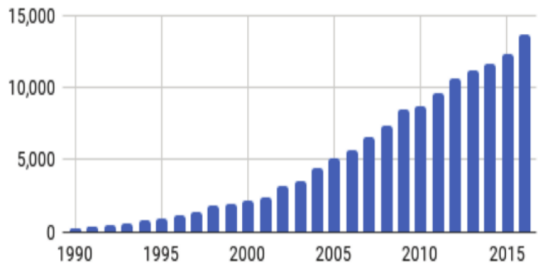
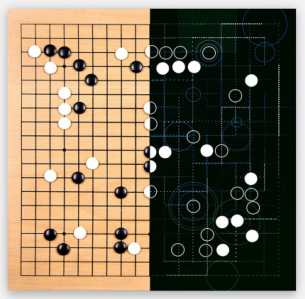
Tabular Reinforcement Learning

Olivier Sigaud

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

Why this class?



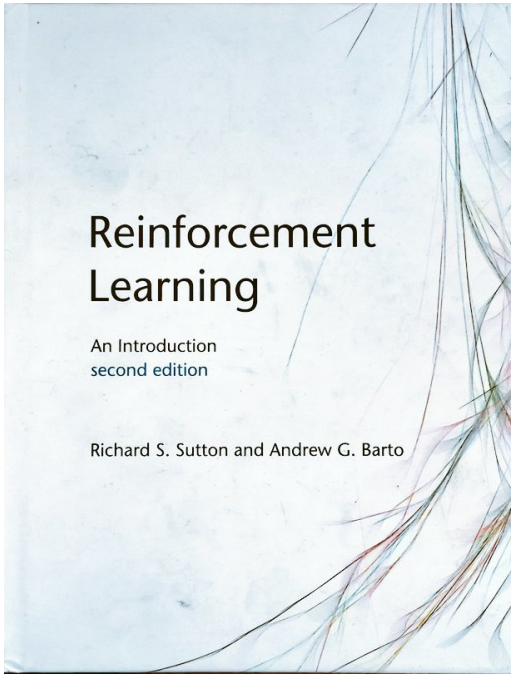


- A lot of buzz about deep reinforcement learning as an engineering tool
- The reinforcement learning framework is also relevant in computational neuroscience and psychology but this aspect will be left out

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015) Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017) Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359

Introductory book: the bible

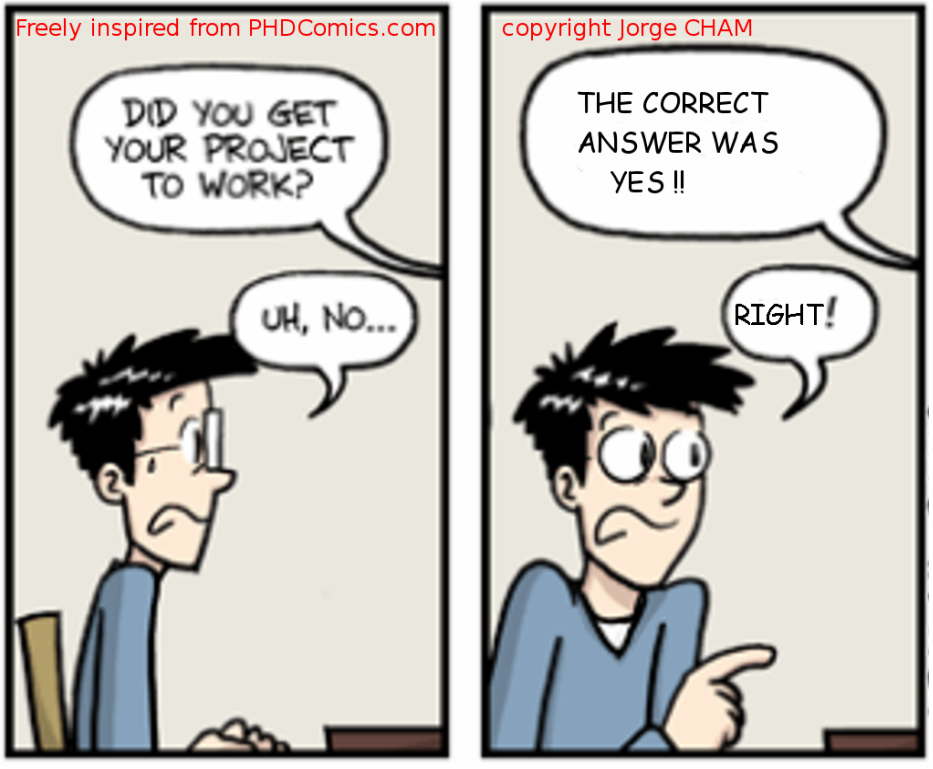


- [Sutton and Barto, 2018]: the ultimate introduction to the field, in the discrete case
- Available online:

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

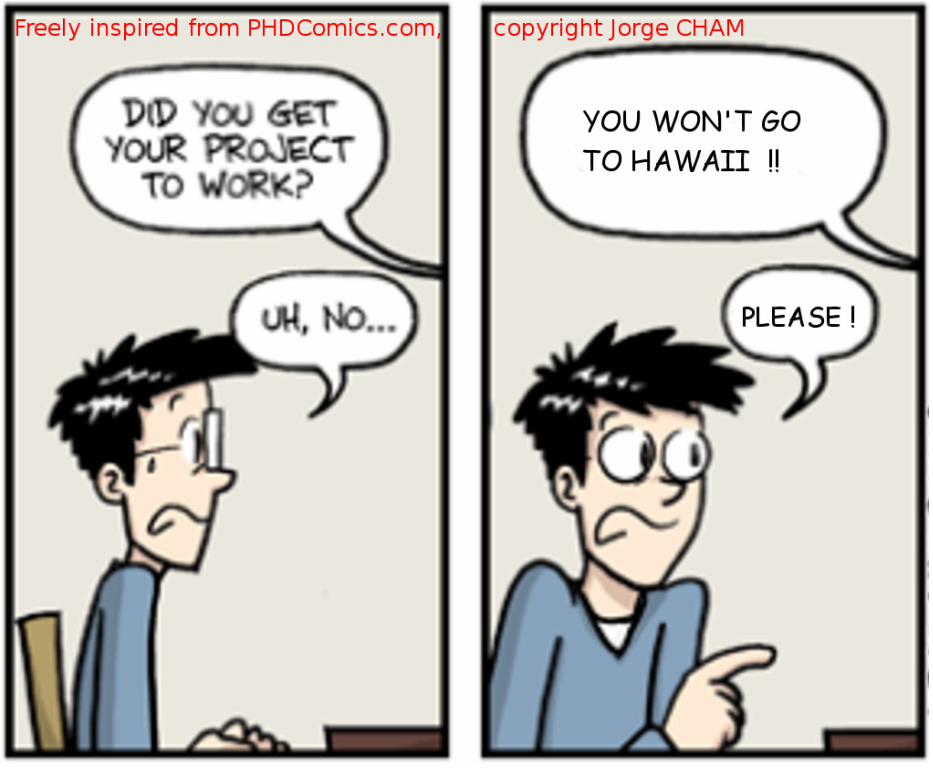
Sutton, R. S. & Barto, A. G. (2018) *Reinforcement Learning: An Introduction*. MIT Press.

Supervised learning



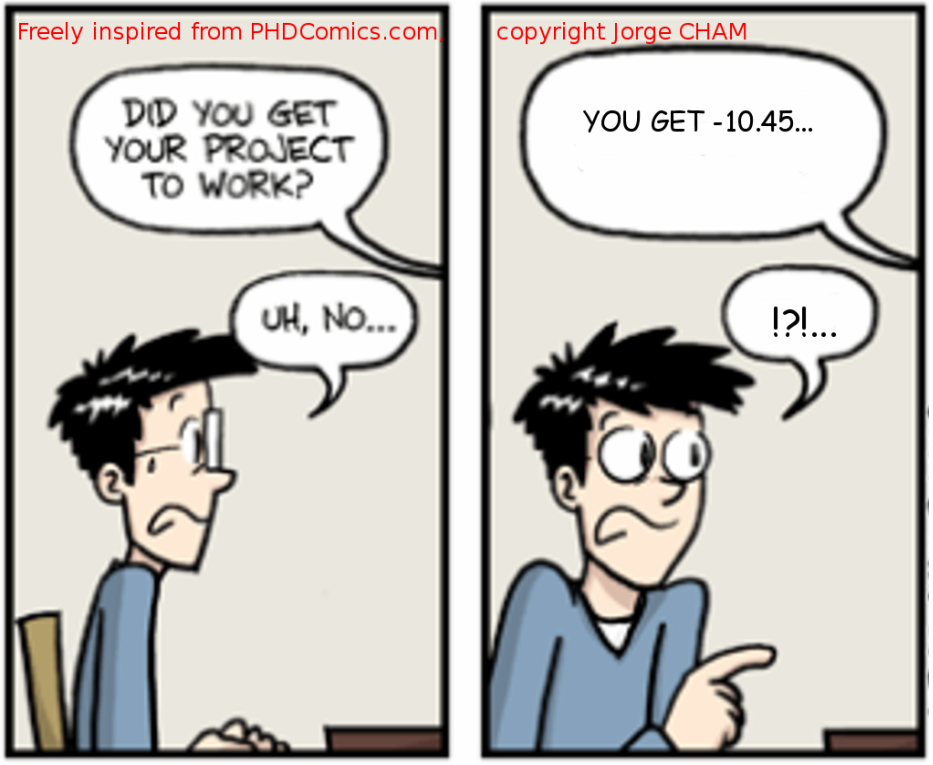
- The supervisor indicates to the agent the expected answer
- The agent corrects a model based on the answer
- Typical mechanism: gradient backpropagation, RLS
- Applications: classification, regression, function approximation...

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

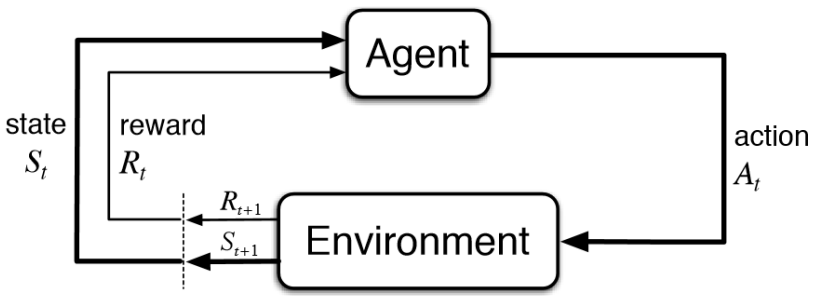
- Exploring can be (very) harmful
- Shall I exploit what I know or look for a better policy?
- Am I optimal? Shall I keep exploring or stop?
- Decrease the rate of exploration along time
- ϵ-*greedy*: take the best action most of the time, and a random action from time to time

Sequentiality: Bandits problems vs Sequential problems

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

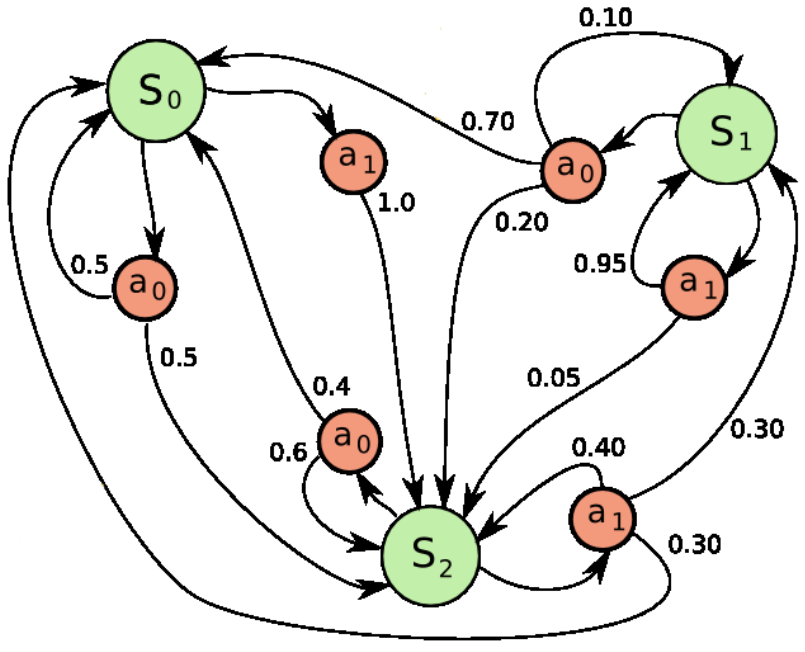
Markov Decision Processes



- 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

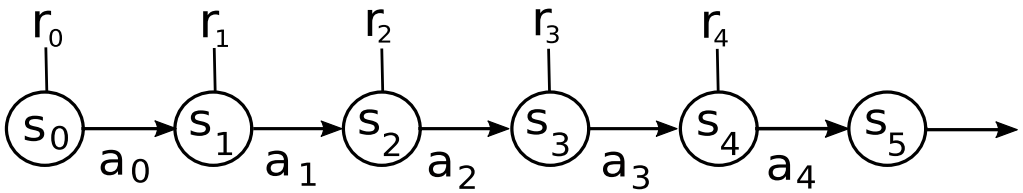
Stochastic transition function



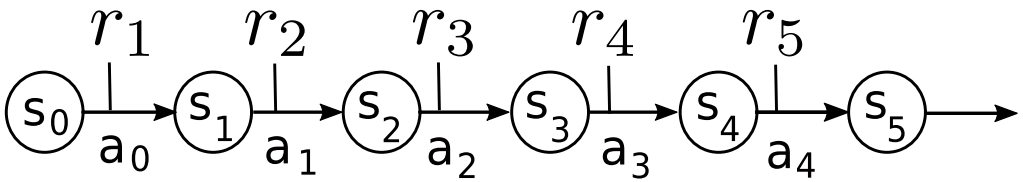
- Deterministic problem = special case of stochastic
- $T(s^t, a^t, s^{t+1}) = p(s'|s, a)$

Rewards: over states or action?

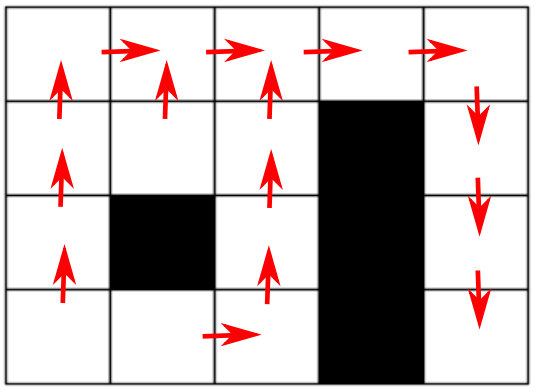
- Reward over states

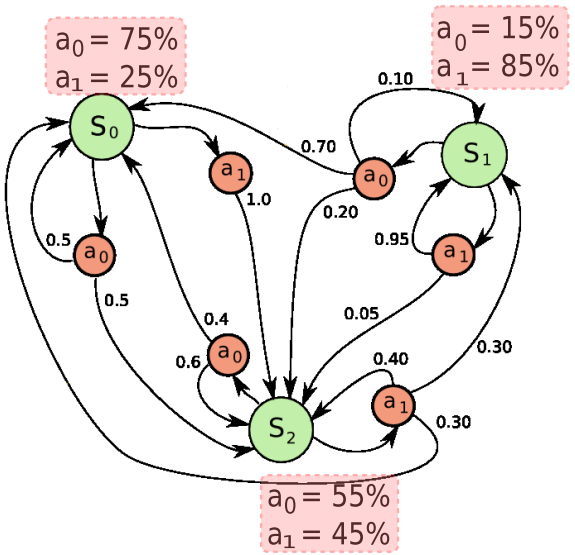


- Reward over actions in states



Deterministic versus stochastic policy

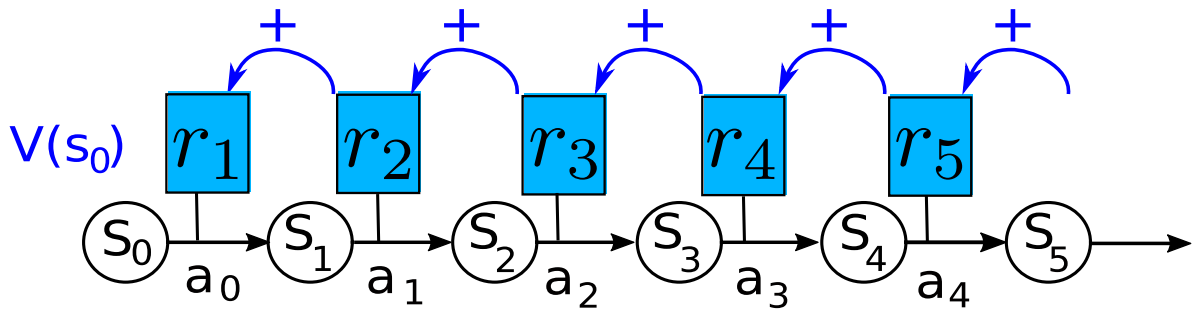




- Goal: find a policy $\pi : S \to A$ maximizing an agregation of rewards on the long run
- Important theorem: for any MDP, there exists a deterministic policy that is optimal

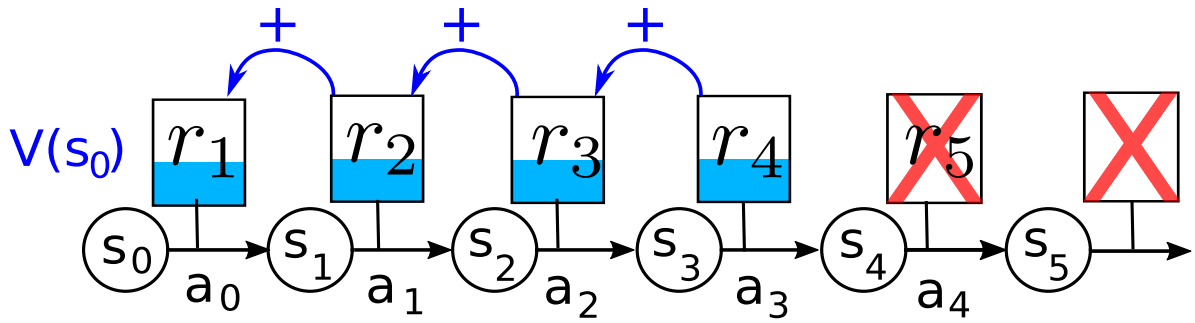
Agregation criterion: mere sum

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



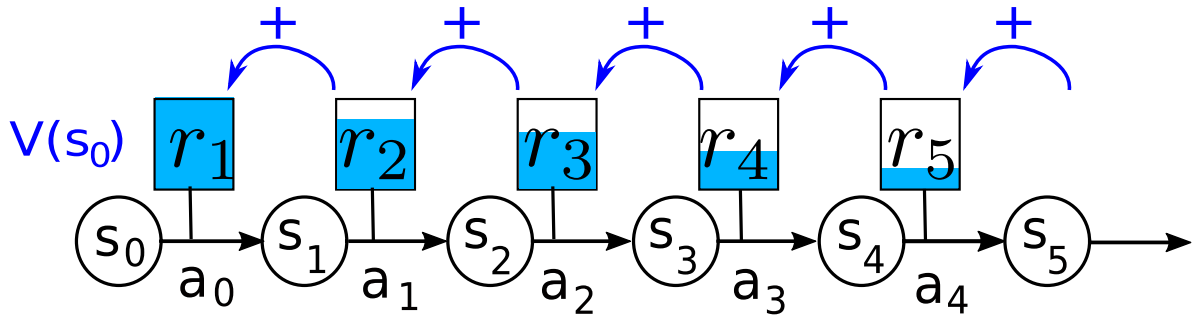
- The sum over a infinite horizon may be infinite, thus hard to compare
- Mere sum (finite horizon N): $V^{\pi}(S_0) = r_1 + r_2 + \ldots + r_N$

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

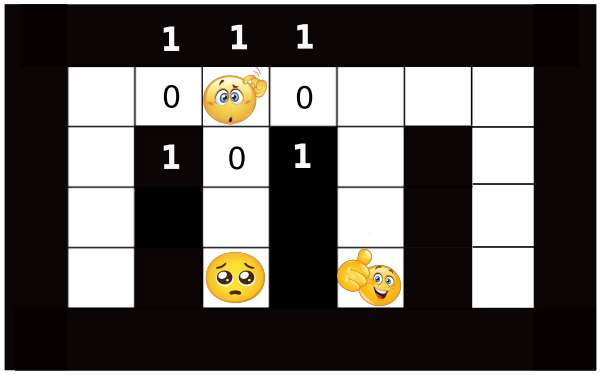


- Discounted criterion: $V^{\pi}(s_{t_0}) = \sum_{t=t_0}^{\infty} \gamma^t r(s_t, \pi(s_t))$
- $\gamma \in [0, 1]$: discount factor
  - if $\gamma = 0$, sensitive only to immediate reward
  - if $\gamma = 1$, future rewards are as important as immediate rewards
- The discounted case is the most used

Markov Property

- An 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

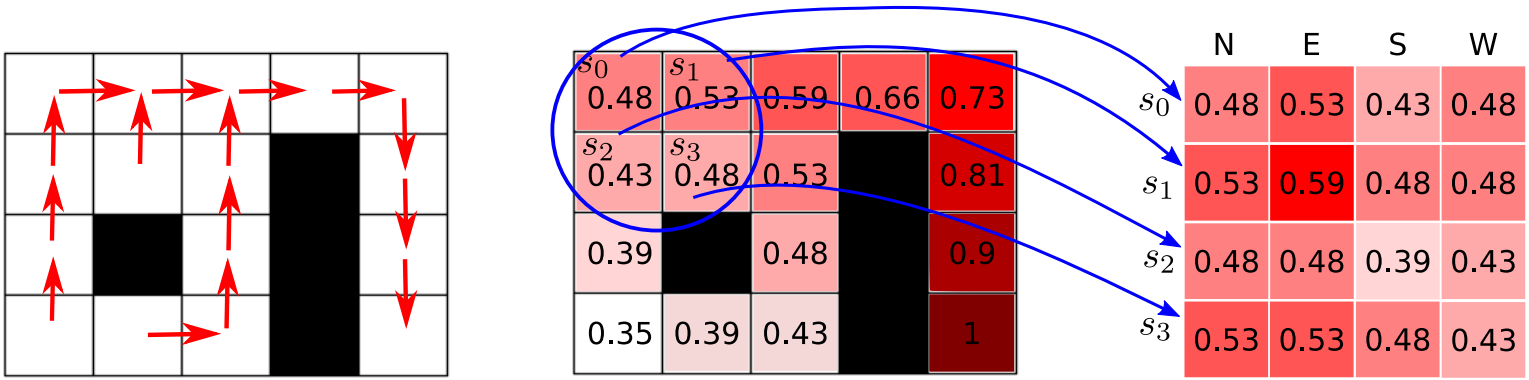


- Markov property is not verified if:
  - the observation does not contain all useful information to take decisions (POMDPs)
  - or if the next state depends on decisions of several agents (Dec-MDPs, Dec-POMDPs, Markov games)
  - or if transitions depend on time (Non-stationary problems)

Introduction

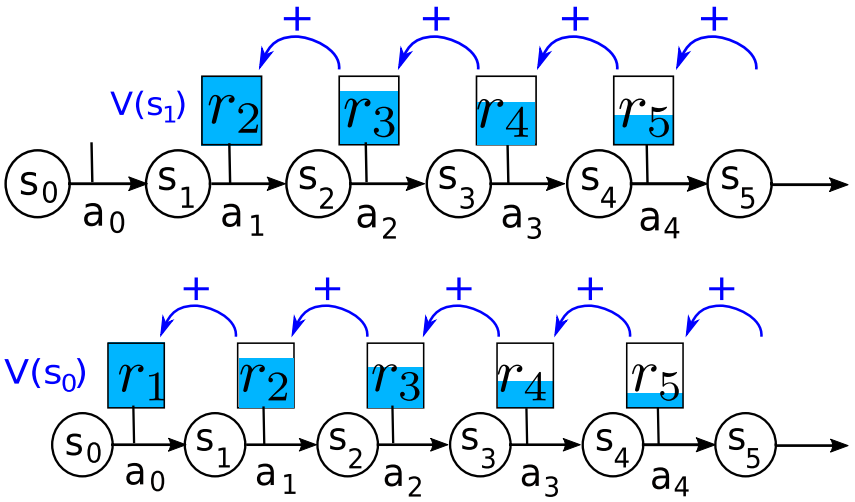
- Once we have defined an MDP
- Dynamic programming methods can find the optimal policy
- Assuming they know everything about the MDP
- Reinforcement Learning applies when the transition and reward functions are unknown
- To define dynamic programming methods, we need value functions

Value and action value functions



- The value function $V^\pi : S \to \mathbb{R}$ records the agregation of reward on the long run for each state (following policy π). It is a vector with one entry per state
- The action value function $Q^\pi : S \times A \to \mathbb{R}$ records the agregation of reward on the long run for doing each action in each state (and then following policy π). It is a matrix with one entry per state and per action
- In the remainder, we focus on V, trivial to transpose to Q

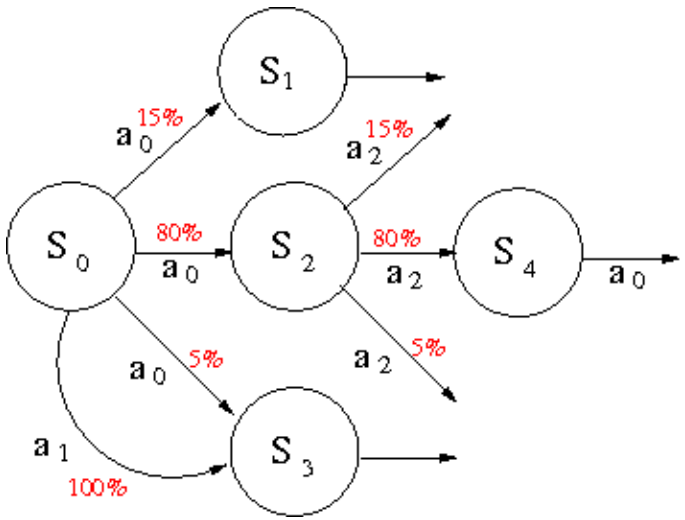
Bellman equation over a Markov chain: recursion



Given the discounted reward agregation criterion:

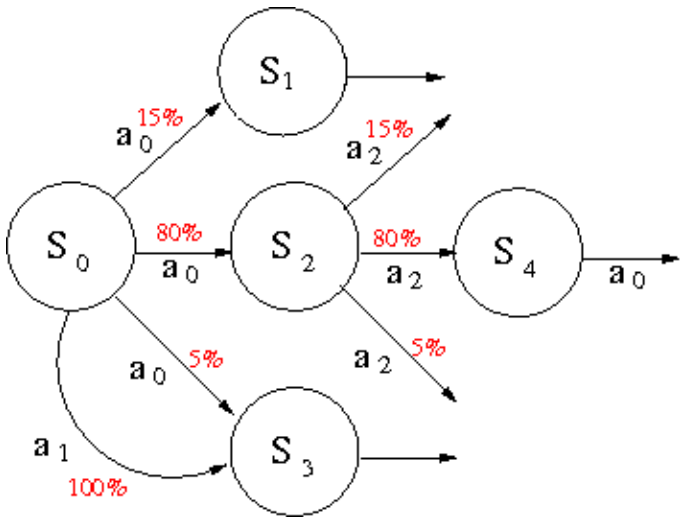
- $V(s_0) = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + ...$
- $V(s_0) = r_1 + \gamma(r_2 + \gamma r_3 + \gamma^2 r_4 + ...)$
- $V(s_0) = r_1 + \gamma V(s_1)$
- More generally $V(s_t) = r_{t+1} + \gamma V(s_{t+1})$

Bellman equation: general case



- 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
- The realizations are the next states
- Deterministic π: $V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V^\pi(s')$

Bellman equation: general case



- 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
- The realizations are the next states
- Stochastic π: $V^\pi(s) = \sum_a \pi(a|s)[r(s,a) + \gamma \sum_{s'} p(s'|s,a)V^\pi(s')]$

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

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

- 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: $V_{n+1} \leftarrow T^* V_n$

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

The Bellman operator (Policy Iteration)

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

$$V^\pi_{n+1}(s) \leftarrow r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V^\pi_n(s')$$

- If $\gamma < 1$, T is contractive
- Converges to optimal value and policy
- Policy Iteration:
  - Policy evaluation:
    $V^\pi_{n+1} \leftarrow T^\pi V^\pi_n$
  - Policy improvement:
    $\forall s \in S, \pi'(s) \leftarrow \arg\max_{a \in A} \sum_{s'} p(s'|s,a)[r(s,a) + \gamma V^\pi_n(s')]$
    or
    $\forall s \in S, \pi'(s) \leftarrow \arg\max_{a \in A} [r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^\pi_n(s')]$
- Note: $\sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] = r + \gamma \sum_{s'} p(s'|s, a) V(s')$

Value Iteration: the algorithm

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

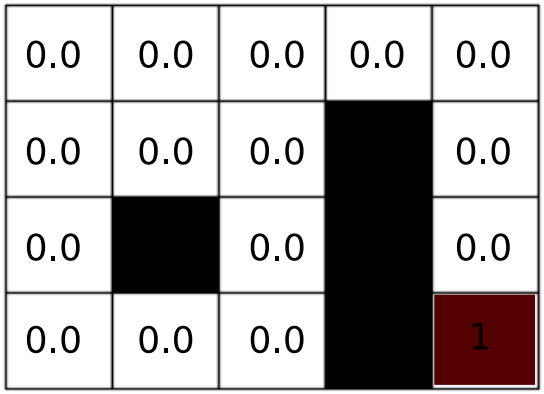
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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

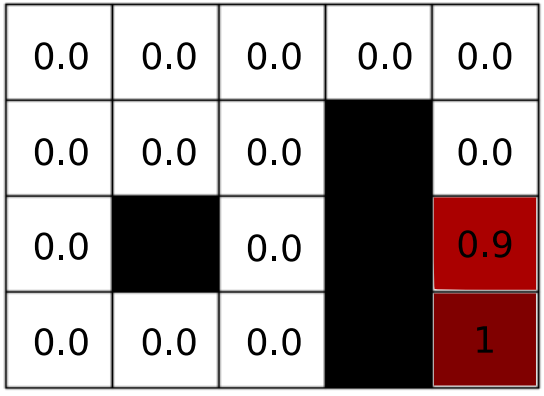
- Taken from Sutton & Barto, 2018, p. 83
- Reminder: $\sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] = r + \gamma \sum_{s'} p(s'|s, a)V(s')$

Value Iteration in practice



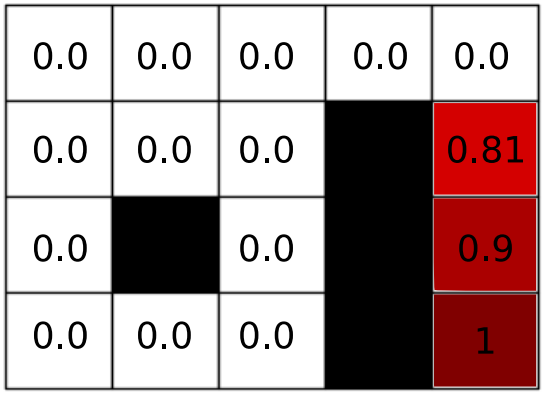
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



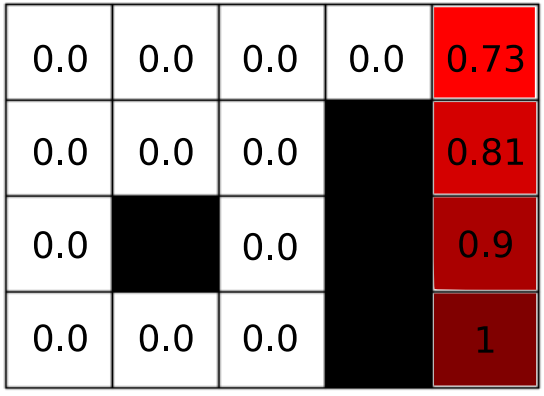
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



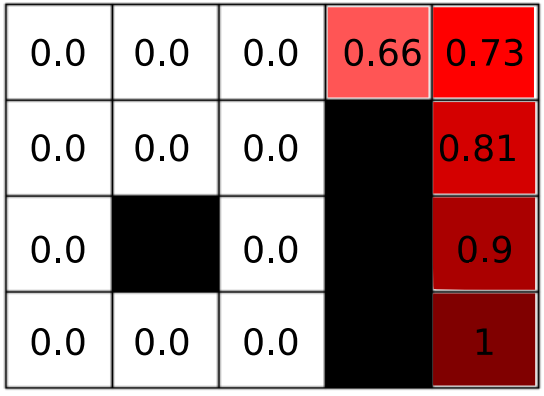
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



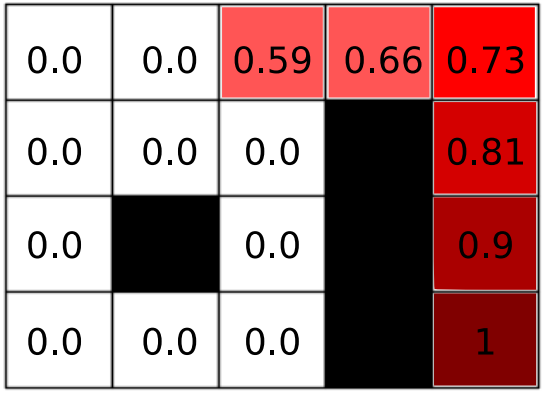
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



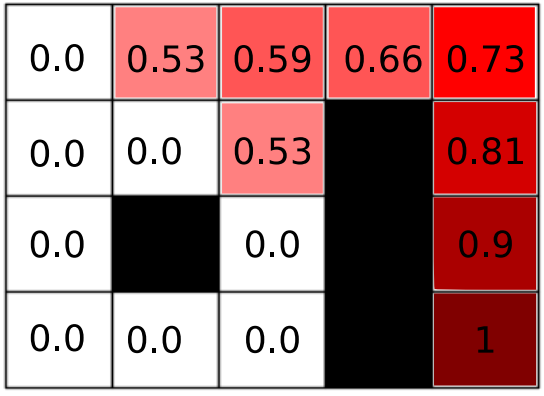
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



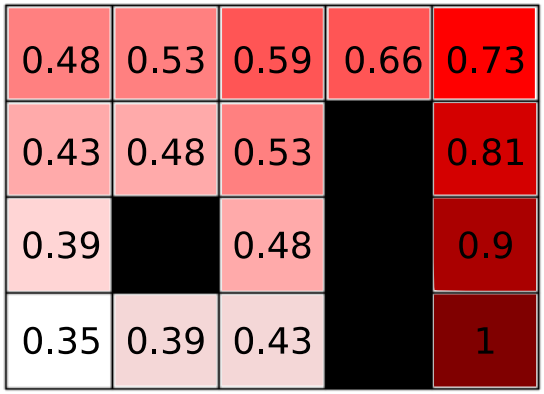
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



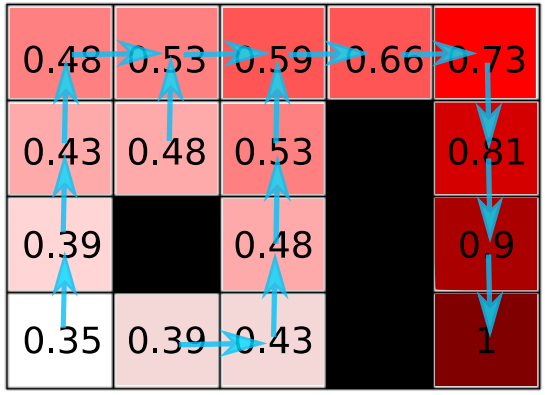
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

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 have iterated on values, and determined a policy out of it (without necessarily representing it if using $Q(s, a)$)

Policy Iteration: the algorithm

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

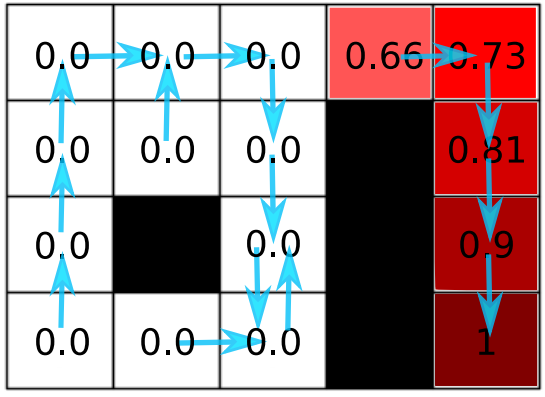
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

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

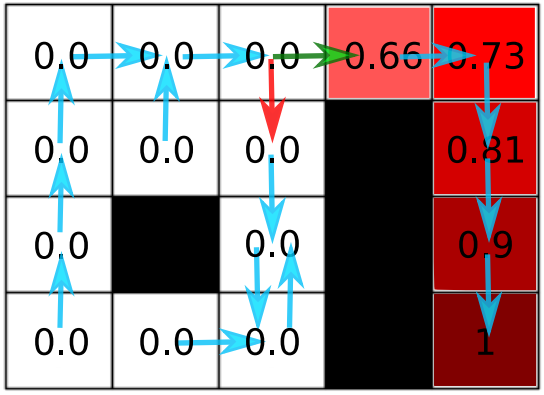
- Taken from Sutton & Barto, 2018, p. 80
- Note: $\sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] = r + \gamma \sum_{s'} p(s'|s, a)V(s')$

Policy Iteration in practice



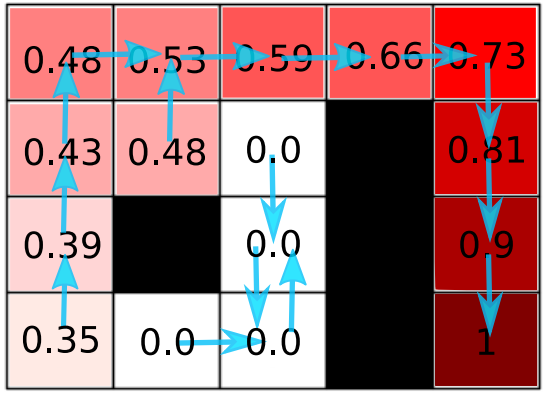
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



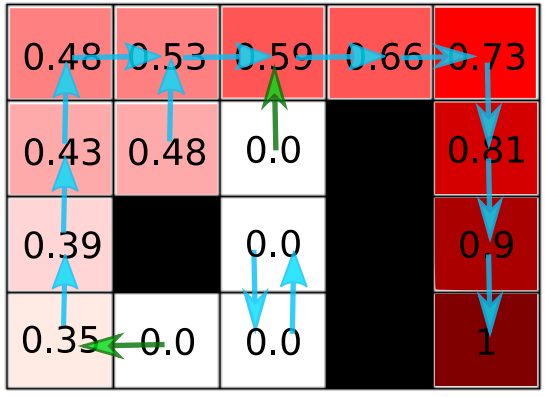
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



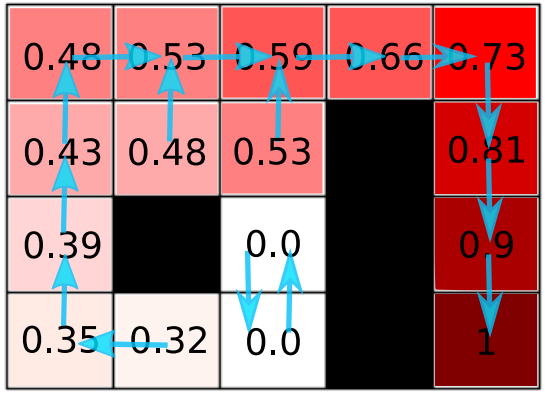
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



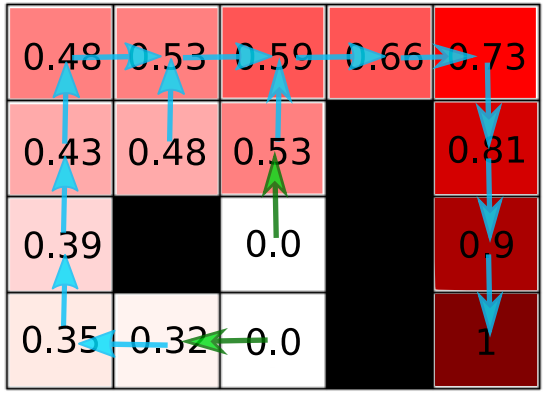
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



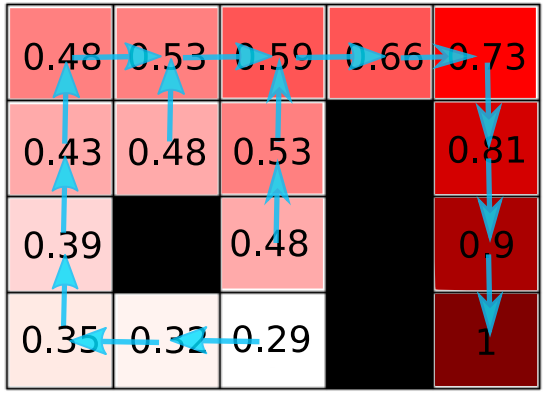
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



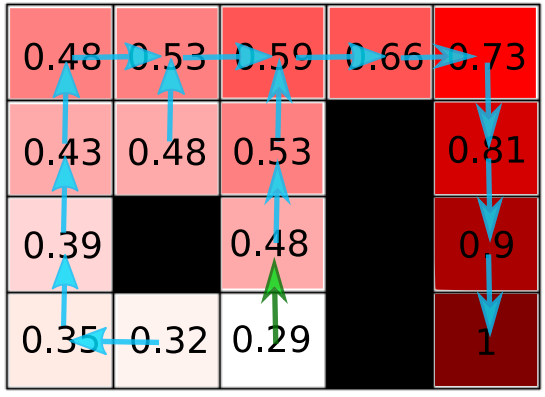
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



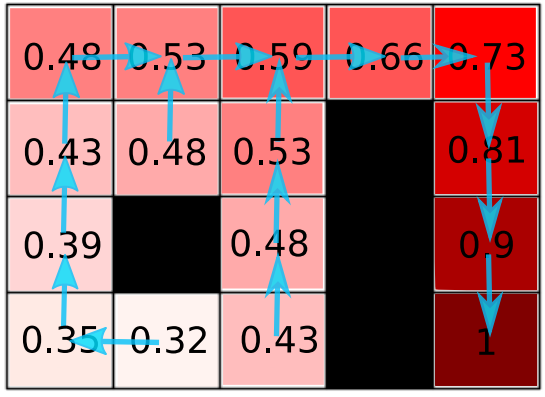
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



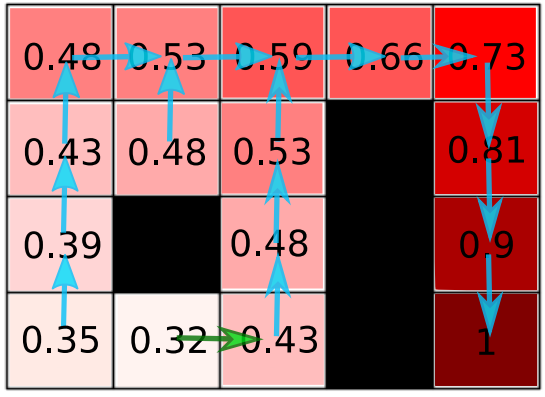
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



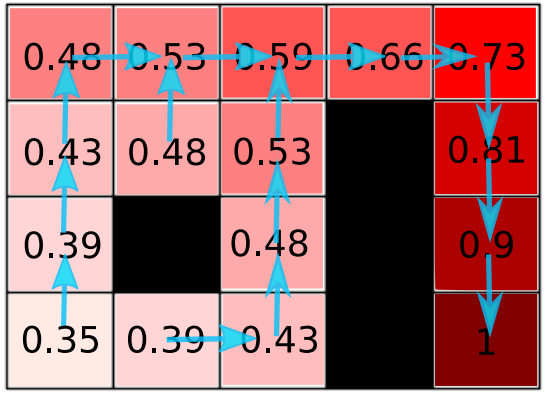
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



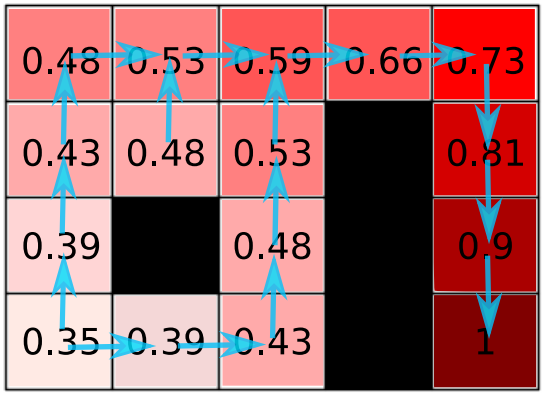
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



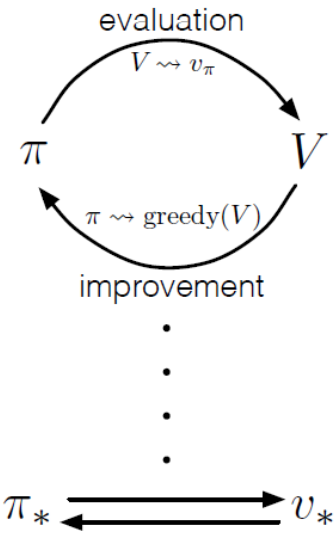
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

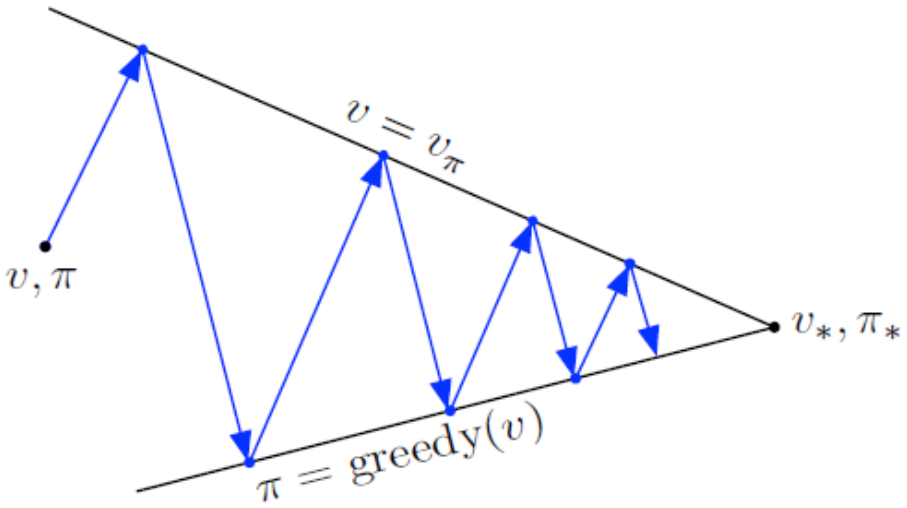
Policy Iteration in practice



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 dynamics 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

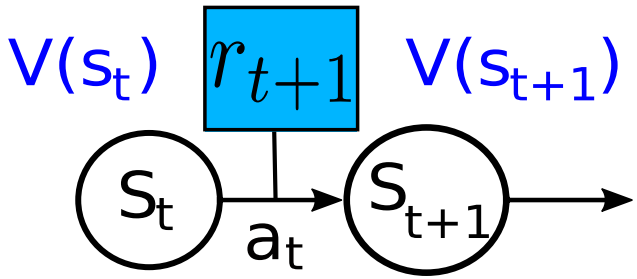
Incremental estimation

- Estimating the average immediate (stochastic) reward in a state s
- $E_k(s) = (r_1 + r_2 + ... + r_k)/k$
- $E_{k+1}(s) = (r_1 + r_2 + ... + r_k + r_{k+1})/(k+1)$
- Thus $E_{k+1}(s) = k/(k+1)E_k(s) + r_{k+1}/(k+1)$
- Or $E_{k+1}(s) = (k+1)/(k+1)E_k(s) - E_k(s)/(k+1) + r_{k+1}/(k+1)$
- Or $E_{k+1}(s) = E_k(s) + 1/(k+1)[r_{k+1} - E_k(s)]$
- Still needs to store k
- Can be approximated as

$$E_{k+1}(s) = E_k(s) + \alpha[r_{k+1} - E_k(s)] \quad (1)$$

- Converges to the true average (slower or faster depending on α) without storing anything
- Equation (1) is everywhere in reinforcement learning

Temporal Difference error



- The goal of TD methods is to estimate the value function $V(s)$
- If estimations $V(s_t)$ and $V(s_{t+1})$ were exact, we would get $V(s_t) = r_{t+1} + \gamma V(s_{t+1})$
- The approximation error is

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2}$$

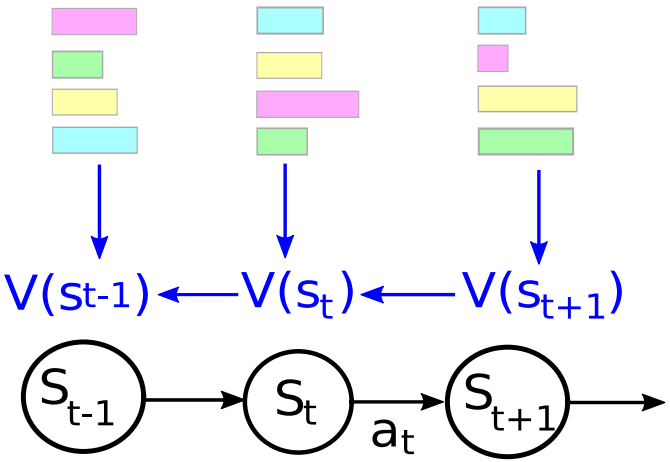
- δ_t measures the error between $V(s_t)$ and the value it should have given $r_{t+1} + \gamma V(s_{t+1})$
- If $\delta_t > 0$, $V(s_t)$ is under-evaluated, otherwise it is over-evaluated
- $V(s_t) \leftarrow V(s_t) + \alpha\delta_t$ should decrease the error (value propagation)

Temporal Difference update rule

$$E_{k+1}(s) = E_k(s) + \alpha[r_{k+1} - E_k(s)] \quad (1)$$

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \quad (2)$$

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad (3)$$



$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad (3)$$

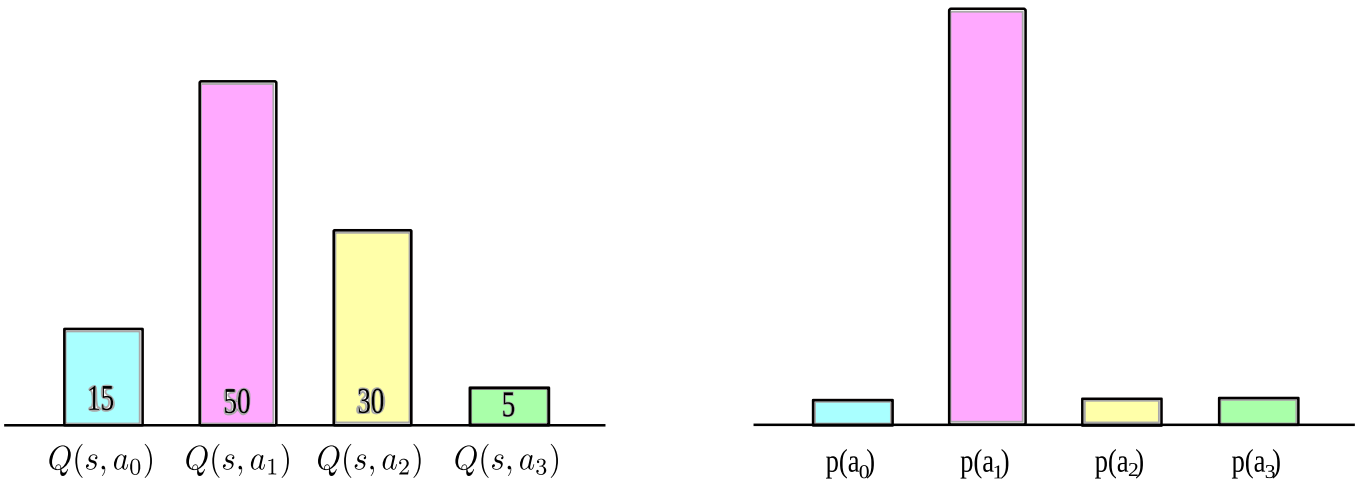
- Combines two estimation processes:
  - incremental estimation (1)
  - value propagation from $V(s_{t+1})$ to $V(s_t)$ (2)

The Policy evaluation algorithm: TD(0)

- An agent performs a sequence
  $s_0, a_0, r_1, \cdots, s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, \cdots$
- Performs local Temporal Difference updates from s_t, s_{t+1} and r_{t+1}
- Proved in 1994 provided ϵ-greedy exploration
- Note: updates can be performed in any order

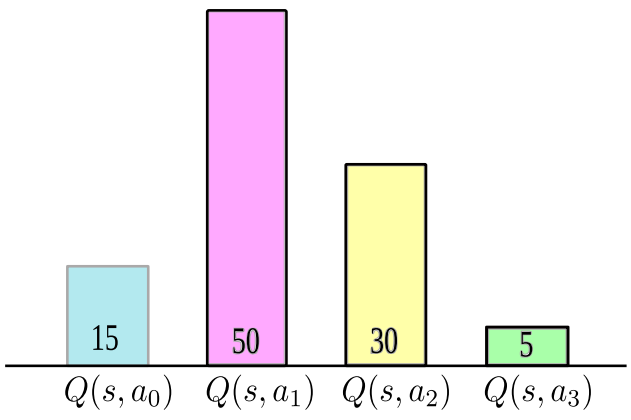
Dayan, P. & Sejnowski, T. (1994). TD(lambda) converges with probability 1. *Machine Learning*, 14(3):295–301.

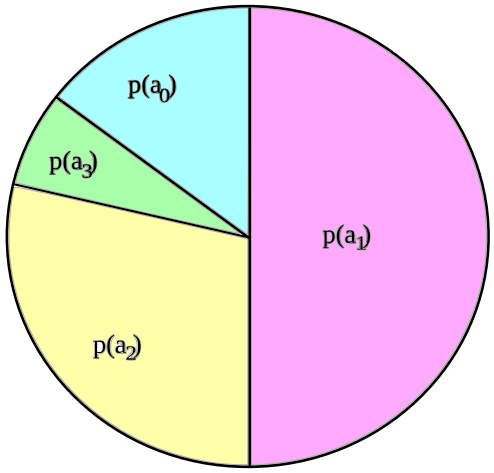
ϵ-greedy exploration



- Choose the best action with a high probability, other actions at random with low probability
- Same properties as random search
- Every state-action pair will be enough visited under an infinite horizon
- Useful for convergence proofs

Roulette wheel

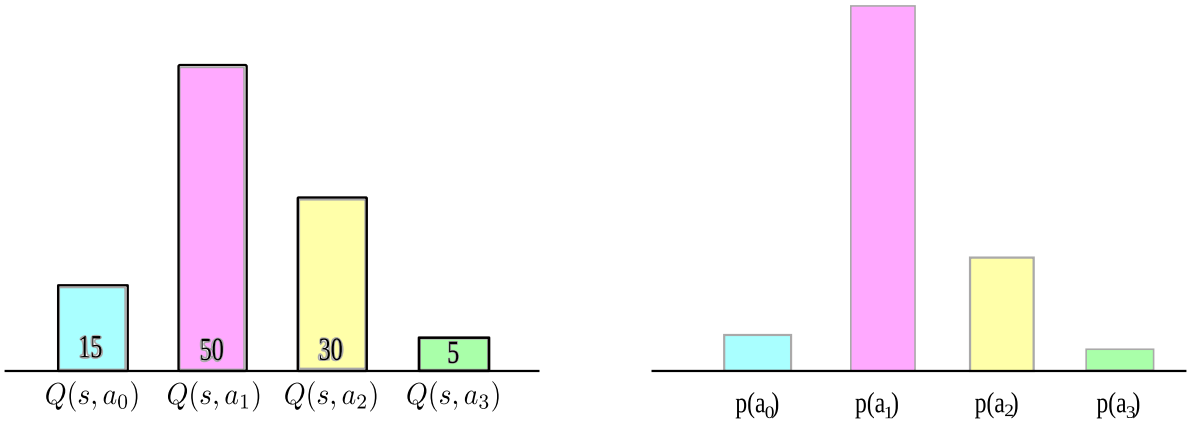




$$p(a_i) = \frac{Q(s,a_i)}{\sum_j Q(s,a_j)}$$

- The probability of choosing each action is proportional to its value

Softmax exploration



$$p(a_i) = \frac{e^{\frac{Q(s,a_i)}{\beta}}}{\sum_j e^{\frac{Q(s,a_j)}{\beta}}}$$

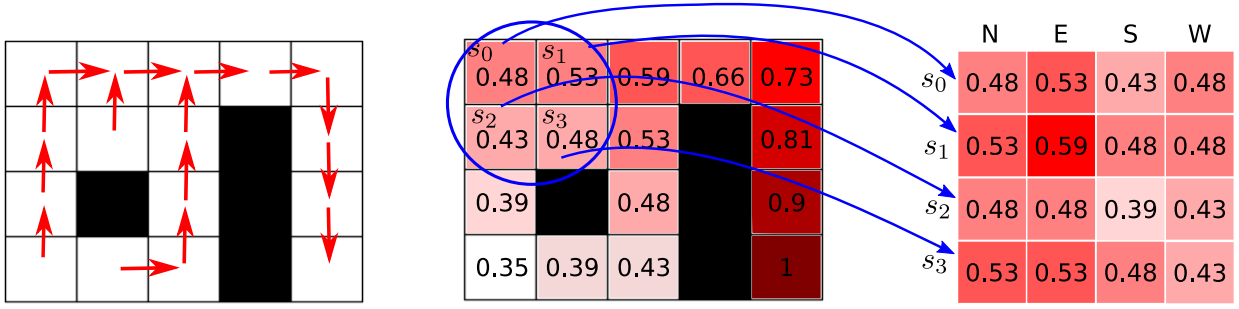
- The parameter β is called the temperature
- If $\beta \to 0$, increase contrast between values
- If $\beta \to \infty$, all actions have the same probability $\to$ random choice
- Meta-learning: tune β dynamically (exploration/exploitation)
- More used in computational neurosciences

George Velentzas, Costas Tzafestas, and Mehdi Khamassi. (2017) Bio-inspired meta-learning for active exploration during non-stationary multi-armed bandit tasks. In *2017 Intelligent Systems Conference (IntelliSys)*, pp. 661–669. IEEE

TD(0): limitation

- TD(0) evaluates $V(s)$
- One cannot infer $\pi(s)$ from $V(s)$ without knowing T: one must know which a leads to the best $V(s')$
- Three solutions:
  - Q-LEARNING, SARSA: Work with $Q(s,a)$ rather than $V(s)$.
  - ACTOR-CRITIC methods: Simultaneously learn V and update π
  - DYNA: Learn a model of T: model-based (or indirect) reinforcement learning

Value function and Action Value function



- The value function $V^\pi : S \to \mathbb{R}$ records the agregation of reward on the long run for each state (following policy π). It is a vector with one entry per state
- The action value function $Q^\pi : S \times A \to \mathbb{R}$ records the agregation of reward on the long run for doing each action in each state (and then following policy π). It is a matrix with one entry per state and per action

SARSA

- Reminder (TD): $V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$
- SARSA: For each observed $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$:
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
- Policy: perform exploration (e.g. ϵ-*greedy*)
- One must know the action a_{t+1}, thus constrains exploration
- On-policy method: more complex convergence proof

Singh, S. P., Jaakkola, T., Littman, M. L., & Szepesvari, C. (2000). Convergence Results for Single-Step On-Policy Reinforcement Learning Algorithms. *Machine Learning*, 38(3):287–308.

SARSA: the algorithm

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

- Taken from Sutton & Barto, 2018

Q-LEARNING

- For each observed $(s_t, a_t, r_{t+1}, s_{t+1})$:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- $\max_{a \in A} Q(s_{t+1}, a)$ instead of $Q(s_{t+1}, a_{t+1})$
- Off-policy method: no more need to know a_{t+1}
- Policy: perform exploration (e.g. ϵ-*greedy*)
- Convergence proven given infinite exploration


Watkins, C. J. C. H. (1989). *Learning with Delayed Rewards*. PhD thesis, Psychology Department, University of Cambridge, England.


Watkins, C. J. C. H. & Dayan, P. (1992) Q-learning. *Machine Learning*, 8:279–292

Q-LEARNING: the algorithm

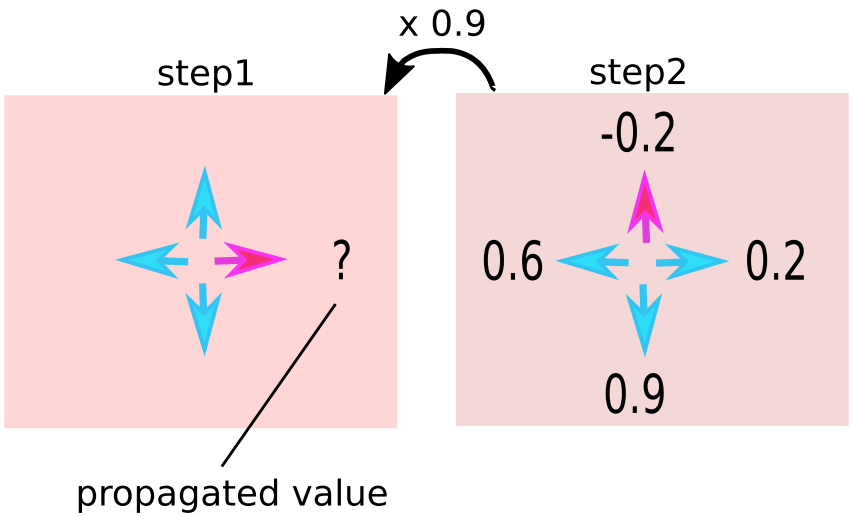
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

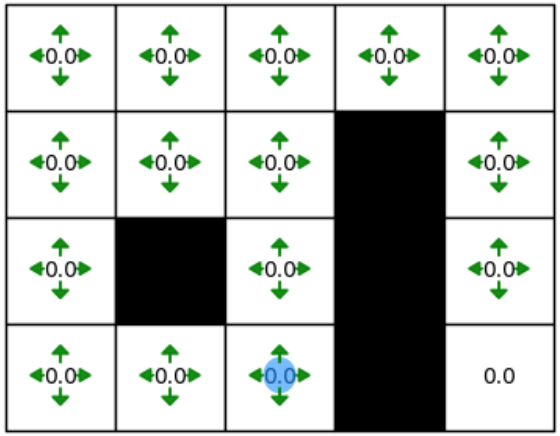
- Taken from Sutton & Barto, 2018

Difference between Q-LEARNING and SARSA



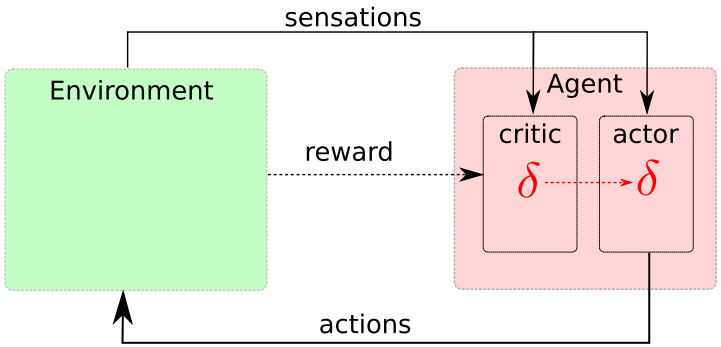
- Consider an agent taking the two pink actions
- With Q-LEARNING, the propagated value ? is $\gamma \operatorname{argmax}_a Q(s_{t+1}, a)$, thus $0.9 \times 0.9 = 0.81$
- With SARSA, it is $\gamma Q(s_{t+1}, a_{t+1})$, thus $0.9 \times -0.2 = -0.18$

Q-LEARNING in practice



- Build a states×actions table (*Q-Table*, eventually incremental)
- Initialise it (randomly or with 0 is not a good choice)
- Apply update equation after each action
- Problem: it is (very) slow

Actor-critic: Naive design

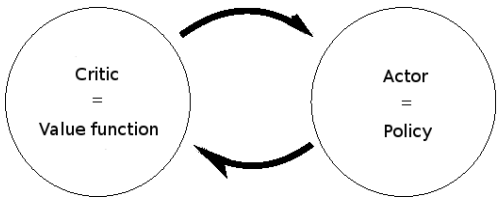


- Discrete states and actions, stochastic policy
- An update in the critic generates a local update in the actor
- Critic: compute δ and update $V(s)$ with $V_{k+1}(s) \leftarrow V_k(s) + \alpha_k \delta_k$
- Actor: $P^{\pi}_{k+1}(a|s) \leftarrow P^{\pi}_{k}(a|s) + \alpha_k \prime \delta_k$
- NB: no need for a max over actions
- NB2: one must know how to "draw" an action from a probabilistic policy (not straightforward for continuous actions)

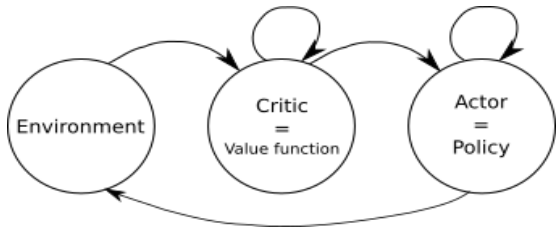
Williams, R. J. and Baird, L. (1990) A mathematical analysis of actor-critic architectures for learning optimal controls through incremental dynamic programming. In *Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems*, pages 96–101

Dynamic Programming and Actor-Critic

Policy Iteration



Actor-Critic



- In both PI and AC, the architecture contains a representation of the value function (the critic) and the policy (the actor)
- In PI, the MDP (T and r) is known
- PI alternates two stages:
  1. Policy evaluation: update ($V(s)$) or ($Q(s,a)$) given the current policy
  2. Policy improvement: follow the value gradient
- In AC, T and r are unknown and not represented (model-free)
- Information from the environment generates updates in the critic, then in the actor

From $Q(s, a)$ to Actor-Critic

state / action	a_0	a_1	a_2	a_3
e_0	0.66	0.88*	0.81	0.73
e_1	0.73	0.63	0.9*	0.43
e_2	0.73	0.9	0.95*	0.73
e_3	0.81	0.9	1.0*	0.81
e_4	0.81	1.0*	0.81	0.9
e_5	0.9	1.0*	0.0	0.9

state	chosen action
e_0	a_1
e_1	a_2
e_2	a_2
e_3	a_2
e_4	a_1
e_5	a_1

- Given a $Q - Table$, one must determine the max at each step
- This becomes expensive if there are numerous actions
- Store the best value for each state
- Update the max by just comparing the changed value and the max
- No more maximum over actions (only in one case)
- Storing the max is equivalent to storing the policy
- Update the policy as a function of value updates

Any question?

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


Dayan, P. and Sejnowski, T. (1994).
TD(lambda) converges with probability 1.
Machine Learning, 14(3):295–301.


Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015).
Human-level control through deep reinforcement learning.
Nature, 518(7540):529–533.


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


Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017).
Mastering chess and shogi by self-play with a general reinforcement learning algorithm.
arXiv preprint arXiv:1712.01815.


Singh, S. P., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000).
Convergence results for single-step on-policy reinforcement-learning algorithms.
Machine learning, 38(3):287–308.


Sutton, R. S. and Barto, A. G. (2018).
Reinforcement Learning: An Introduction (Second edition).
MIT Press.


Velentzas, G., Tzafestas, C., and Khamassi, M. (2017).
Bio-inspired meta-learning for active exploration during non-stationary multi-armed bandit tasks.
In *2017 Intelligent Systems Conference (IntelliSys)*, pages 661–669. IEEE.


Watkins, C. J. C. H. (1989).
Learning with Delayed Rewards.
PhD thesis, Psychology Department, University of Cambridge, England.

Watkins, C. J. C. H. and Dayan, P. (1992).

Q-learning.
Machine Learning, 8:279–292.

Williams, R. J. and Baird, L. (1990).

A mathematical analysis of actor-critic architectures for learning optimal controls through incremental dynamic programming.
In *Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems*, pages 96–101. Citeseer.