), 2015 IEEE Conference on, pp. 107-113. IEEE, 2015.

Silver, D., and Veness, J. 2010. Monte-Carlo planning in large POMDPs. In NIPS 23.

Christopher Amato and Frans A. Oliehoek. "Scalable Planning and Learning for Multiagent POMDPs." In AAAI, pp. 1995-2002. 2015.

Sammie Katt, Frans A. Oliehoek, and Christopher Amato. "Learning in POMDPs with Monte Carlo Tree Search." In International Conference on Machine Learning, pp. 1819-1827. 2017.

Monte Carlo Tree Search: overview

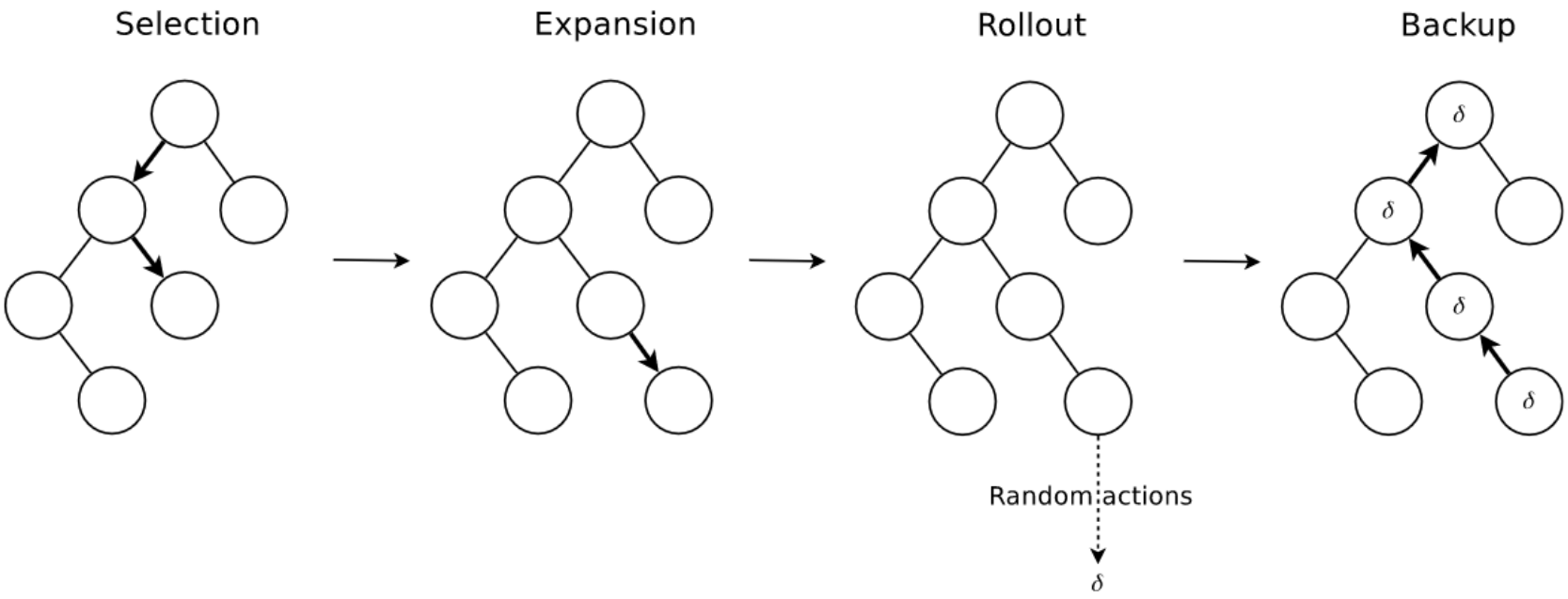


Figure: assumes deterministic transitions, for convenience. (What would be different for stochastic transitions?)

Picture by Maarten de Waard

Monte Carlo Tree Search: overview

- Find expandable state-node (i.e., not all actions have been tried yet)
- Select action to try
- Selection strategy: e.g., UCT (based on bandit literature), game-specific (e.g., crazy stone for Go).
- Observe next state and add to tree as node
- From that node, do random roll-out(s)
- Backup the expected return through the tree

Monte Carlo Tree Search: Recent application

En dan nu... een filmpje!!!

Claes, D., Oliehoek, F., Baier, H., and Tuyls, K. (2017). Decentralised Online Planning for Multi-Robot Warehouse Commissioning. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (pp. 492-500). International Foundation for Autonomous Agents and Multiagent Systems.