

# Reinforcement Learning

## Part II: Markov Decision Process

Michele Sebag – Herilalaina Rakotoarison  
CNRS – INRIA – Univ Paris-Saclay

Orsay – Nov 25th. 2019

*Credit for slides: Rich Sutton, Freek Stulp, Olivier Pietquin*



## Markov Decision Process

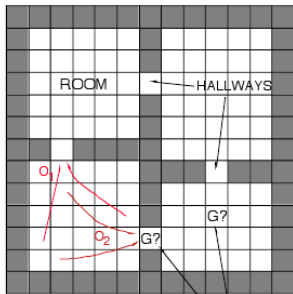
### Dynamic Programming

- Temporal differences and eligibility traces

- Q-learning

- Partial summary

# Ingredients

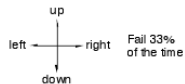


Goal states are given a terminal value of 1

4 rooms

4 hallways

4 unreliable primitive actions



8 multi-step options  
(to each room's 2 hallways)

Given goal location,  
quickly plan shortest route

All rewards zero  
 $\gamma = .9$

## Issues

- ▶ How does the world behave ?
- ▶ How does the agent behave ?
- ▶ What is the goal

- ▶ Markov Decision Process (S,A,p,r)
- ▶ Policy  $\pi : S \mapsto A$
- ▶ Optimize expected cumulative rewards

# Markov Decision Process

- ▶ State space  $S$  Terminal states  $T \subset S$
- ▶ Action space  $A$
- ▶ Transition  $p(s, a, s')$  : probability of arriving in  $s'$  after doing  $a$  in  $s$
- ▶ Reward  $r(s, a)$ : goodies for doing  $a$  in  $s$   
sometimes,  $r(s)$ : just for being in  $s$

## Markov property

Future only depends upon current state

## Remark

This can always hold.

But ?

# Markov Decision Process

- ▶ State space  $S$  Terminal states  $T \subset S$
- ▶ Action space  $A$
- ▶ Transition  $p(s, a, s')$  : probability of arriving in  $s'$  after doing  $a$  in  $s$
- ▶ Reward  $r(s, a)$ : goodies for doing  $a$  in  $s$   
sometimes,  $r(s)$ : just for being in  $s$

## Markov property

Future only depends upon current state

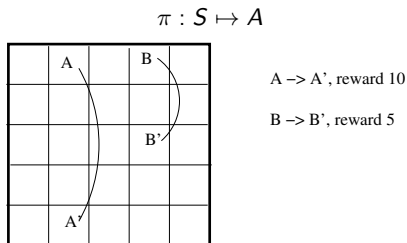
## Remark

This can always hold.

But ?

more expensive

## Policy – Quality



$$s_0, a_0 = \pi(s_0), r_0, s_1, a_1 = \pi(s_1), r_1, s_2, \dots$$

### Episodic

$$R(\pi) = r(s_0) + r(s_1) + \dots + r(s_K)$$

### Continuous

$$R(\pi) = \sum \gamma^{k+1} r(s_k)$$

- ▶  $s_0$  drawn after probability  $p_{Init}$
- ▶  $s_i$  drawn with probability  $p(s_{i-1}, \pi(s_{i-1}, \cdot))$

# Designing an RL problem

## Choices

- ▶ Which state space ?
- ▶ Size of the search space
- ▶ Reward function
- ▶ How unpredictable is the environment (if multiple agents...)
- ▶ Which discount factor ?

## Some problems...

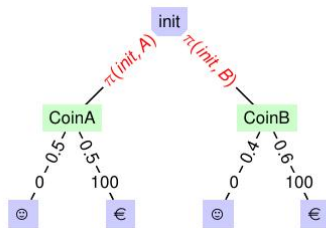
- ▶ Optimal autonomous driving (safe, fast, comfortable)
  - ▶ Optimal trading on the stock-market
  - ▶ Policy that optimizes your happiness during your life
  - ▶ Policy that optimizes long-term happiness of humanity
- Which discount factor ?

## Features of RL problems

- ▶ Finite vs. Infinite
- ▶ Discrete vs. Continuous
- ▶ Model-based vs. Model-free
- ▶ Episodic vs. Continuing
- ▶ Markovian vs. Non-Markovian
- ▶ Observable vs. Partially Observ.



# The coin problem



## Compute return

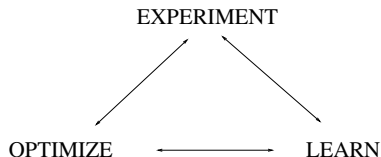
- ▶ Random policy

## Which optimal policy ?

# The global RL problem

## 3 interleaved tasks

- ▶ Learn a world model ( $p, r$ )
- ▶ Decide/select (the best) action
- ▶ Explore the world



# Milestones

**MDP** Main Building block

**General settings**

	Model-based	Model-free
Finite	<b>Dynamic Programming</b>	Discrete RL
Infinite	(optimal control)	Continuous RL

## Markov Decision Process

### Dynamic Programming

- Temporal differences and eligibility traces

- Q-learning

- Partial summary

# Algorithmic paradigms

## Greedy optimization

Define incrementally a solution, based on myopic optimization of some criterion

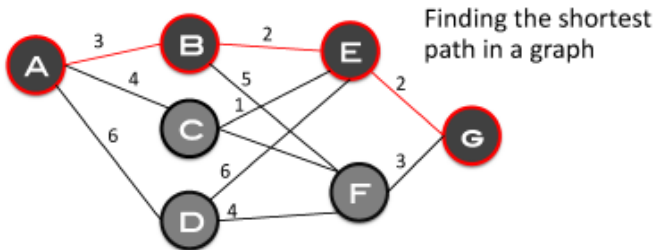
## Divide and conquer

- ▶ Define subproblems
- ▶ Find optimal solutions for subproblems
- ▶ Combine solutions to subproblems

## Dynamic programming

- ▶ Recursively decompose the problem in subproblems
- ▶ Bottom-up: solve small sub-problems
- ▶ Assemble their solutions to solve larger subproblems
- ▶ Can be close to brute-force search.

## Dynamic programming, an example



### Algorithm

recursion + memoization

- ▶  $D(N, N') = \infty$
- ▶  $D(N, N) = 0$
- ▶  $D(N, N') = c(\text{Edge}(N, N'))$  iff it exists
- ▶ Repeat until no change

- ▶ If  $D(N_1, N_2) > D(N_1, N_3) + D(N_3, N_2)$ ,

$$D(N_1, N_2) = D(N_1, N_3) + D(N_3, N_2)$$

Exo: Computational complexity ?

# Dynamic Programming

Bellman, 50s

## Context

- ▶ Computer Science: theory, AI, graphics,
- ▶ Information theory
- ▶ Control theory
- ▶ BioInformatics
- ▶ Operation research

## Algorithms

- ▶ Viterbi for Hidden Markov Models
- ▶ Smith-Waterman for sequence alignment
- ▶ diff in Unix for comparing two files
- ▶ Bellman-Ford for shortest paths in graphs

# Value function

## Intuition

- ▶ What is the value of being in a state ?
- ▶ The value is good if this state is associated to a (delayed) reward

## Caveat

- ▶ The value depends on the state
- ▶ The value depends on the policy
- ▶  $V_{\pi}(s)$  is the expected cumulative reward when starting in  $s$  and following  $\pi$

## Observation

$$\begin{aligned}R_t &= r_0 + \gamma r_1 + \dots + \gamma^k r_k + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k r_k\end{aligned}$$

## Expectation

$$V_{\pi}(s) = \mathbb{E}[R_t | s_0 = s]$$



## Bellman equation

$$\begin{aligned}V_{\pi}(s) &= \mathbb{E}[R_t | s_0 = s] \\&= \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_k | s_0 = s] \\&= \mathbb{E}[r(s)] + \mathbb{E}[\sum_{k=1}^{\infty} \gamma^k r_k | s_0 = s] \\&= \mathbb{E}[r(s)] + \sum_{s'} p(s, \pi(s), s') \mathbb{E}[\sum_{k=1}^{\infty} \gamma^k r_k | s_1 = s'] \\&= \mathbb{E}[r(s)] + \gamma \sum_{s'} p(s, \pi(s), s') \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_k | s_0 = s'] \\&= \mathbb{E}[r(s)] + \gamma \sum_{s'} p(s, \pi(s), s') V(s')\end{aligned}$$

## Bellman equation

- ▶ A theoretical property of value functions
- ▶ Optimal Bellman equation

Define

$$V^*(s) = \max_{\pi} V_{\pi}(s)$$

Then,  $\pi^*$  is an optimal policy if and only if

$$V_{\pi^*} = V_*$$

$$\pi^*(s) = \arg \max_a p(s, a, s') V_*(s')$$

(What is needed to compute  $\pi^*(s)$  from  $V_*$  ?)

# Policy evaluation

## Truncate at $k$ time steps

$$V_{\pi,k}(s) = \mathbb{E} \left[ \sum_{\ell=1}^k \gamma^{\ell} r_{\ell} \mid s_0 = s \right]$$

$$\lim_{k \rightarrow \infty} V_{\pi,k}(s) = V_{\pi}(s)$$

( $V_{\pi,k}$  is an approximation of  $V_{\pi}$ ; can we bound the approximation error ?)

# Iterative policy evaluation

**Given policy**  $\pi$

**Init**

$$\forall s \in S, V_{\pi}(s) = 0$$

**Loop**

$$\Delta = 0$$

For each

$$s \in S$$

$$v = V(s)$$

$$V(s) = r(s) + \gamma \sum_{s'} p(s, \pi(s), s') V(s')$$

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

**Until**  $\Delta < \epsilon$

**Output**  $V \approx V_{\pi}$

# Policy Improvement

## Intuition

- ▶ Build  $V_{\pi}(s)$
- ▶ You are in  $s$
- ▶ This is the model-based setting
- ▶ Can you think of better than doing  $\pi(s)$  ?

# Policy Improvement

## Intuition

- ▶ Build  $V_\pi(s)$
- ▶ You are in  $s$
- ▶ This is the model-based setting
- ▶ Can you think of better than doing  $\pi(s)$  ?

## Improved $\pi'$

$$\pi'(s) = \arg \max_a \{p(s, a, s') V_\pi(s')\}$$

## Algorithm

1. Define  $\pi$
2. Build  $V_\pi$
3.  $\pi'$ : Policy improvement( $\pi$ )
4.  $\pi = \pi'$ ; Goto 2

This converges toward optimal  $\pi^*$

but takes for ever

# Value Iteration

**Policy evaluation**

recall

$$V_{\pi, k+1}(s) = r(s) + \gamma \sum_{s'} p(s, \pi(s), s') V_{\pi, k}(s')$$

**Value iteration**

more greedy

# Value Iteration

## Policy evaluation

recall

$$V_{\pi, k+1}(s) = r(s) + \gamma \sum_{s'} p(s, \pi(s), s') V_{\pi, k}(s')$$

## Value iteration

more greedy

$$V_{k+1} = r(s) + \gamma \arg \max_a \sum_{s'} p(s, a, s') V_k(s')$$



## Policy evaluation vs Value iteration

	Policy evaluation	Value iteration
Init	$\pi$	$V$
loop	$a = \pi(s)$	$a = \mathit{argmax}$
Output	$V_\pi$	Greedy policy ( $V$ )

# Initialization

## Random ?

- ▶ Educated initialisation is better
- ▶ See Inverse Reinforcement Learning
- ▶ <https://www.youtube.com/watch?v=0JL04JJjocc>
- ▶ <https://www.youtube.com/watch?v=VCdxqn0fcnE>
- ▶ More: ICML 2004, Pieter Abbeel and Andrew Ng

# Policy iteration

## Principle

- ▶ Modify  $\pi$  step 1
- ▶ Update  $V$  until convergence step 2

## Getting faster

- ▶ Don't wait until  $V$  has converged before modifying  $\pi$ .

# Discussion

## Policy and value iteration

- ▶ Must wait until the end of the episode
- ▶ Episodes might be long

## Can we update $V$ on the fly ?

- ▶ I have estimates of how long it takes to go to RER, to catch the train, to arrive at Cité-U
- ▶ Something happens on the way (bump into a friend, chat, delay, miss the train,...)
- ▶ I can update my estimates of when I'll be home...

# TD(0)

1. Initialize  $V$  and  $\pi$
2. Loop on episode
  - 2.1 Initialize  $s$
  - 2.2 Repeat

Select action  $a = \pi(s)$

Observe  $s'$  and reward  $r$

$$V(s) \leftarrow V(s) + \alpha \underbrace{(r + \gamma V(s') - V(s))}_R$$

$s \leftarrow s'$

- 2.3 Until  $s'$  terminal state

## Discussion

### Update on the spot ?

- ▶ Might be brittle
- ▶ Instead one can consider several steps

$$R = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$$

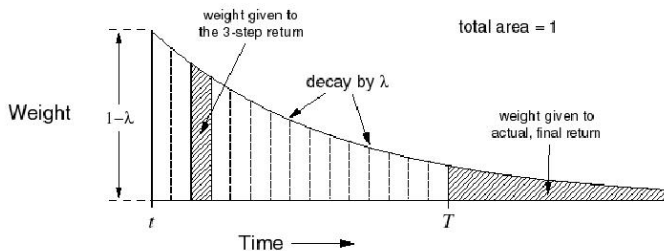
### Find an intermediate between

- ▶ Policy iteration
- ▶ TD(0)

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

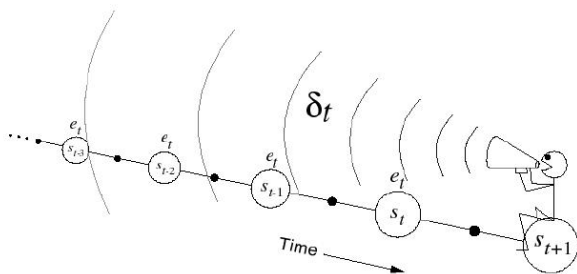
$$R_t = r_{t+1} + \gamma V_t(s_{t+1})$$

## TD( $\lambda$ ), intuition



$$R_t^\lambda = \underbrace{(1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)}}_{\text{weight given to the 3-step return}} + \underbrace{\lambda^{T-t-1} R_T}_{\text{weight given to actual, final return}}$$

## TD( $\lambda$ ), intuition, followed



$$\delta_t = r_{t+1} + \mathcal{W}_t(s_{t+1}) - V_t(s_t)$$



# TD( $\lambda$ )

1. Initialize  $V$  and  $\pi$
2. Loop on episode
  - 2.1 Initialize  $s$
  - 2.2 Repeat

$$a = \pi(s)$$

Observe  $s'$  and reward  $r$

$$\delta \leftarrow r + V(s') - V(s)$$

$$e(s) \leftarrow e(s) + \delta$$

For all  $s''$

$$V(s'') \leftarrow V(s'') + \alpha \delta e(s'')$$

$$e(s'') \leftarrow \gamma \lambda e(s'')$$

$$s \leftarrow s'$$

- 2.3 Until  $s'$  terminal state

# Q-learning

**Principle:** Iterate

- ▶ During an episode (from initial state until reaching a final state)
- ▶ At some point explore and choose another action;
- ▶ If it improves, update  $Q(s, a)$ :

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\left[ \underbrace{\alpha}_{\text{learning rate}} \times \left( \underbrace{r(s_{t+1})}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{max future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right) \right]}_{\text{learned value}}$$

**Equivalent to**

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

# Partial summary

## Strengths

- ▶ Optimality guarantees (converge to global optimum)...

## Weaknesses

- ▶ ...if each state is visited often, and each action is tried in each state
- ▶ Number of states: exponential wrt number of features

# Discussion

## Values and emotions

More: Antonio Damasio. Descartes' Error: Emotion, Reason, and the Human Brain