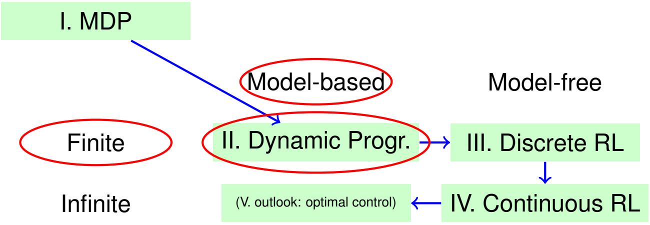
AIC/RL – Discrete Reinforcement Learning (Part III)
Monte Carlo methods

Freek Stulp

Université Paris-Saclay

Where are we?



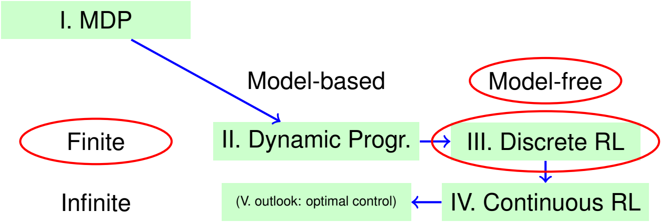
Dynamic Programming (Part II) in three formulae

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

$$\pi(s) = \operatorname*{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s') \right] \qquad \text{Policy improvement} \qquad (2)$$

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right] \qquad \text{Value iteration} \qquad (3)$$

Where are we?


I. MDP
Model-based
Model-free
Finite
II. Dynamic Progr.
III. Discrete RL
Infinite
(V. outlook: optimal control)
IV. Continuous RL

Model-based vs. Model-free

- Environment as an MDP: $\{S, A, \mathcal{P}, \mathcal{R}\}$
  - S Possible states
  - A Possible actions
  - $\mathcal{P}$ Transition function
  - $\mathcal{R}$ Reward function

Model-based

- Agent knows $\mathcal{P}$ and $\mathcal{R}$
  - Can use $\mathcal{P}/\mathcal{R}$ to compute values
  - Dynamic Programming
- Compute values "in your head"

Model-free

- Agent does not know $\mathcal{P}$ and/or $\mathcal{R}$
  - Cannot do Dyn. Prog.
    $V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right]$
- Estimate values from direct interaction with the environment
  - Only rely on actual observations
  - What you do influences what you see

Model-based vs. Model-free

Important topics in Part III

- Update values from observations...
  - **Monte-Carlo**: from returns (i.e. at end of each episode)
  - **Temporal Differencing**: from immediate rewards (i.e. after each action)
- Learning **state/action values** Q instead of state values V
- Should I learn better values, or exploit the ones I have?
  - **Exploration/Exploitation trade-off**

Model-based

- Agent knows $\mathcal{P}$ and $\mathcal{R}$
  - Can use $\mathcal{P}/\mathcal{R}$ to compute values
  - Dynamic Programming
- Compute values "in your head"

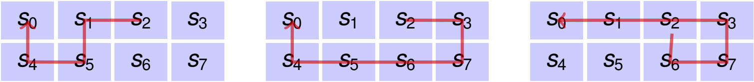
Model-free

- Agent does not know $\mathcal{P}$ and/or $\mathcal{R}$
  - Cannot do Dyn. Prog.

    $V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s') \right]$
- Estimate values from direct interaction with the environment
  - Only rely on actual observations
  - What you do influences what you see

Monte Carlo Policy Evaluation (SuBa5.1)

- Example: random policy, deterministic gridworld, start in s_2



$R(s_2)^1 = 97$ $R(s_2)^2 = 95$ $R(s_2)^3 = 95$

- Given these episodes, what is $V^{\pi}(s_2) = \mathrm{E}_{\pi}\left\{R_t | s_t = s_2\right\}$?
  - Even if we don't have $\mathcal{P}$ or $\mathcal{R}$, we can estimate it given these observations!
- Use arithmetic mean: average of observed returns so far

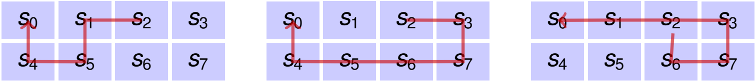
$$V^{\pi}(s_2) = \frac{1}{N}\sum_{e=1}^{N} R(s_2)^e \tag{4}$$

$$= \frac{1}{3}(97 + 95 + 95) \tag{5}$$

$$= 95.67 \tag{6}$$

Monte Carlo Policy Evaluation (SuBa5.1)

- Example: random policy, deterministic gridworld, start in s_2



$R(s_2)^1 = 97$ $\quad$ $R(s_2)^2 = 95$ $\quad$ $R(s_2)^3 = 95$

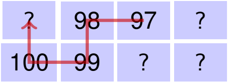
Initialize:
 $\pi \leftarrow$ policy to be evaluated
 $V \leftarrow$ an arbitrary state-value function
 $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

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

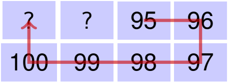
Figure : Monte Carlo Policy Evaluation (first-visit MC)

Monte Carlo Policy Evaluation (SUBA5.1)

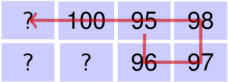
- Example: random policy, deterministic gridworld, start in s_2



$R(s_2)^1 = 97$



$R(s_2)^2 = 95$



$R(s_2)^3 = 95$

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

Figure : Monte Carlo Policy Evaluation (first-visit MC)

Monte Carlo Policy Evaluation (SuBa5.1)

- Algorithmic considerations (useful during TD)
  - Need to store one list of returns for each state, i.e. *Returns*(s)
  - During an episode: store states visited and rewards received

```
Initialize:
    π ← policy to be evaluated
    V ← an arbitrary state-value function
    Returns(s) ← an empty list, for all s ∈ S

Repeat forever:
    (a) Generate an episode using π
    (b) For each state s appearing in the episode:
            R ← return following the first occurrence of s
            Append R to Returns(s)
            V(s) ← average(Returns(s))
```

Figure : Monte Carlo Policy Evaluation (first-visit MC)

Monte Carlo Policy Evaluation (SuBa5.1)

- Algorithmic considerations (useful during TD)
  - Need to store one list of returns for each state, i.e. *Returns*(s)
  - During an episode: store states visited and rewards received

$$
\begin{array}{rccccc}
t = & 1 & 2 & 3 & 4 & 5 \\
s_t = & s^2 & s^1 & s^5 & s^4 & s^0 \\
r_t = & \text{-1} & \text{-1} & \text{-1} & 100 & \text{T} \\
R(s_t) = & 97 & 98 & 99 & 100 & \text{T}
\end{array}
$$

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

Figure : Monte Carlo Policy Evaluation (first-visit MC)

V-values and policies

Remember Dynamic Programming?

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right] \qquad \text{Policy evaluation} \qquad (4)$$

$$\pi(s) = \operatorname{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right] \qquad \text{Policy improvement} \qquad (5)$$

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right] \qquad \text{Value iteration} \qquad (6)$$

- Previous slide: model-free policy evaluation with Monte-Carlo estimation
  - Now how about improving the policy with $\operatorname{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} V_k(s')$
  - Won't work... we do not know $\mathcal{P}^a_{ss'}$

s_0	s_1	s_2	s_3
s_4	s_5	s_6	s_7

$\operatorname{argmax}_a \sum_{s'} \mathcal{P}^a_{s^2 s'} V_k(s') =$LEFT

T	100	99	98
100	99	98	97

$V^\pi(s)$

V-values and policies

Remember Dynamic Programming?

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right] \quad \text{Policy evaluation} \quad (4)$$

$$\pi(s) = \operatorname{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s') \right] \quad \text{Policy improvement} \quad (5)$$

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right] \quad \text{Value iteration} \quad (6)$$

- Previous slide: model-free policy evaluation with Monte-Carlo estimation
  - Now how about improving the policy with $\operatorname{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} V_k(s')$
  - Won't work... we do not know $\mathcal{P}^a_{ss'}$

s_4	s_0	s_1	
s_5		s_2	s_3
s_6	s_7		

$\operatorname{argmax}_a \sum_{s'} \mathcal{P}^a_{s_2 s'} V_k(s') = $UP

100	T	100	
99		99	98
98	97		

$V^{\pi}(s)$

V-values and policies

Consequence of model-free RL

Even if we know the (optimal) value function
we cannot compute the optimal policy because
we don't know which action takes us to the state with the best value

- Previous slide: model-free policy evaluation with Monte-Carlo estimation
  - Now how about improving the policy with $\text{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} V_k(s')$
  - Won't work... we do not know $\mathcal{P}^a_{ss'}$

s_4	s_0	s_1	
s_5		s_2	s_3
s_6	s_7		

$\text{argmax}_a \sum_{s'} \mathcal{P}^a_{s^2 s'} V_k(s') = \text{UP}$

100	T	100	
99		99	98
98	97		

$V^\pi(s)$

V-values and policies

Consequence of model-free RL

Even if we know the (optimal) value function
we cannot compute the optimal policy because
we don't know which action takes us to the state with the best value

Bummer! That's the end of this course. Goodbye.

- Previous slide: model-free policy evaluation with Monte-Carlo estimation
  - Now how about improving the policy with $\text{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} V_k(s')$
  - Won't work... we do not know $\mathcal{P}^a_{ss'}$

s_4	s_0	s_1	
s_5		s_2	s_3
s_6	s_7		

$\text{argmax}_a \sum_{s'} \mathcal{P}^a_{s^2 s'} V_k(s') = \text{UP}$

100	T	100	
99		99	98
98	97		

$V^\pi(s)$

Solution: state/action value $Q(s,a)$ (SUBA3.7)

- State value $V^\pi(s) = \mathrm{E}_\pi\{R_t | s_t = s\}$
  - "expected return when starting in s, and following π thereafter"
- State/action value $Q^\pi(s,a) = \mathrm{E}_\pi\{R_t | s_t = s, a_t = a\}$
  - "exp. return starting from s, taking the action a, and following π thereafter"
  - informal: takes one decision away from the policy

How can we acquire $Q^\pi(s,a)$?

- Model-based Dynamic Programming: compute $V^\pi(s)$, then $Q^\pi(s,a)$ is

$$Q^\pi(s,a) = \mathrm{E}_\pi\{r_{t+t} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a\} \tag{4}$$

$$= \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}_{ss'} + \gamma V^\pi(s')\right] \tag{5}$$

- Model-free Monte Carlo: also store actions during an episode

$t =$	1	2	3	4	5
$s_t =$	s^2	s^1	s^5	s^4	s^0
$a_t =$	LEFT	DOWN	LEFT	UP	T
$r_t =$	-1	-1	-1	100	T
$R(s_t) =$	97	98	99	100	T

  - then $Q^\pi(s,a)$ is average of $Returns(s,a)$

Solution: state/action value $Q(s,a)$ (SuBa3.7)

- State value $V^{\pi}(s) = \mathrm{E}_{\pi}\{R_t | s_t = s\}$
  - "expected return when starting in s, and following π thereafter"
- State/action value $Q^{\pi}(s,a) = \mathrm{E}_{\pi}\{R_t | s_t = s, a_t = a\}$
  - "exp. return starting from s, taking the action a, and following π thereafter"
  - informal: takes one decision away from the policy

- We've acquired $Q(s,a)$, great.
- But why is it a solution to the problem of not being able to choose the best action in model-free RL?

Solution: state/action value $Q(s,a)$ (SuBa3.7)

- State value $V^{\pi}(s) = \mathrm{E}_{\pi}\{R_t | s_t = s\}$
  - "expected return when starting in s, and following π thereafter"
- State/action value $Q^{\pi}(s,a) = \mathrm{E}_{\pi}\{R_t | s_t = s, a_t = a\}$
  - "exp. return starting from s, taking the action a, and following π thereafter"
  - informal: takes one decision away from the policy

$V^{\pi}(s)$ for random policy (rounded!)

T	85	76	72
89	82	75	71

Solution: state/action value $Q(s,a)$ (SuBa3.7)

- State value $V^\pi(s) = \mathrm{E}_\pi\{R_t | s_t = s\}$
  - "expected return when starting in s, and following π thereafter"
- State/action value $Q^\pi(s,a) = \mathrm{E}_\pi\{R_t | s_t = s, a_t = a\}$
  - "exp. return starting from s, taking the action a, and following π thereafter"
  - informal: takes one decision away from the policy

$V^\pi(s)$ for random policy (rounded!)

T	85	76	72
89	82	75	71

choose action?

T	?	?	?
?	?	?	?

Solution: state/action value $Q(s,a)$ (SuBA3.7)

- State value $V^{\pi}(s) = \mathrm{E}_{\pi}\{R_t | s_t = s\}$
  - "expected return when starting in s, and following π thereafter"
- State/action value $Q^{\pi}(s,a) = \mathrm{E}_{\pi}\{R_t | s_t = s, a_t = a\}$
  - "exp. return starting from s, taking the action a, and following π thereafter"
  - informal: takes one decision away from the policy

$V^{\pi}(s)$ for random policy (rounded!)

T	85	76	72
89	82	75	71

$Q(s, UP)$

T	84	75	71
100	84	75	71

$Q(s, LEFT)$

T	**100**	**84**	**75**
88	**88**	**81**	**74**

$Q(s, RIGHT)$

T	75	71	71
81	74	70	70

$Q(s, DOWN)$

T	81	74	70
88	81	74	70

Solution: state/action value $Q(s,a)$ (SUBA3.7)

- State value $V^{\pi}(s) = \mathrm{E}_{\pi}\{R_t | s_t = s\}$
  - "expected return when starting in s, and following π thereafter"
- State/action value $Q^{\pi}(s,a) = \mathrm{E}_{\pi}\{R_t | s_t = s, a_t = a\}$
  - "exp. return starting from s, taking the action a, and following π thereafter"
  - informal: takes one decision away from the policy

$V^{\pi}(s)$ for random policy (rounded!)

T	85	76	72
89	82	75	71

$Q(s, UP)$

T	84	75	71
100	84	75	71

$Q(s, LEFT)$

T	**100**	**84**	**75**
88	**88**	**81**	**74**

$Q(s, RIGHT)$

T	75	71	71
81	74	70	70

$Q(s, DOWN)$

T	81	74	70
88	81	74	70

$\mathrm{argmax}_a\, Q(s,a)$

T	$<$	$<$	$<$
$\wedge$	$<$	$<$	$<$

Summary so far

Problem: In model-free RL without $\mathcal{P}^a_{ss'}$, Dynamic Programming not possible

Solution: Use actually observed returns to estimate values

- Monte Carlo: $V^\pi(s)$ is average of $Returns(s)$, which are observed returns

Problem: Even with $V^\pi(s)$, we don't have $\mathcal{P}^a_{ss'}$ to improve our policy

Solution: Learn $Q^\pi(s,a)$ instead, e.g. with Monte Carlo

- choose $\text{argmax}_a\, Q(s,a)$ to improve your policy

Next problem...

- Dumb strategy I: always choose random action
  - Your estimations of $\text{argmax}_a\, Q^\pi(s,a)$ become good, but you never use them to improve your policy
- Dumb strategy II: choose $\text{argmax}_a\, Q^\pi(s,a)$ from the beginning
  - But you won't have learned the right Q-values yet!
- "Exploration/Exploitation Trade-off"

Exploration/Exploitation Trade-off: finding your favourite restaurant

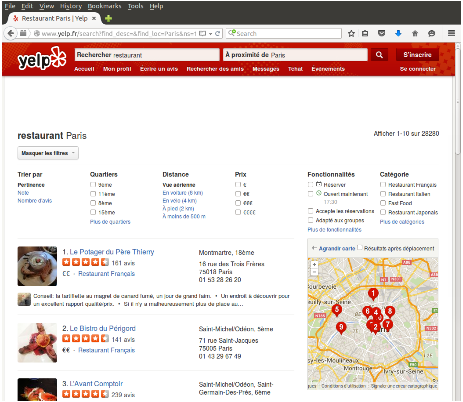


Figure : >28000 restaurants in Paris

Exploration/Exploitation Trade-off

- Pure exploration strategy
  - try ALL restaurants in Paris (takes ≈80 years, no time to choose the best...)
- Pure greedy exploitation strategy
  - go to first restaurant in list; never go anywhere else (may be a bad one...)
- A mixture of both: ϵ-greedy
  - Try your favourite restaurant so far with probabily $1 - \epsilon$
  - Try a new restaurant with probabily ϵ

$$\pi(s) = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \text{argmax}_a\, Q(s,a) & \text{with probability } 1 - \epsilon \end{cases} \tag{4}$$

- Common strategy
  - intialize ϵ to 1 (pure exploration)
  - decay it after each episode with factor $0 < \beta < 1$, i.e. $\epsilon \leftarrow \beta\epsilon$

Exploration/Exploitation Trade-off

"ϵ-greedy with decay" is typical strategy in human life

Student: $\epsilon = \beta^{20} \cdot 1 \approx 0.5$

Elderly: $\epsilon = \beta^{90} \cdot 1 \approx 0.0$

$$\pi(s) = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \text{argmax}_a\, Q(s,a) & \text{with probability } 1-\epsilon \end{cases} \tag{4}$$

- Common strategy
  - intialize ϵ to 1 (pure exploration)
  - decay it after each episode with factor $0 < \beta < 1$, i.e. $\epsilon \leftarrow \beta\epsilon$

Putting it all together

RL Superman: uses Monte-Carlo to estimate state-action values using ϵ-greedy exploration with decaying ϵ

- Why super?
  - Learns Q-values from observed returns (doesn't require a model)
  - Estimates become better over time with more experience
  - Can choose the best action as $\text{argmax}_a\, Q^\pi(s,a)$
  - Starts out with exploration ($\epsilon = 1$), but slowly becomes greedy ($\epsilon \approx 0$)
- But...
  - requires a lot of experience to get good estimates and policy
  - works for small finite MDPs only
  - applicable to episodic problems only
  - wastes time evaluating bad policies
- Solution: Temporal Difference Learning (next course!)

Up next in the next exercise

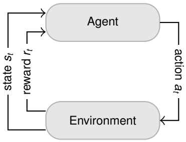


Figure : Agent-environment interface

- Code provided
  - `environments.Environment` → `EnvMaze`, `EnvMDP`
    - represents the environment
    - environment can contain an MDP
  - `agents.Agent` → `AgentRandom`
    - model-free: agent never has access to `MarkovDecisionProcess` !
  - `experiment_episodic.py`: communication between agent and environment
- Your aim: implement several model-free agents in `agents/` package
  - `environments/` and `experiment_episodic.py` require little/no changes