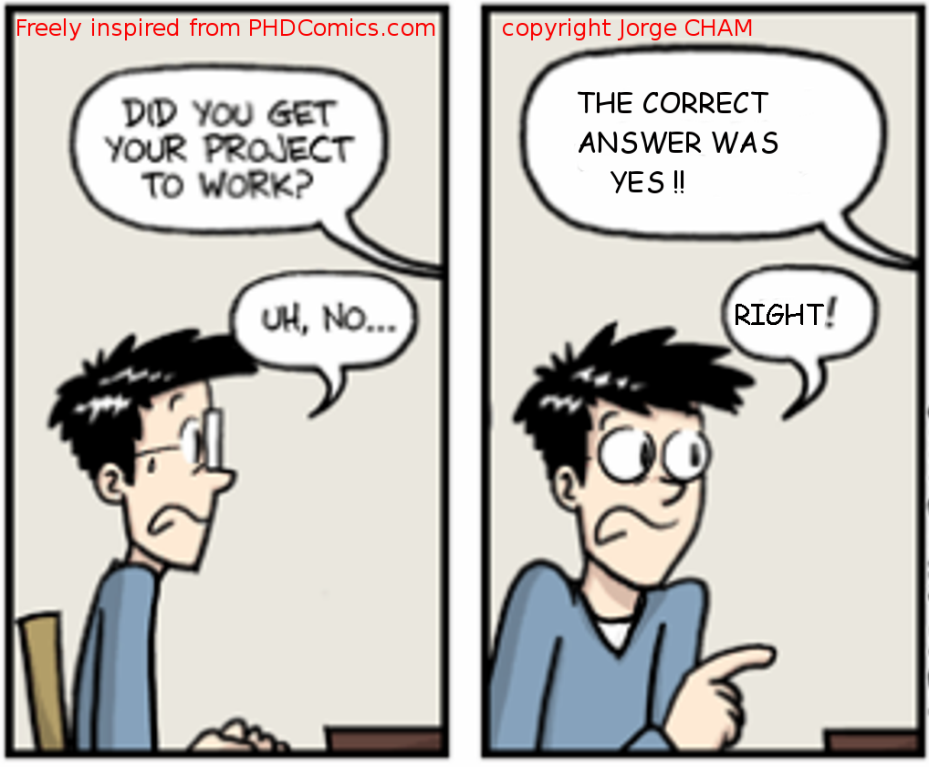
Tabular Reinforcement Learning

Dynamic programming

Olivier Sigaud

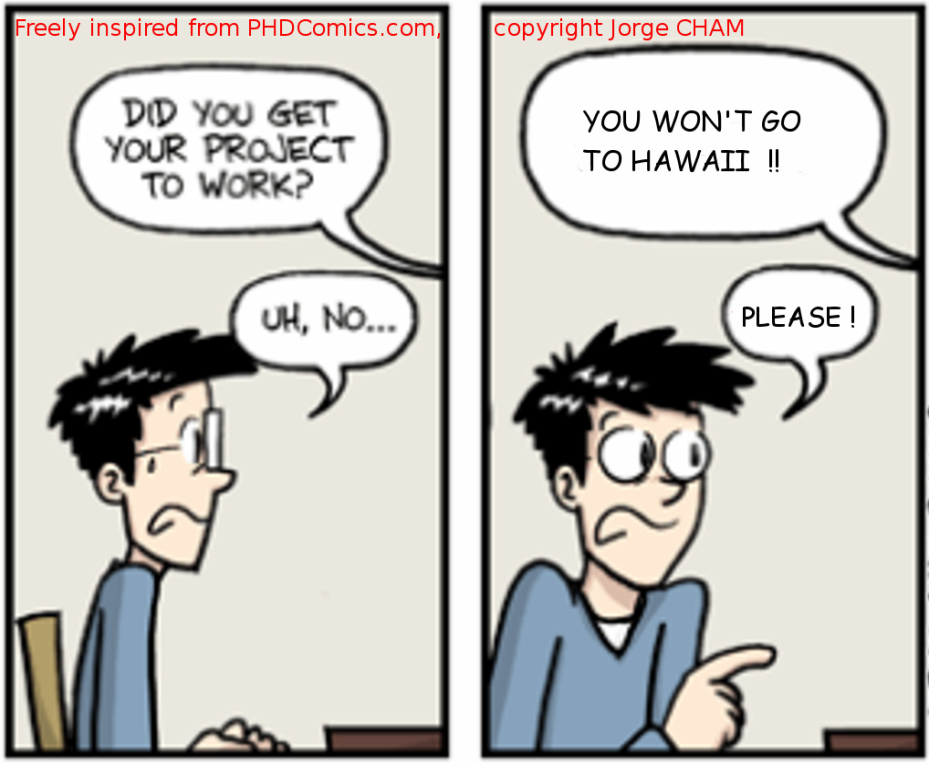
Sorbonne Université
http://people.isir.upmc.fr/sigaud

Supervised learning



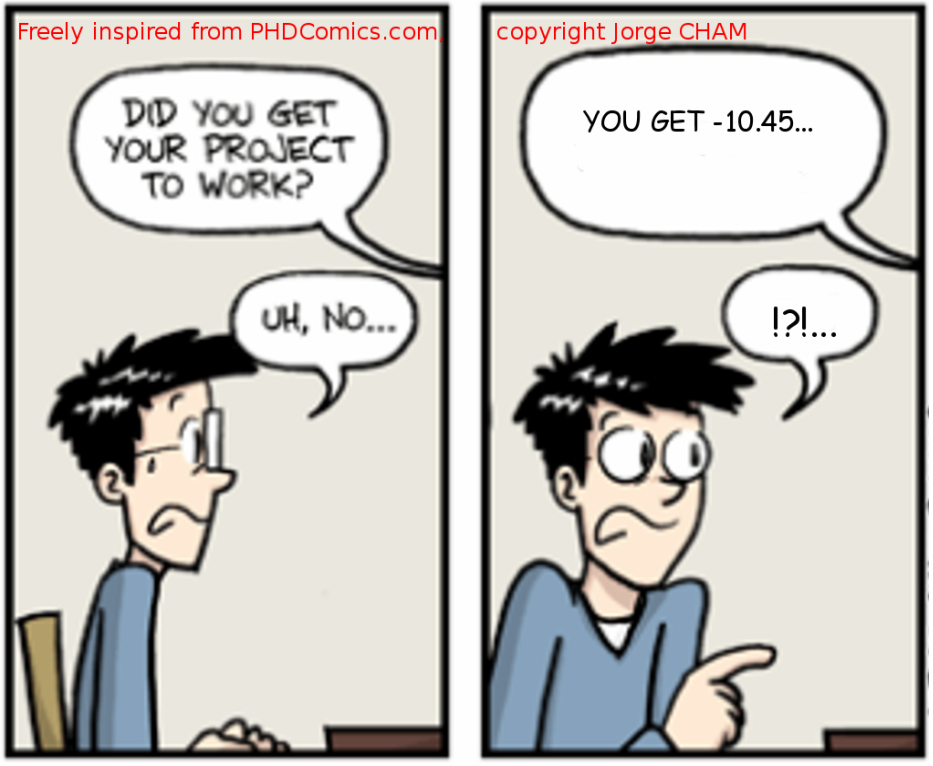
- The supervisor indicates to the agent the expected answer
- The agent corrects a model based on the answer
- Typical mechanism: gradient backpropagation, RLS
- Applications: classification, regression, function approximation...

Cost-Sensitive Learning



- The environment provides the value of action (reward, penalty)
- Application: behaviour optimization

Reinforcement learning



- In RL, the value signal is given as a scalar
- How good is -10.45?
- Necessity of exploration

The exploration/exploitation trade-off

- Exploring can be (very) harmful
- Shall I exploit what I know or look for a better policy?
- Am I optimal? Shall I keep exploring or stop?
- Decrease the rate of exploration along time
- ϵ-*greedy*: take the best action most of the time, and a random action from time to time

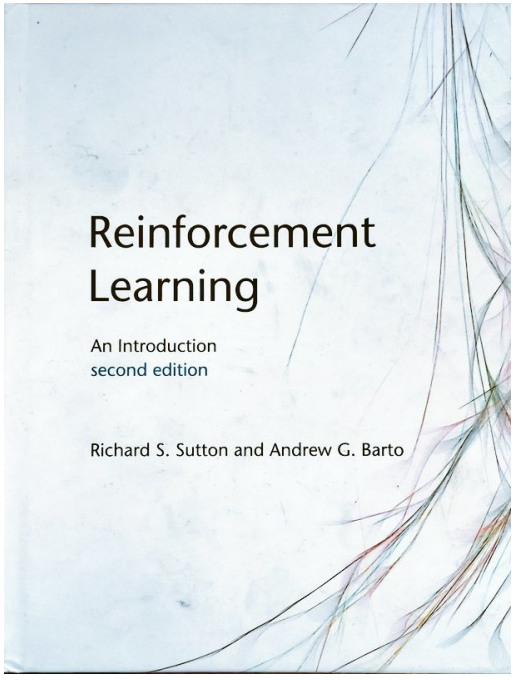
Sequentiality: Bandits problems vs Sequential problems

$\neq$

s_0 0.48	s_3 0.53	0.59	0.66	0.73
s_1 0.43	s_2 0.48	0.53		0.81
0.39		0.48		0.9
0.35	0.39	0.43		1

- A multi-armed bandit problem is a one step/state RL problem where each arm is an action (and reward is stochastic for more fun)
- RL problem are sequential decision making problems
- The state at step $t+1$ depends on the action at state t
- This makes exploration/optimization much more difficult

Introductory book: the bible



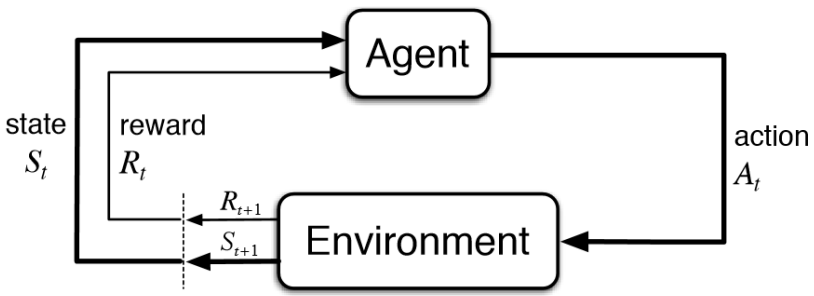
- [Sutton and Barto, 2018]: the ultimate introduction to the field, in the discrete case
- Available online:

  `https://drive.google.com/file/d/1xeUDVGWGUUv1-ccUMAZHJLej2C7aAFWY/view`
- First version in 1998

Sutton, R. S. & Barto, A. G. (2018) *Reinforcement Learning: An Introduction*. MIT Press.

Markov Decision Processes

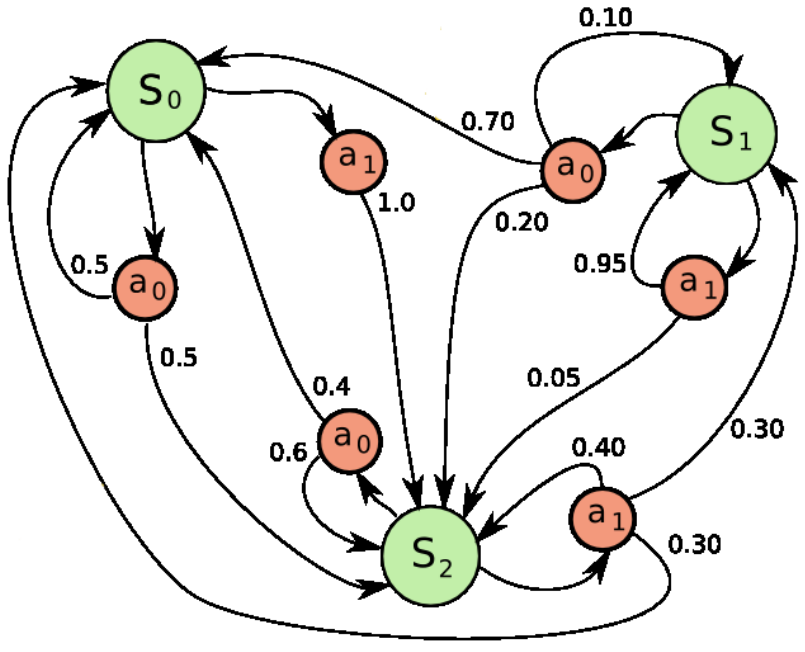
Markov Decision Processes



- S: state space
- A: action space
- $T : S \times A \to \Pi(S)$: transition function
- $r : S \times A \to \mathbb{R}$: reward function

- An MDP describes a problem, not a solution to that problem

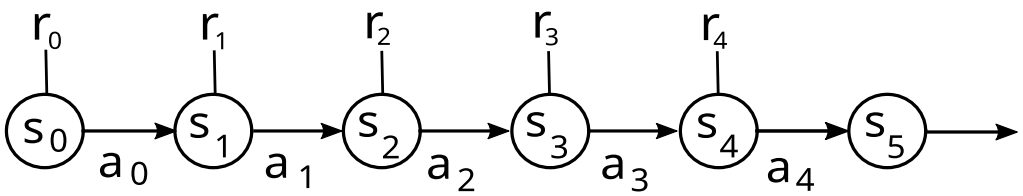
Stochastic transition function



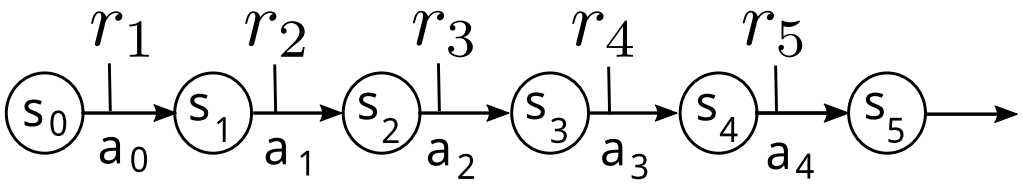
- Deterministic problem = special case of stochastic
- $T(s^t, a^t, s^{t+1}) = p(s'|s, a)$

Rewards: over states or action?

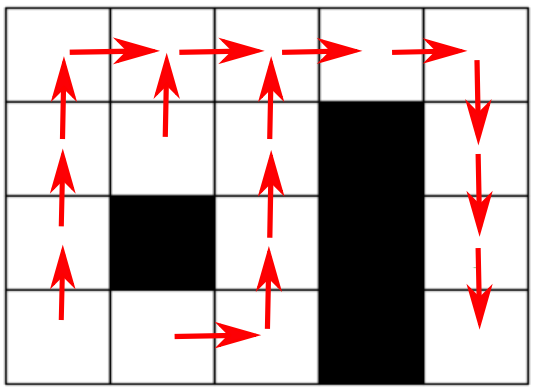
- Reward over states

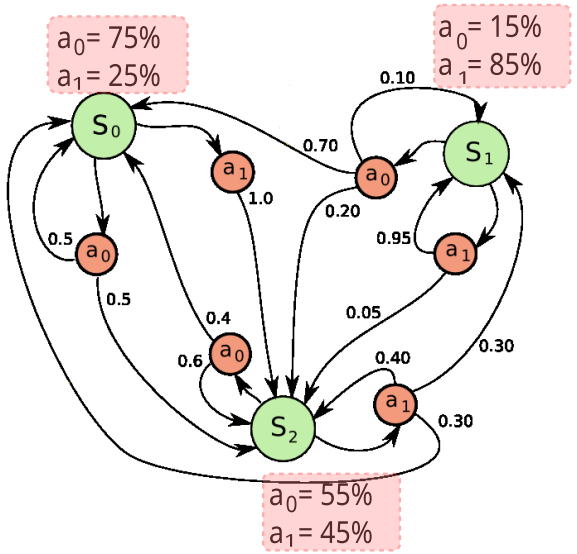


- Reward over actions in states



Deterministic versus stochastic policy

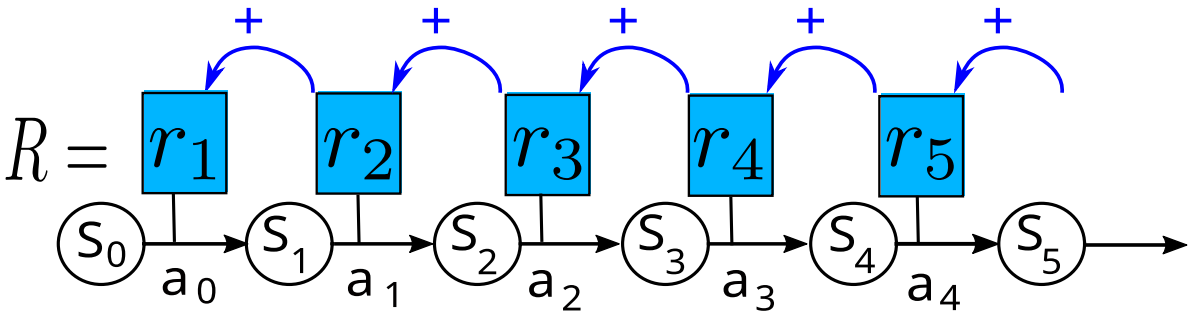




- The agent is driven by a policy
- Goal: find a policy $\pi : S \to A$ maximizing an agregation of rewards (the return R) on the long run
- The policy itself can be deterministic or stochastic
- Important theorem: for any MDP, there exists a deterministic policy that is optimal

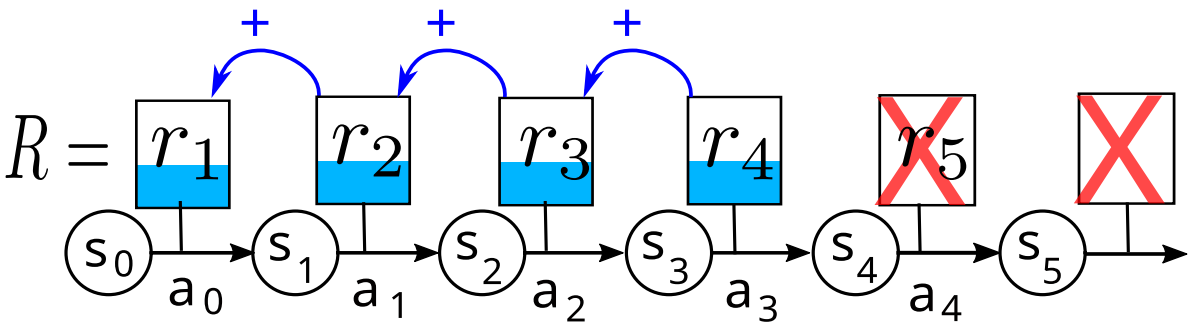
Agregation criterion: mere sum

- The computation of value functions assumes the choice of an agregation criterion (discounted, average, etc.)



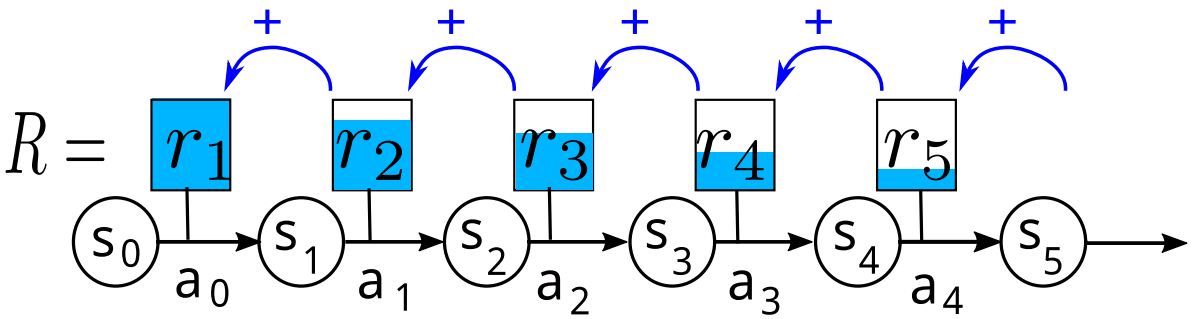
- The sum over a infinite horizon may be infinite, thus hard to compare
- Mere sum (finite horizon N): $V^{\pi}(S_0) = r_1 + r_2 + \ldots + r_N$

Agregation criterion: average over a window



- Average criterion on a window: $V^{\pi}(S_0) = \frac{r_1+r_2+r_3+r_4}{4}$...

Agregation criterion: discounted

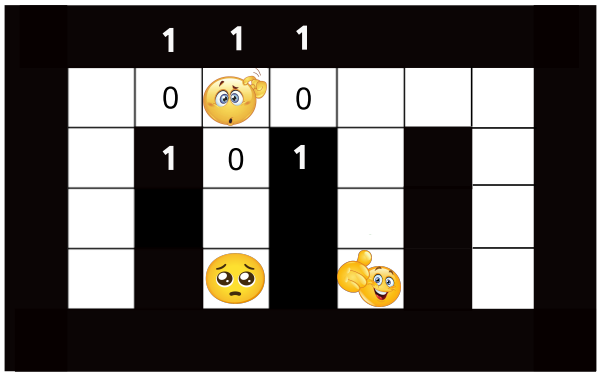


- Discounted criterion: $V^{\pi}(s_{t_0}) = \sum_{t=t_0}^{\infty} \gamma^t r(s_t, \pi(s_t))$
- $\gamma \in [0, 1]$: discount factor
  - if $\gamma = 0$, sensitive only to immediate reward
  - if $\gamma = 1$, future rewards are as important as immediate rewards
- The discounted case is the most used
- The discount factor can be a parameter of the agent or the environment

Markov Property

- A deterministic MDP defines s_{t+1} and r_{t+1} as $f(s_t, a_t)$
- Markov property : $p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...s_0, a_0)$
- In an MDP, a memory of the past does not provide any useful advantage
- Reactive agents $a_{t+1} = f(s_t)$, without internal states nor memory, can be optimal

Markov property: Limitations



- Markov property is not verified if:
  - the observation does not contain all useful information to take decisions (POMDPs)
  - or if the next state depends on decisions of several agents (Dec-MDPs, Dec-POMDPs, Markov games)
  - or if transitions depend on time (Non-stationary problems)

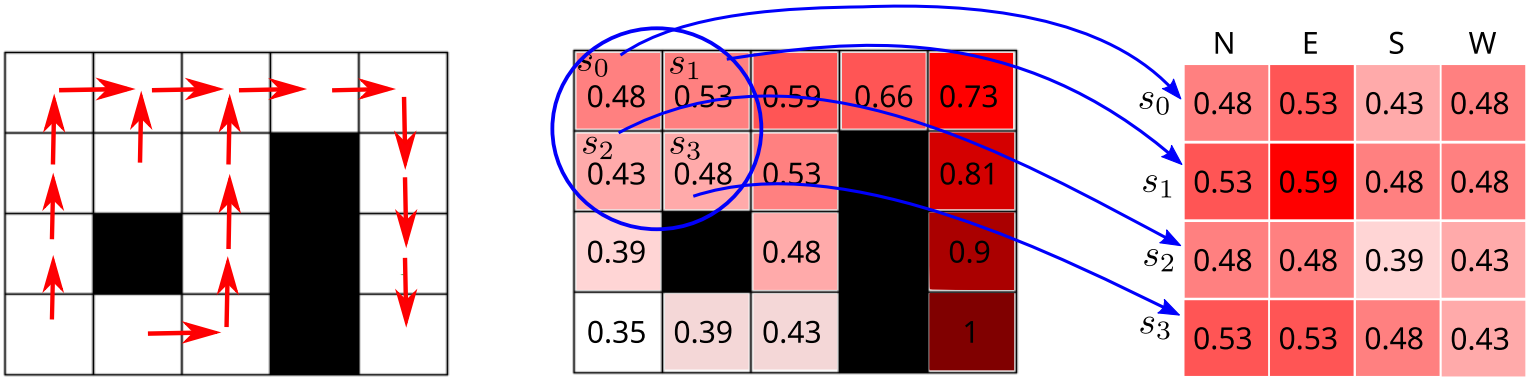
Dynamic Programming

Dynamic Programming

- Once we have defined an MDP
- Dynamic programming methods can find the optimal policy
- Assuming they know everything about the MDP
- Reinforcement Learning applies when the transition and reward functions are unknown
- To define dynamic programming methods, we need value functions

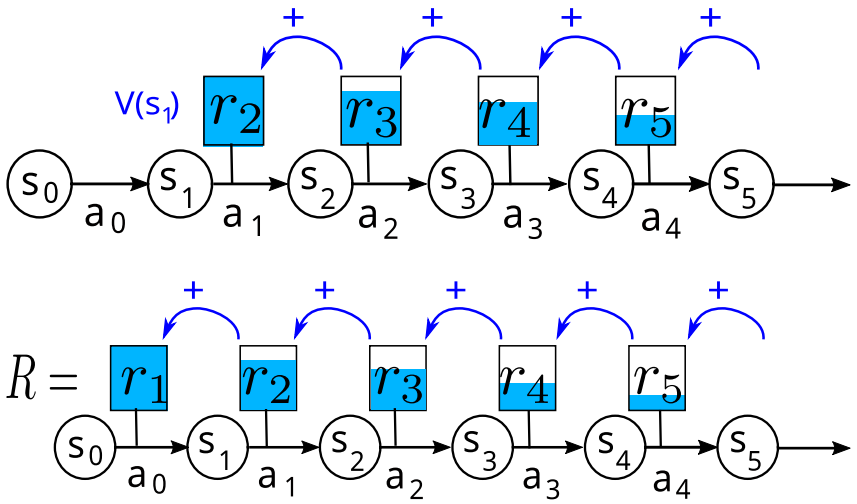
Policy-dependent value and action-value functions

Policy-dependent value and action value functions



- For each state, the value function $V^\pi : S \to \mathbb{R}$ records the return on the long run following policy π.
- $V^\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty}[\gamma^k r_{t+k+1}|s_t = s]$
- It is a vector with one entry per state
- For each (state, action) pair, the action value function $Q^\pi : S \times A \to \mathbb{R}$ records the return on the long run for doing this action in this state and then following policy π.
- $Q^\pi(s,a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty}[\gamma^k r_{t+k+1}|s_t = s, a_t = a]$
- It is a matrix with one entry per state and per action
- In the remainder, we focus on V, trivial to transpose to Q

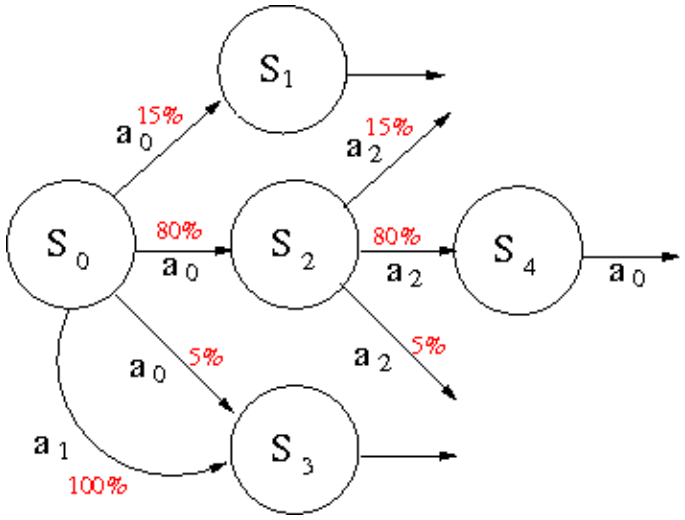
Bellman equation over a Markov chain: recursion



Given the discounted reward agregation criterion:

- $V(s_0) = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + ...$
- $V(s_0) = r_1 + \gamma(r_2 + \gamma r_3 + \gamma^2 r_4 + ...)$
- $V(s_0) = r_1 + \gamma V(s_1)$
- More generally $V(s_t) = r_{t+1} + \gamma V(s_{t+1})$

Bellman equation over a stochastic tree



- Generalisation of $V(s_t) = r_{t+1} + \gamma V(s_{t+1})$ over all possible trajectories
- The expectation of a random variable is the sum of the realizations weighted by their probabilities
- Deterministic π: $V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V^\pi(s')$
  - Realizations are the next states
- Stochastic π: $V^\pi(s) = \sum_a \pi(a|s)[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^\pi(s')]$
  - Realizations are the chosen actions and the next states

Relations between V and Q

$$V^{\pi}(s) = \sum_{a} \pi(a|s) Q^{\pi}(s,a)$$

Can be seen as a sort of expectation over Q-values

$$Q^{\pi}(s,a) = \sum_{s'} p(s'|s,a)[r(s,a) + \gamma V(s')]$$

- By application of one step Bellman recursion

Puterman, M. L. (2014) *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Optimal value and action-value functions, optimal policy

Optimal values and optimal policy

- The optimal value and action-value functions are noted V^* and Q^*
- They are defined as:
- $$\forall s \in S, V^*(s) = \max_\pi V^\pi(s)$$
- $$\forall (s,a) \in S \times A, Q^*(s,a) = \max_\pi Q^\pi(s,a)$$
- The optimal policy π^* is the policy that achieves optimal value and optimal action-value functions:
- $$\pi = \pi^* \iff V^\pi = V^* \iff Q^\pi = Q^*$$
- Now the question is: how do we find V^* and Q^*?

Recursive operators and convergence

- If we define an operator T such that $X_{n+1} \leftarrow TX_n$
- It T is contractive, then through repeated application of T, X_n will converge to some fixed point
- For instance, if T divides by 2, X_n converges to 0

The Bellman optimality operator (Value Iteration)

- We call Bellman optimality operator (noted T^*) the application

$$\forall s \in S, V_{n+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_n(s') \Big]$$

- If $\gamma < 1$, T^* is contractive
- By iterating, computes the value of the current policy
- The optimal value function is the fixed-point of T^*: $V^* = T^* V^*$
- Value iteration: start from any V_0, then $V_{n+1} \leftarrow T^* V_n$

Value Iteration: the algorithm

Value Iteration, for estimating $\pi \approx \pi_*$

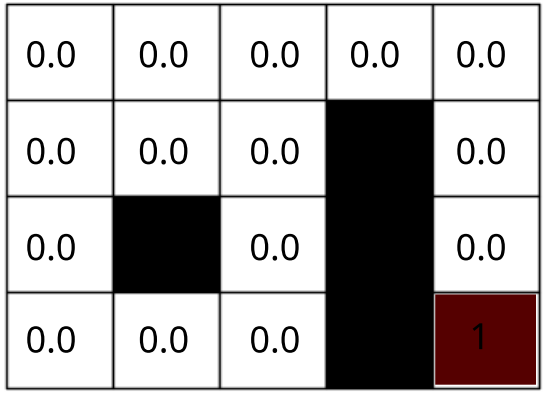
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$

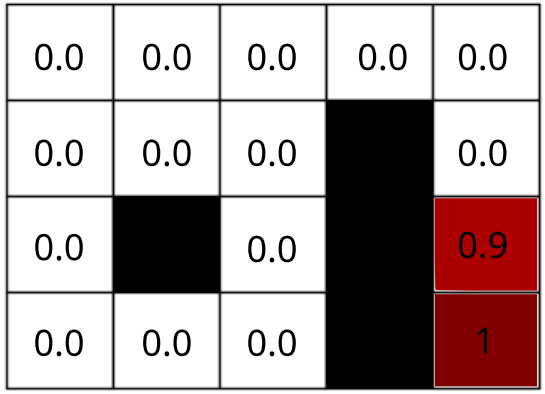
- Taken from Sutton & Barto, 2018, p. 83
- Reminder: $\sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] = r + \gamma \sum_{s'} p(s'|s, a)V(s')$
- (because r is deterministic)

Value Iteration in practice



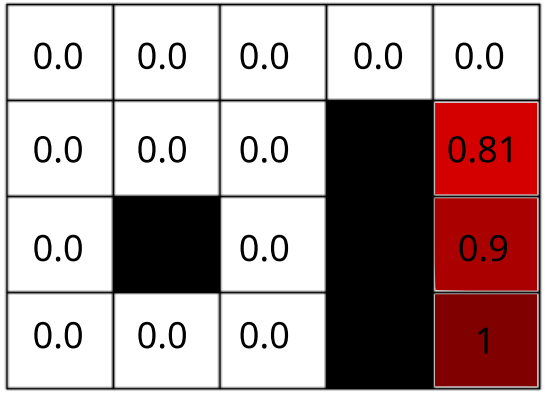
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



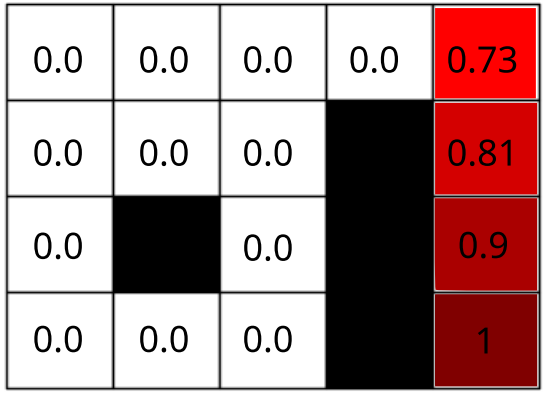
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



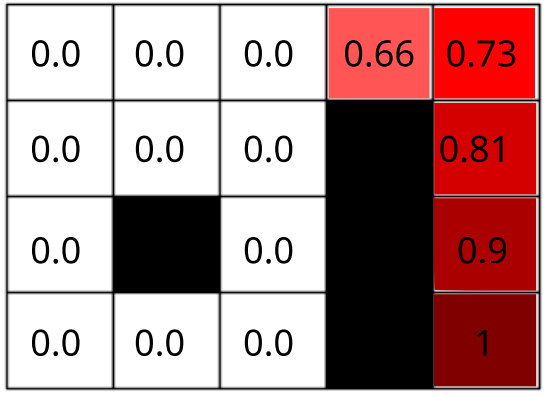
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



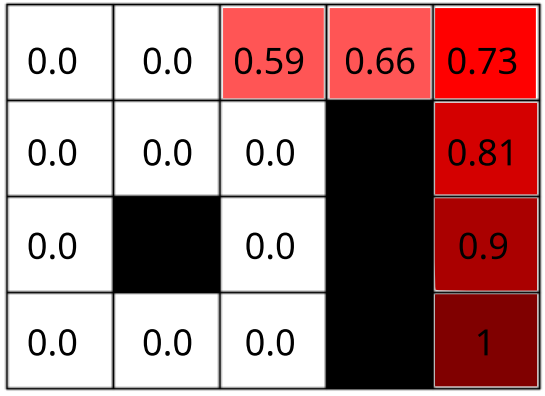
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



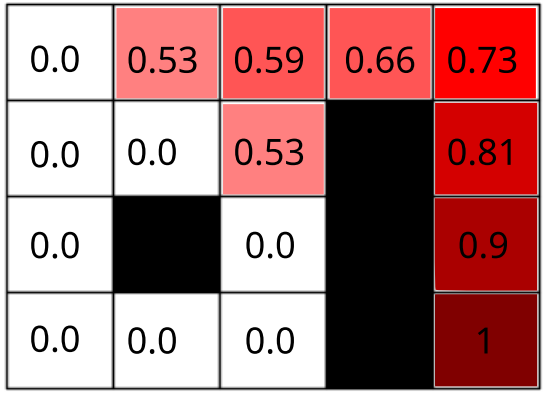
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



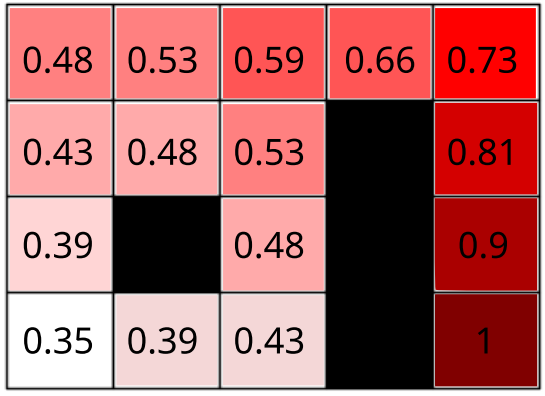
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



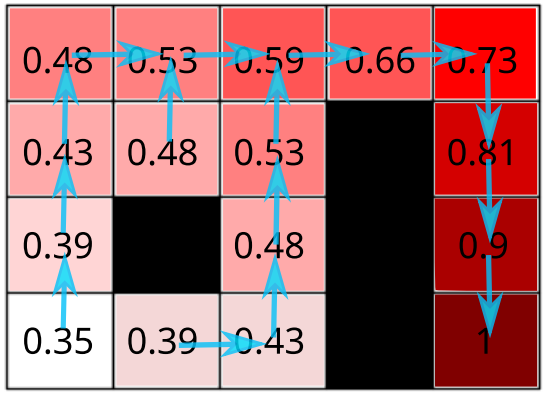
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



- We stop once changes are below a threshold
- Given a V function, determine a deterministic policy by $\pi(s) = \arg\max_{a \in A} \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \right]$
- Given a Q function, we do not need to represent the policy: just choose $\pi(s) = \arg\max_{a \in A} Q(s,a)$

The Bellman operator (Policy Iteration)

- We call Bellman operator (noted T^π) the application

$$\forall s \in S, V_{n+1}^\pi(s) \leftarrow r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V_n^\pi(s')$$

- If $\gamma < 1$, T is contractive
- Policy Iteration:
  - Start from any policy π_0
  - Policy evaluation:
    $V_{n+1}^\pi \leftarrow T^\pi V_n^\pi$
  - Policy improvement:
    $\forall s \in S, \pi_{n+1}(s) \leftarrow \arg\max_{a \in A} [r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_n^\pi(s')]$
- Converges to optimal value and policy
- Does not make much sense when using Q (the critic and the policy are the same)

Policy Iteration: the algorithm

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

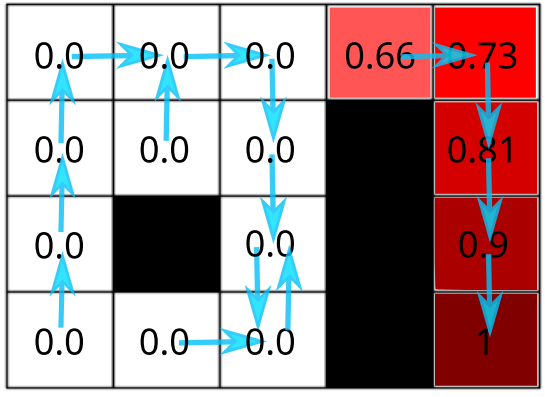
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

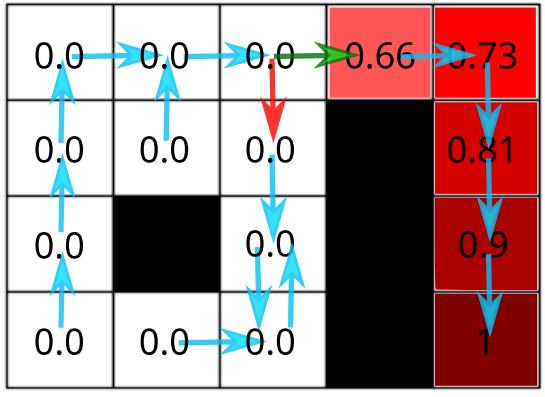
- Taken from Sutton & Barto, 2018, p. 80
- Note: $\sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] = r + \gamma \sum_{s'} p(s'|s, a)V(s')$
- (because r is deterministic)

Policy Iteration in practice



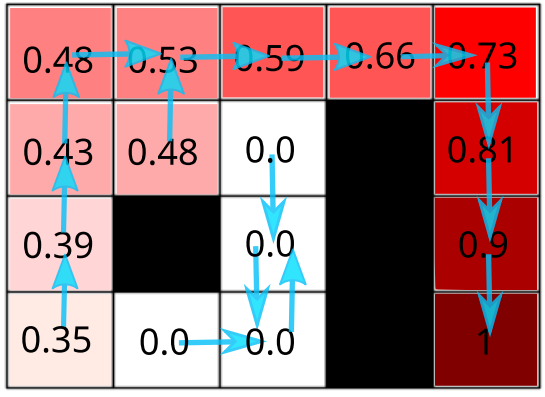
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



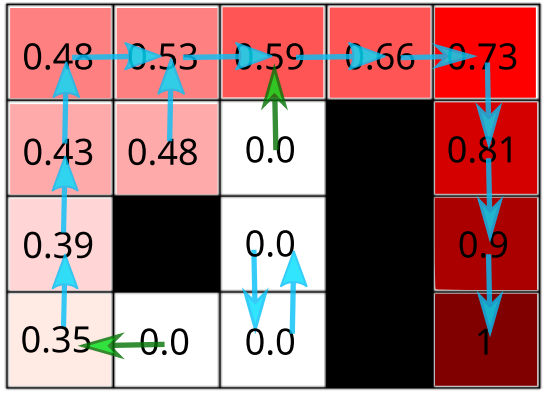
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



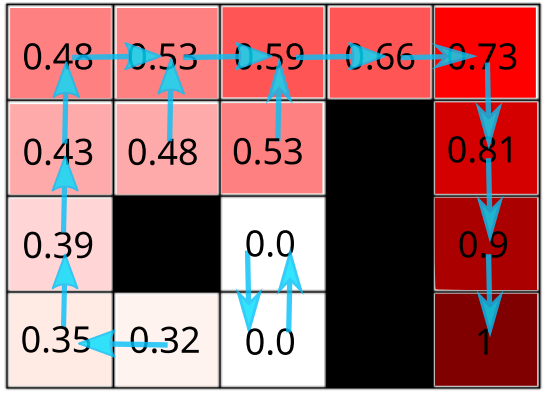
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



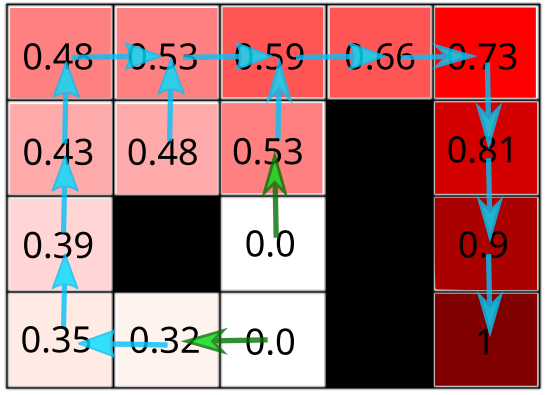
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



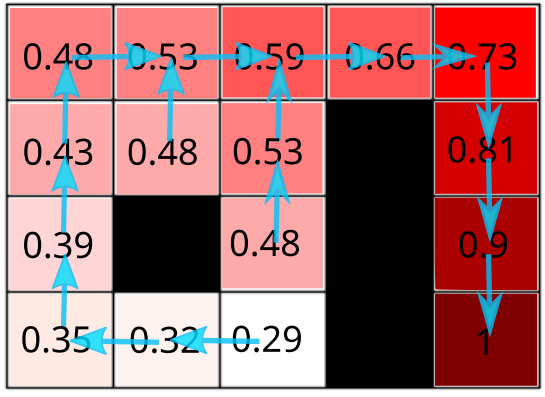
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



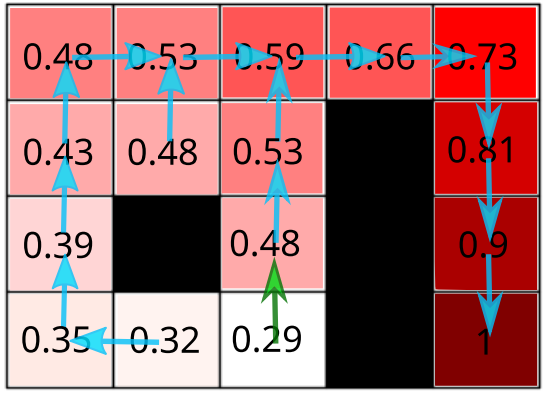
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



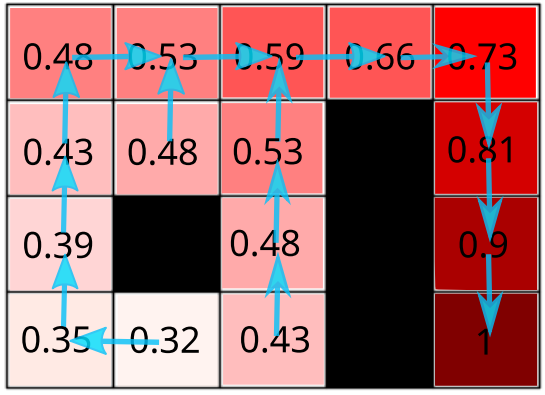
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



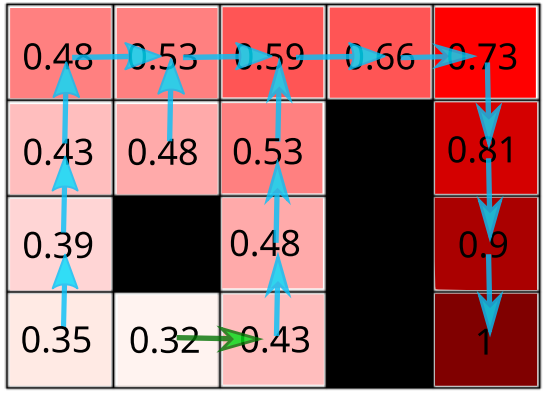
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



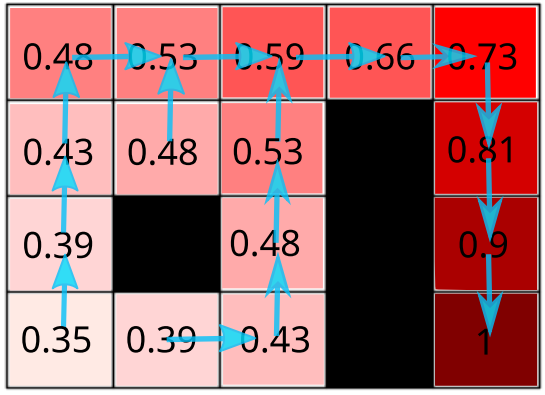
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



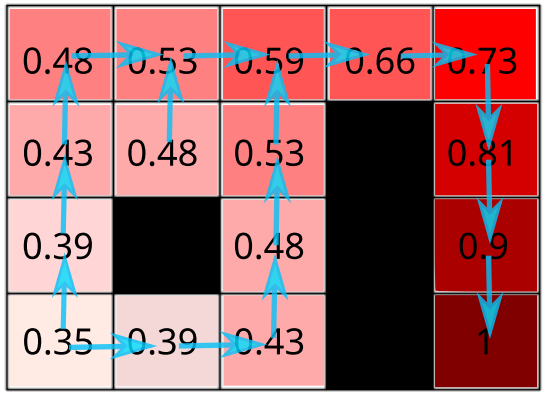
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



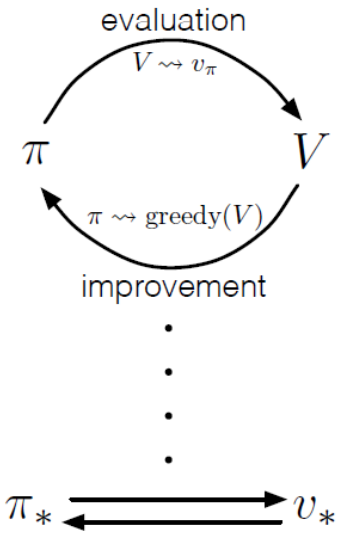
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

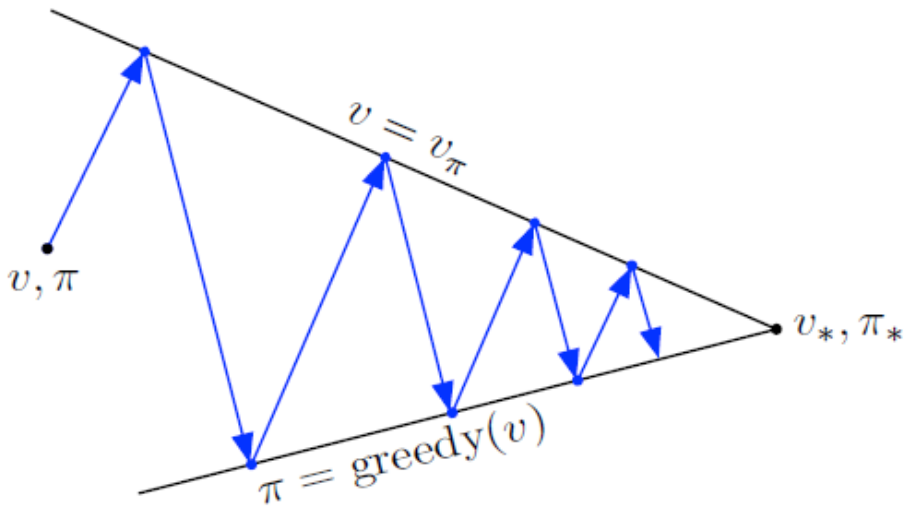
Policy Iteration in practice



Here we have managed a policy and a value representations at all steps

Generalized Policy Iteration





- Policy iteration evaluates each intermediate policy up to convergence. This is slow.
- Instead, evaluate the policy for N iterations, or even not for all states.
- Asynchronous dynamic programming: decoupling policy evaluation and improvement
- Taken from Sutton & Barto, 2018

Reinforcement learning

- In Dynamic Programming (planning), T and r are given
- Reinforcement learning goal: build π^* without knowing T and r
- Model-free approach: build π^* without estimating T nor r
- Actor-critic approach: special case of model-free
- Model-based approach: build a model of T and r and use it to improve the policy

Any question?

Send mail to: Olivier.Sigaud@isir.upmc.fr

Puterman, M. L. (2014).

Markov decision processes: discrete stochastic dynamic programming.
John Wiley & Sons.

Sutton, R. S. and Barto, A. G. (2018).

Reinforcement Learning: An Introduction (Second edition).
MIT Press.