Reinforcement Learning

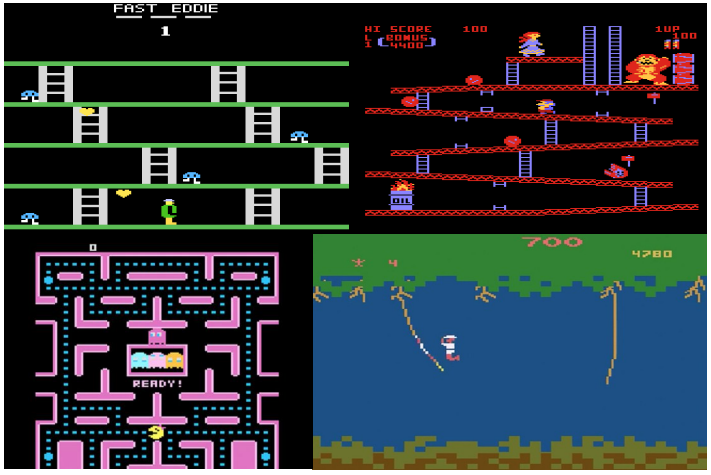
From the basics to Deep RL

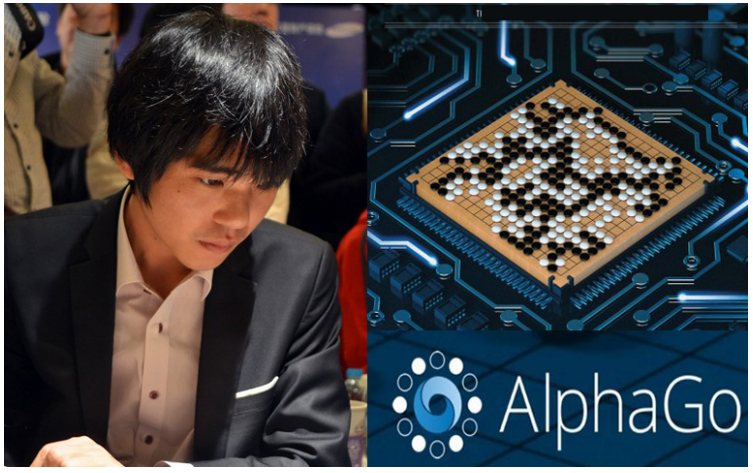
Olivier Sigaud

Université Pierre et Marie Curie, PARIS 6
http://people.isir.upmc.fr/sigaud

September 25, 2018

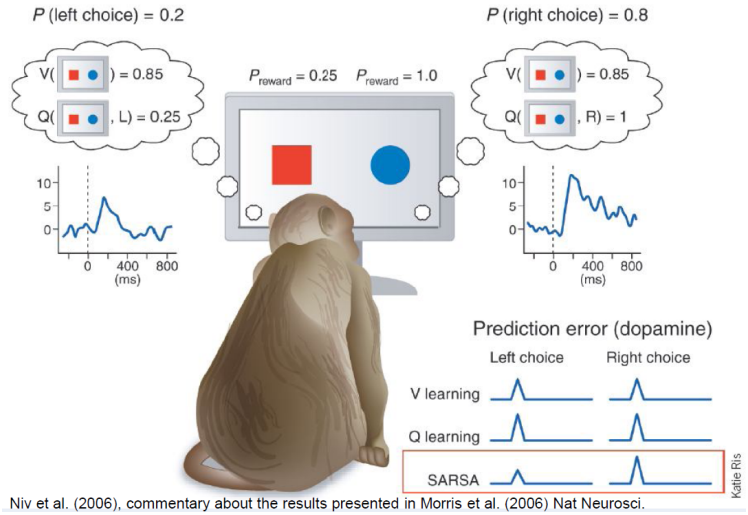
Why this class (1)?



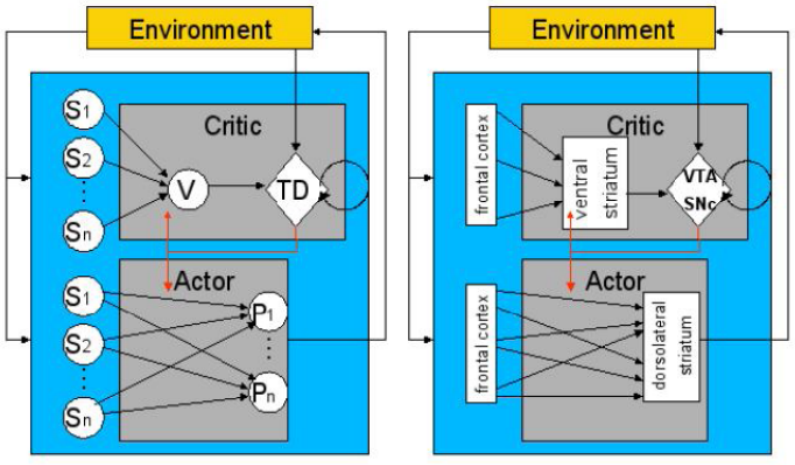


- A lot of buzz about deep reinforcement learning as an engineering tool

Why this class (2)?



Niv et al. (2006), commentary about the results presented in Morris et al. (2006) Nat Neurosci.



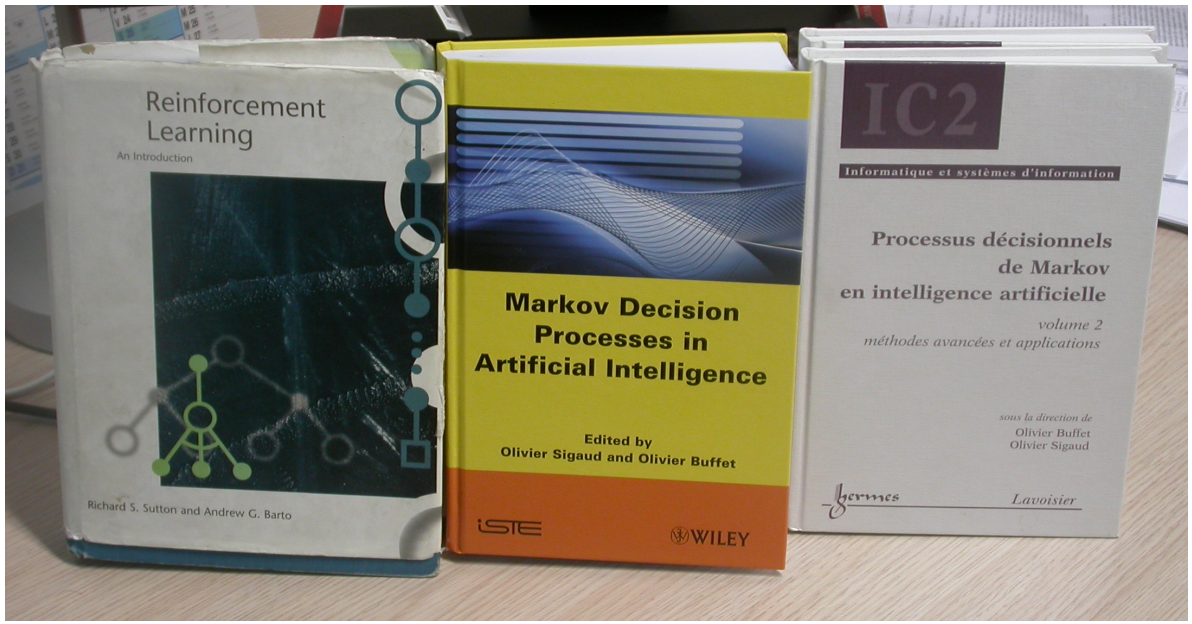
From Takahashi, Schoenbaum and Niv, Frontiers in Neurosciences, pp. 86-97, july 2008

- The reinforcement learning framework is relevant for computational neuroscience
- This aspect will be left out

Outline

- Goals of this class:
- Present the basics of discrete RL and dynamic programming
  - Dynamic programming
  - Model-free Reinforcement Learning
  - Actor-critic approach
  - Model-based Reinforcement Learning
- Then give a quick view of recent deep reinforcement learning research

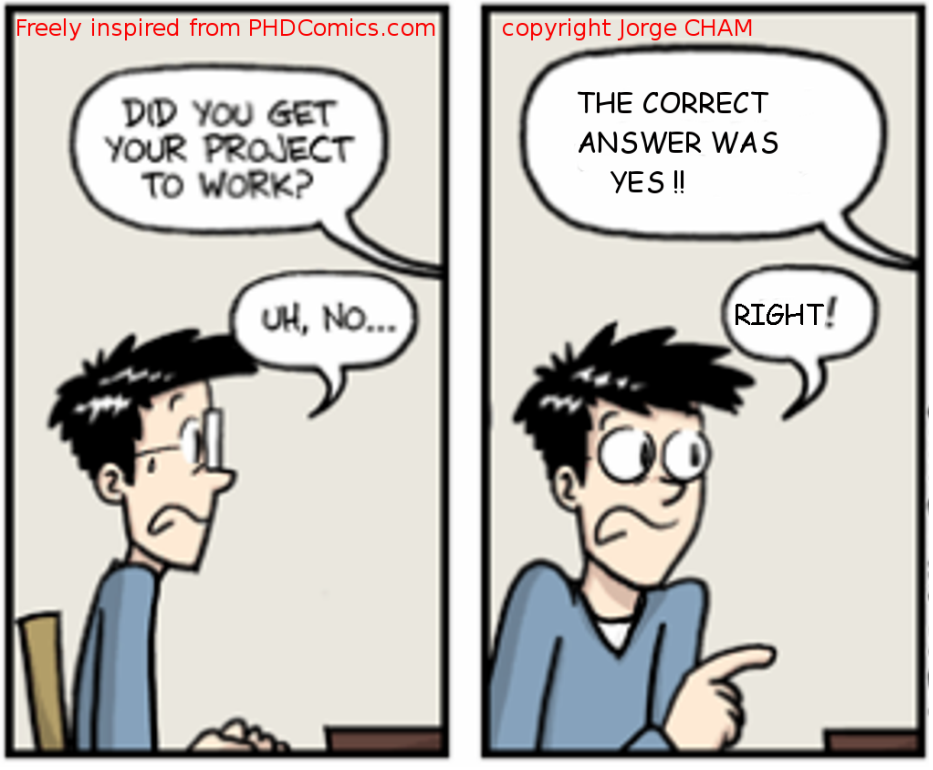
Introductory books



1. [Sutton & Barto, 1998]: the ultimate introduction to the field, in the discrete case
2. New edition available: `https://drive.google.com/file/d/1xeUDVGWGUUv1-ccUMAZHJLej2C7aAFWY/view`
3. [Buffet & Sigaud, 2008]: in french
4. [Sigaud & Buffet, 2010]: (improved) translation of 3

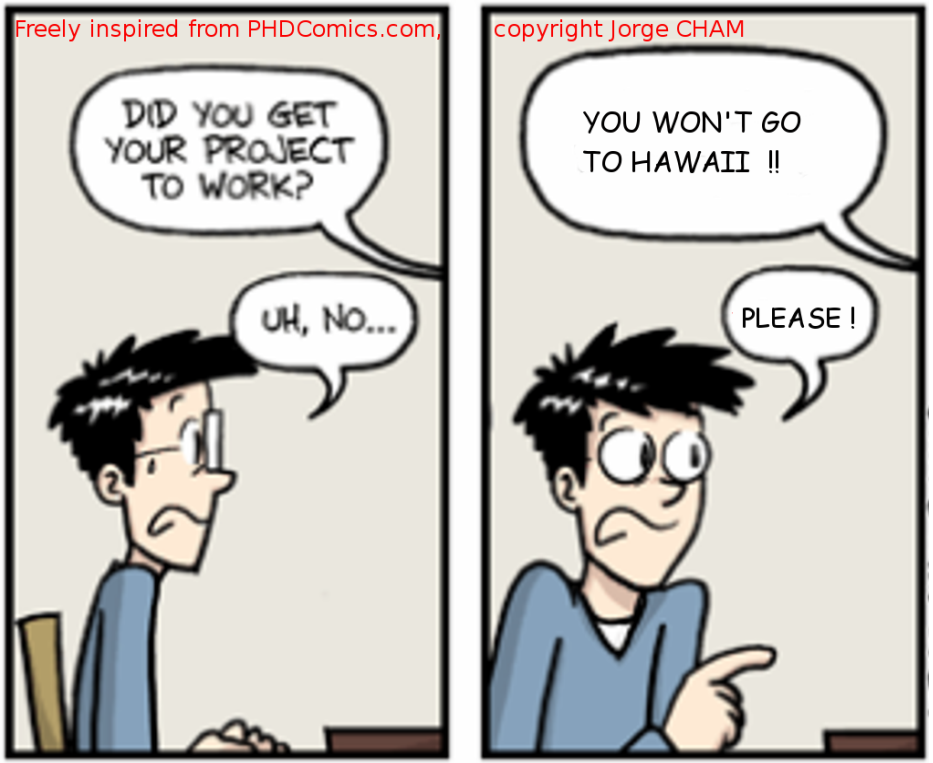
Sutton, R. S. & Barto, A. G. (1998) *Reinforcement Learning: An Introduction*. MIT Press.

Supervised learning



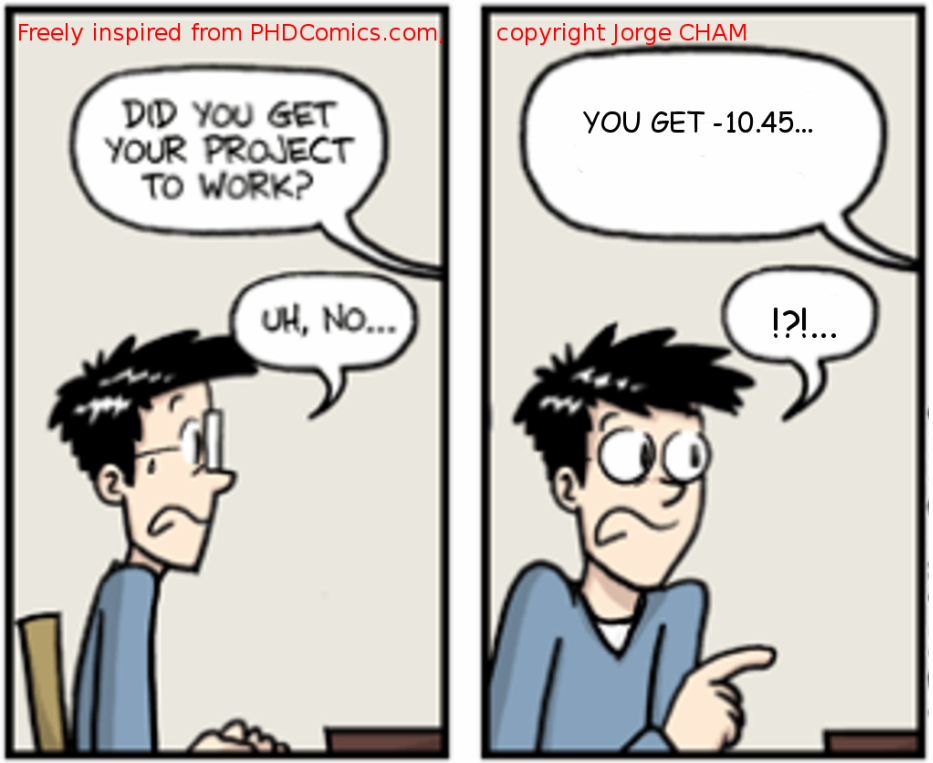
- The supervisor indicates to the agent the expected answer
- The agent corrects a model based on the answer
- Typical mechanism: gradient backpropagation, RLS
- Applications: classification, regression, function approximation...

Cost-Sensitive Learning



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

Reinforcement learning

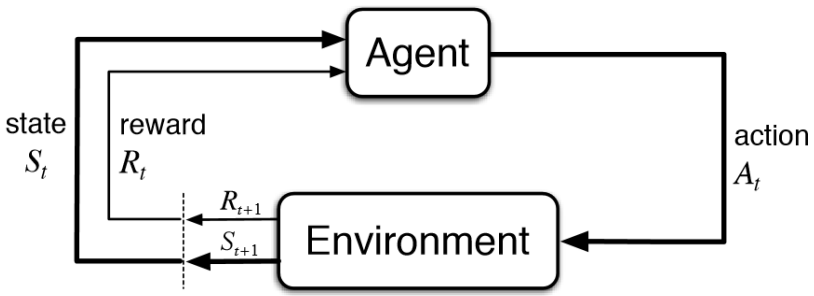


- In RL, the value signal is given as a scalar
- How good is -10.45?
- Necessity of exploration

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

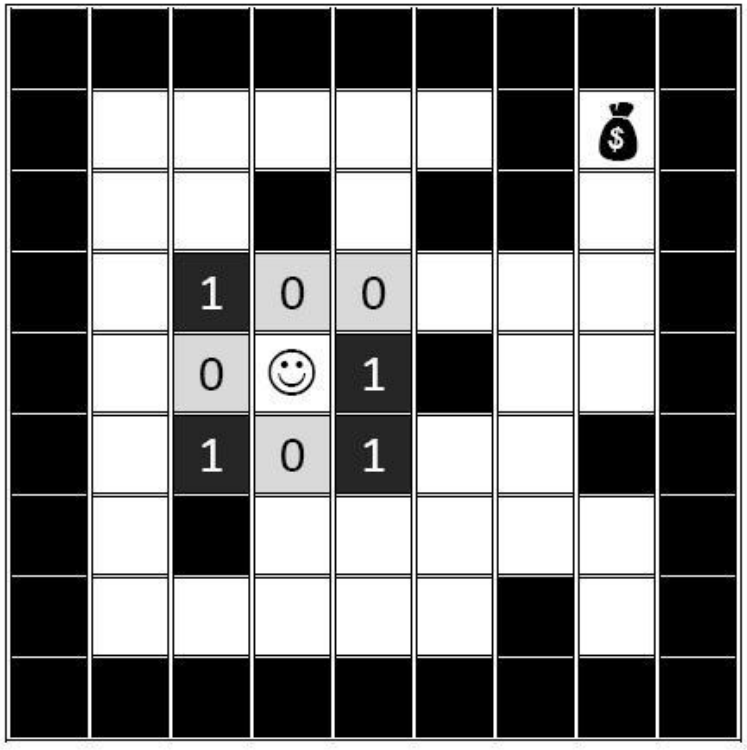
Markov Decision Processes

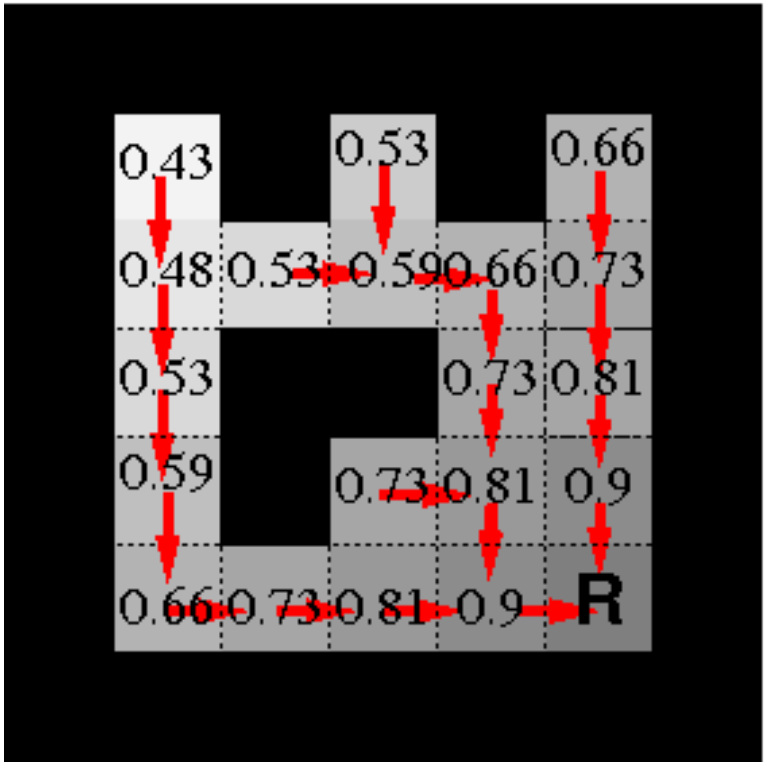


- S: states space
- A: action space
- $T : S \times A \rightarrow \Pi(S)$: transition function
- $r : S \times A \rightarrow \mathbb{R}$: reward function

- An MDP defines s^{t+1} and r^{t+1} as $f(s_t, a_t)$
- It describes a problem, not a solution
- Markov property : $p(s^{t+1}|s^t, a^t) = p(s^{t+1}|s^t, a^t, s^{t-1}, a^{t-1}, ...s^0, a^0)$
- Reactive agents $a_{t+1} = f(s_t)$, without internal states nor memory
- In an MDP, a memory of the past does not provide any useful advantage

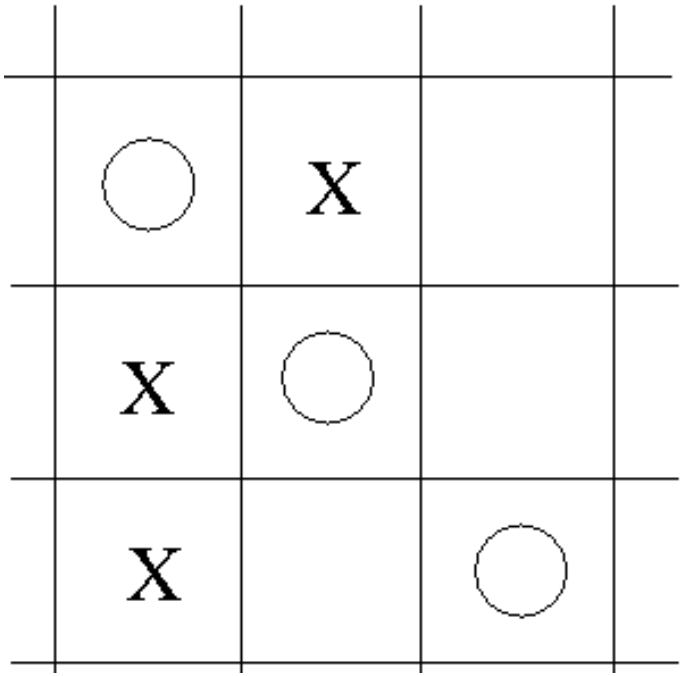
Markov property: Limitations





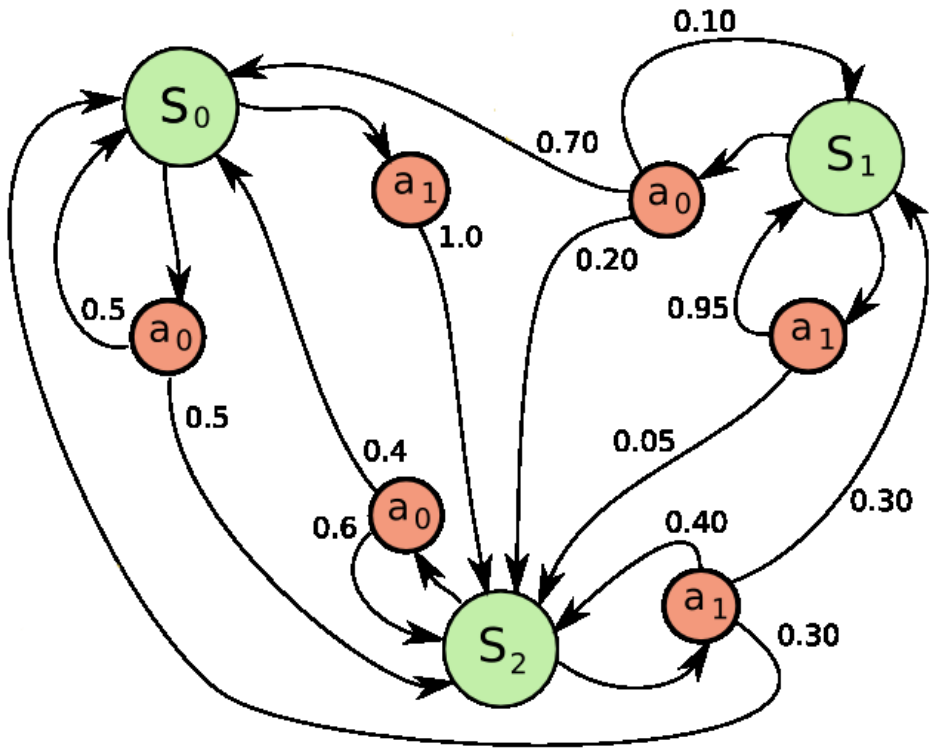
- Markov property is not verified if:
  - the state does not contain all useful information to take decisions
  - or if the next depends on decisions of several agents
  - or if transitions depend on time

Counter-example: tic-tac-toe



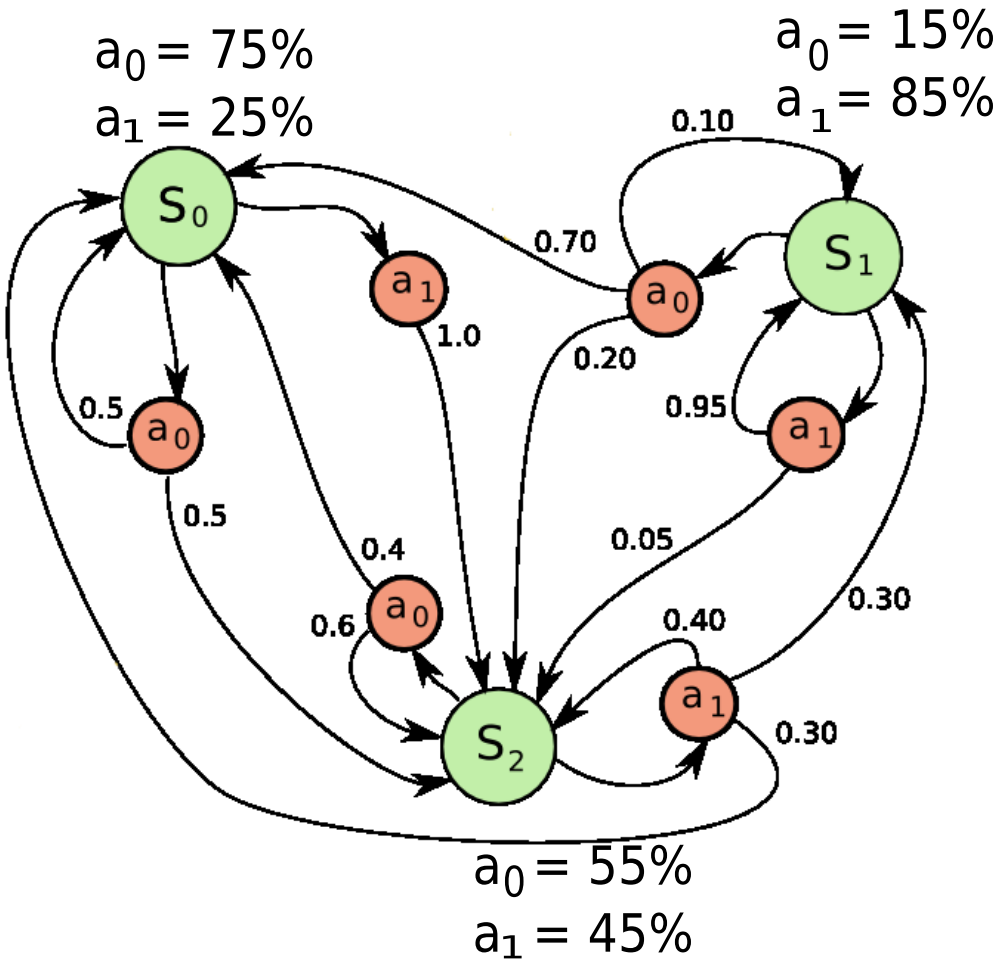
- The state is not always a location
- The opponent can be seen as a stochastic part of the environment
- Better framework = Markov games

A stochastic problem



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

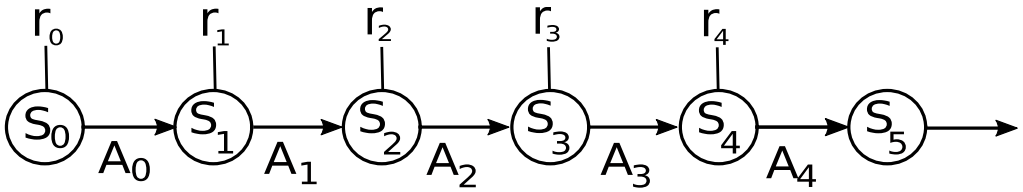
A stochastic policy



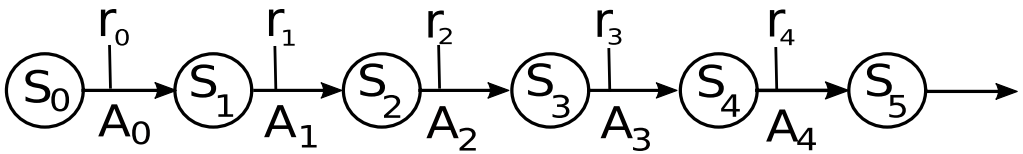
- For any MDP, there exists a deterministic policy that is optimal

Rewards over a Markov chain: on states or action?

- Reward over states

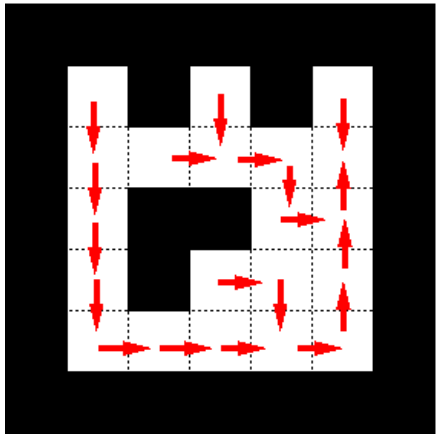


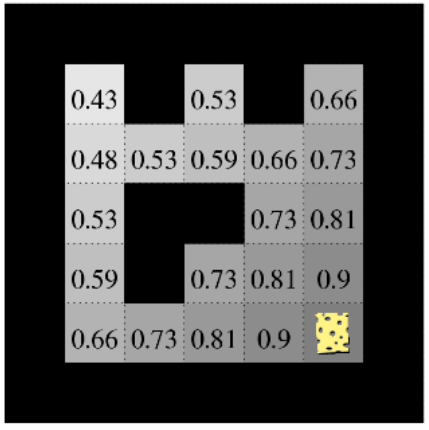
- Reward over actions in states



- Below, we assume the latter (we note $r(s, a)$)

Policy and value functions



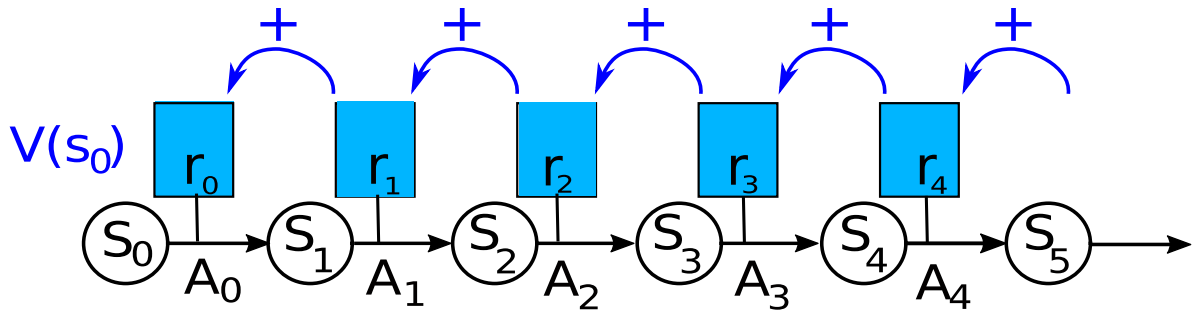


state / action	a_0	a_1	a_2	a_3
e_0	0.66	0.88	0.81	0.73
e_1	0.73	0.63	0.9	0.43
e_2	0.73	0.9	0.95	0.73
e_3	0.81	0.9	1.0	0.81
e_4	0.81	1.0	0.81	0.9
e_5	0.9	1.0	0.0	0.9

- Goal: find a policy $\pi : S \to A$ maximizing the agregation of reward on the long run
- 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

Agregation criteria

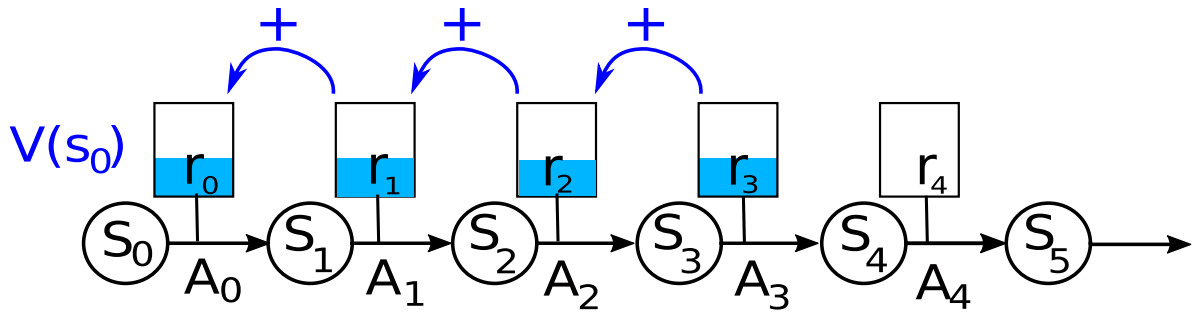
- The computation of value functions assumes the choice of an agregation criterion (discounted, average, etc.)



- The sum over a infinite horizon may be infinite, thus hard to compare
- Mere sum (finite horizon N): $V^{\pi}(S_0) = r_0 + r_1 + r_2 + \ldots + r_N$

Agregation criteria

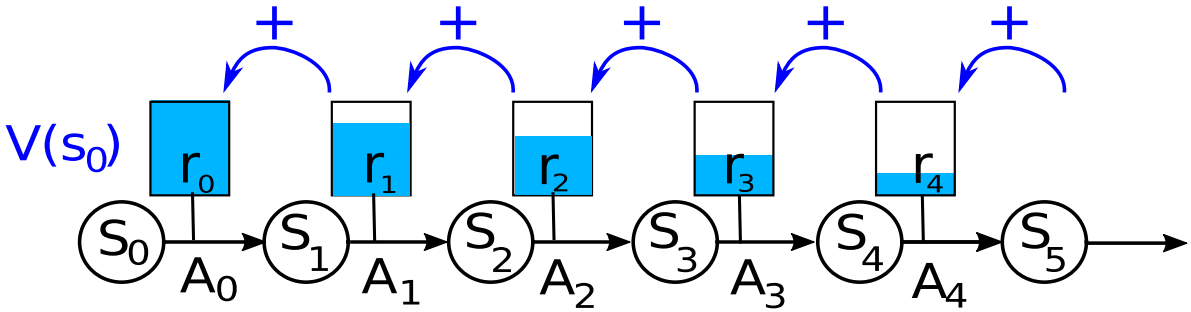
- The computation of value functions assumes the choice of an agregation criterion (discounted, average, etc.)



- Average criterion on a window: $V^{\pi}(S_0) = \frac{r_0+r_1+r_2}{3}$...

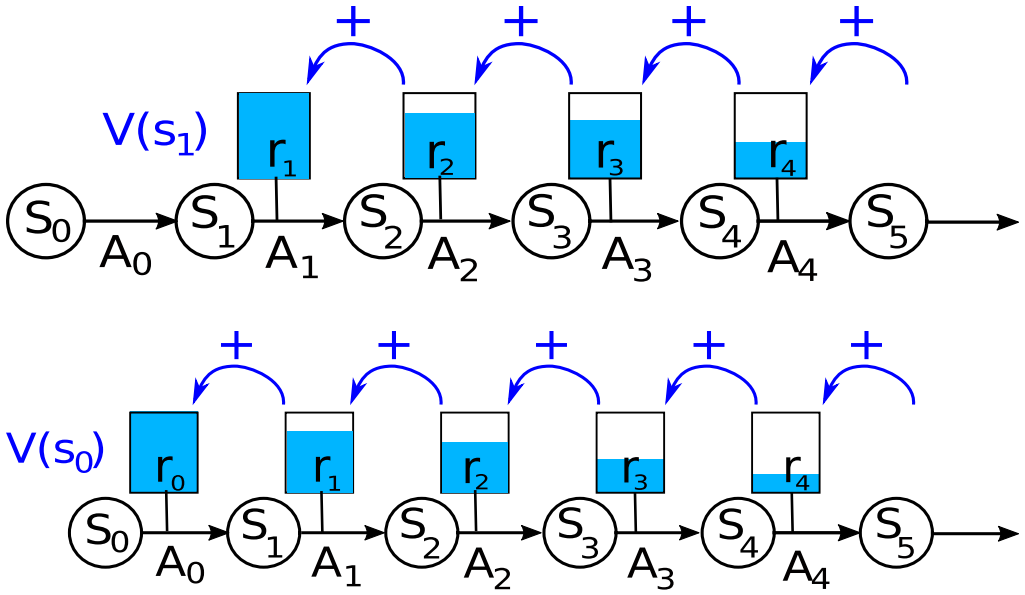
Agregation criteria

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



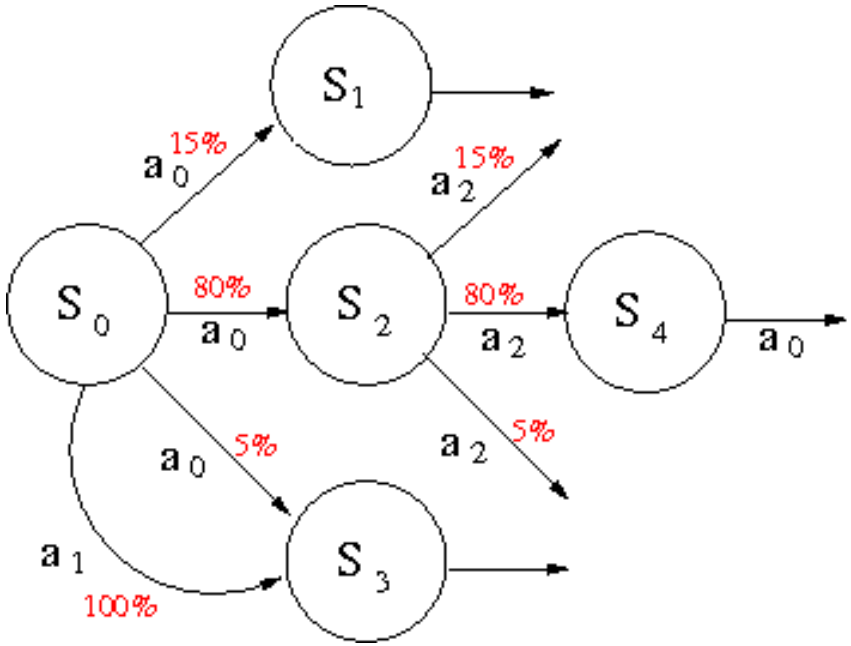
- Discounted criterion: $V^{\pi}(s_{t_0}) = \sum_{t=t_0}^{\infty} \gamma^t r(s_t, \pi(s_t))$
- $\gamma \in [0, 1]$: discount factor
  - if $\gamma = 0$, sensitive only to immediate reward
  - if $\gamma = 1$, future rewards are as important as immediate rewards
- The discounted case is the most used

Bellman equation over a Markov chain: recursion



- Given the discounted reward agregation criterion:
- $V(s_0) = r_0 + \gamma V(s_1)$

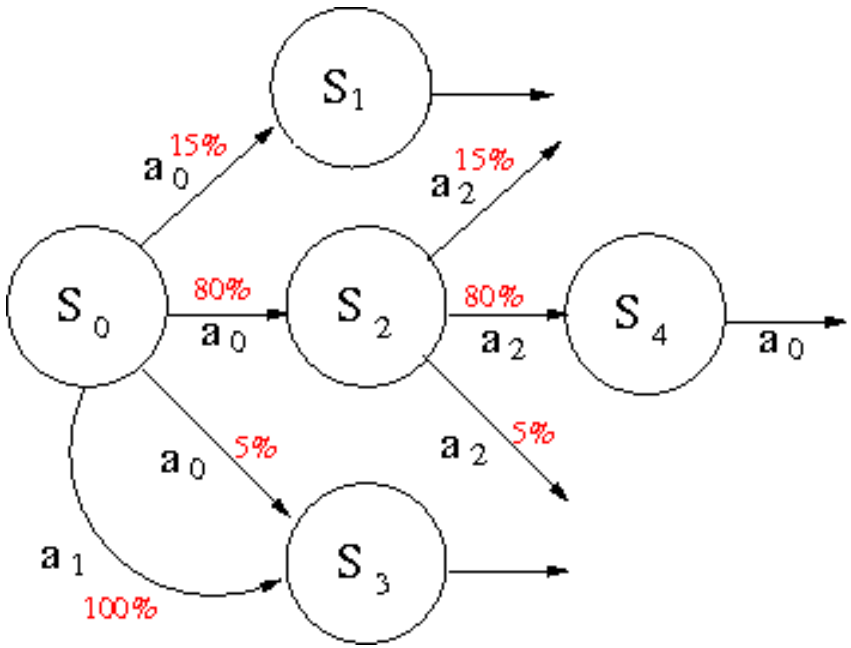
Bellman equation: general case



- Generalisation of the recursion $V(s_0) = r_0 + \gamma V(s_1)$ over all possible trajectories
- 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 the recursion $V(s_0) = r_0 + \gamma V(s_1)$ over all possible trajectories
- Stochastic π:

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

Bellman operator and dynamic programming

- We get $V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V^\pi(s')$
- We call Bellman operator (noted T^π) the application

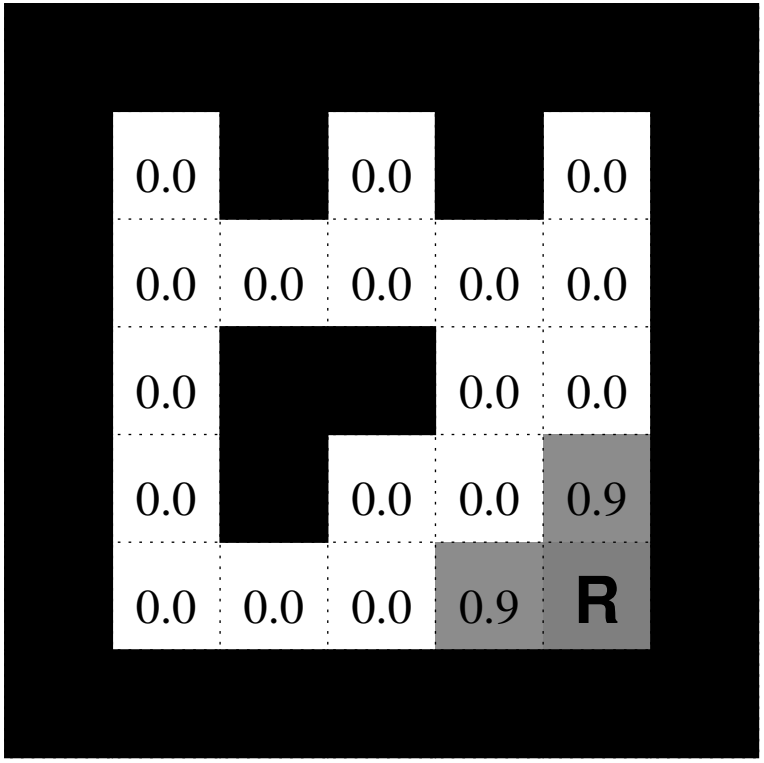
$$V^\pi(s) \leftarrow r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s))$$

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

$$V^\pi(s) \leftarrow \max_{a \in A} \Big[r(s, a) + \gamma \sum_{s'} p(s'|s, a) V(s') \Big]$$

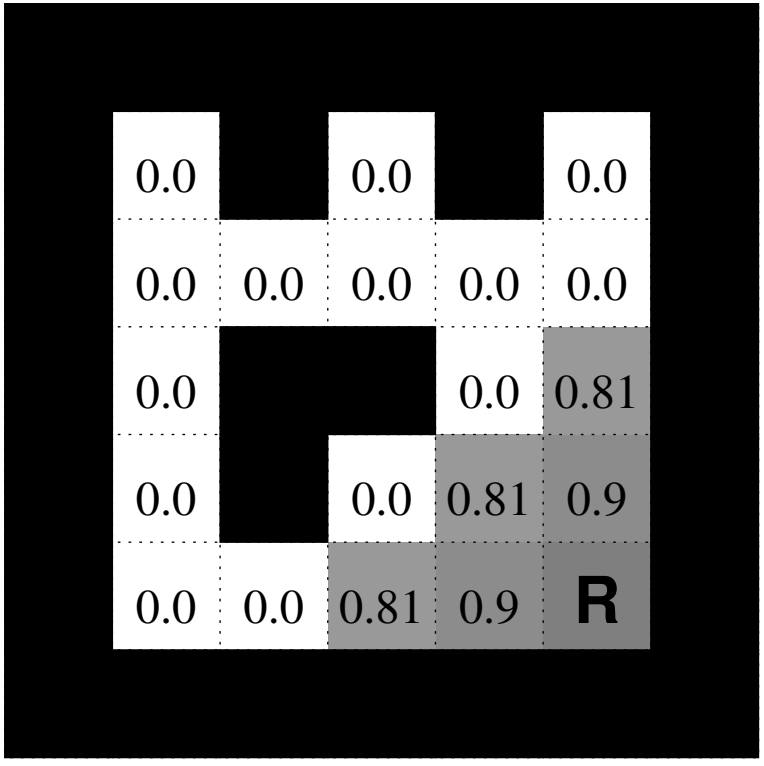
- The optimal value function is a fixed-point of the Bellman optimality operator T^*: $V^* = T^* V^*$
- Value iteration: $V_{i+1} \leftarrow T^* V_i$
- Policy Iteration: policy evaluation (with $V^\pi_{i+1} \leftarrow T^\pi V^\pi_i$) + policy improvement with
  $\forall s \in S, \pi'(s) \leftarrow \arg\max_{a \in A} \sum_{s'} p(s'|s, a)[r(s, a) + \gamma V^\pi(s')]$

Value Iteration in practice



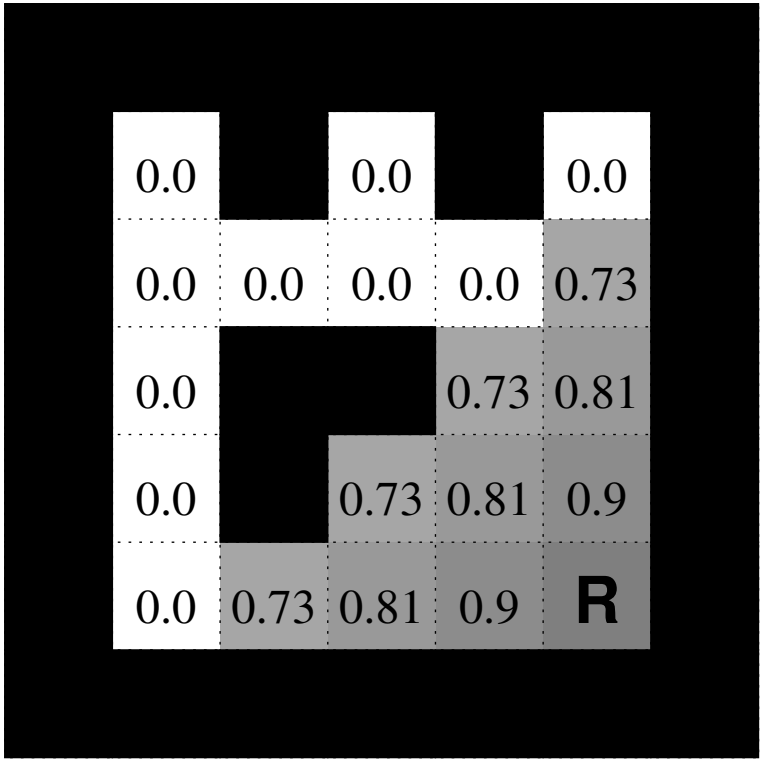
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



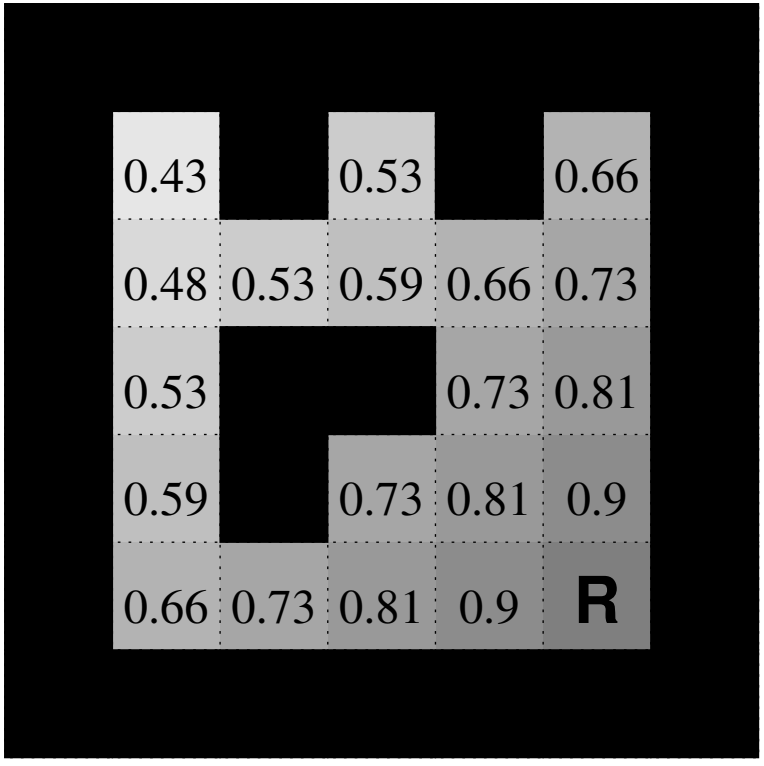
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



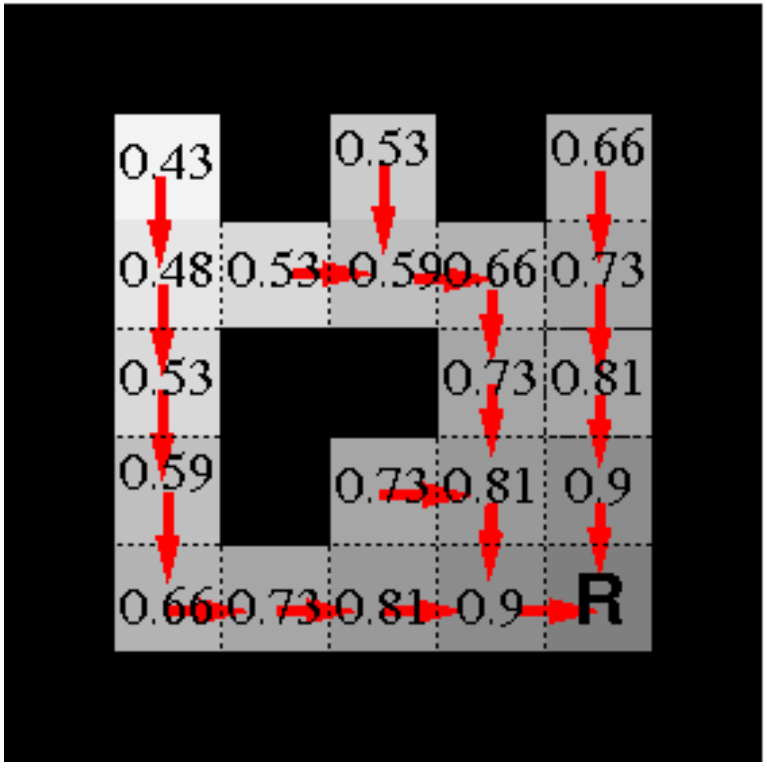
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



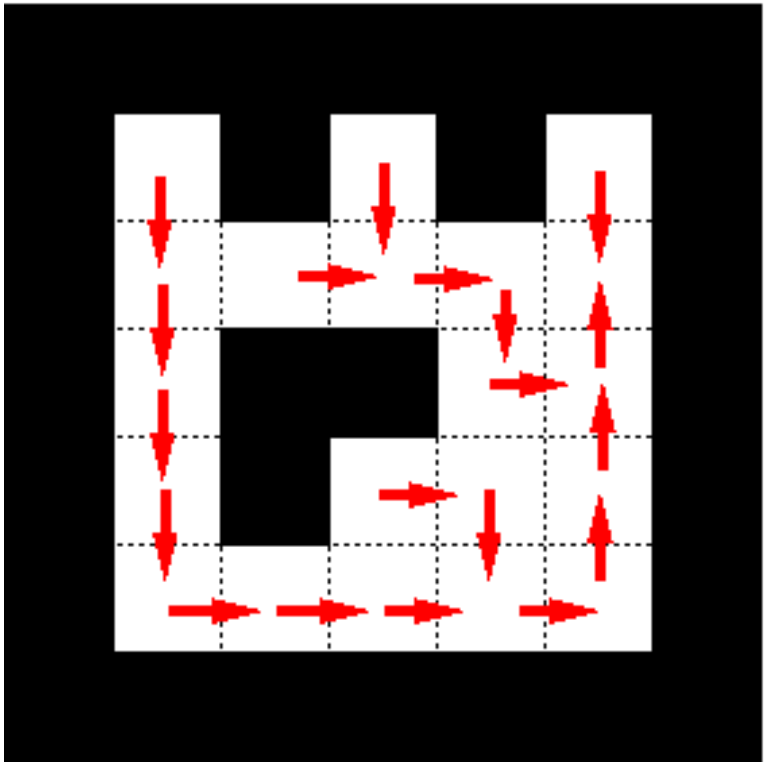
$$\forall s \in S, V_{i+1}(s) \leftarrow \max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_i(s') \Big]$$

Value Iteration in practice



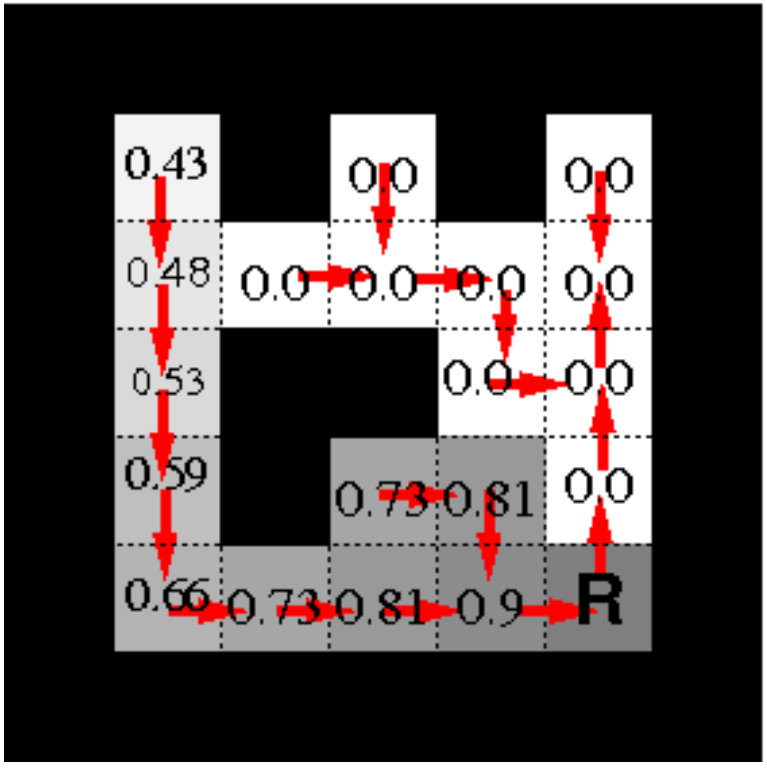
$$\pi^*(s) = \arg\max_{a \in A} \Big[r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^*(s') \Big]$$

Policy Iteration in practice



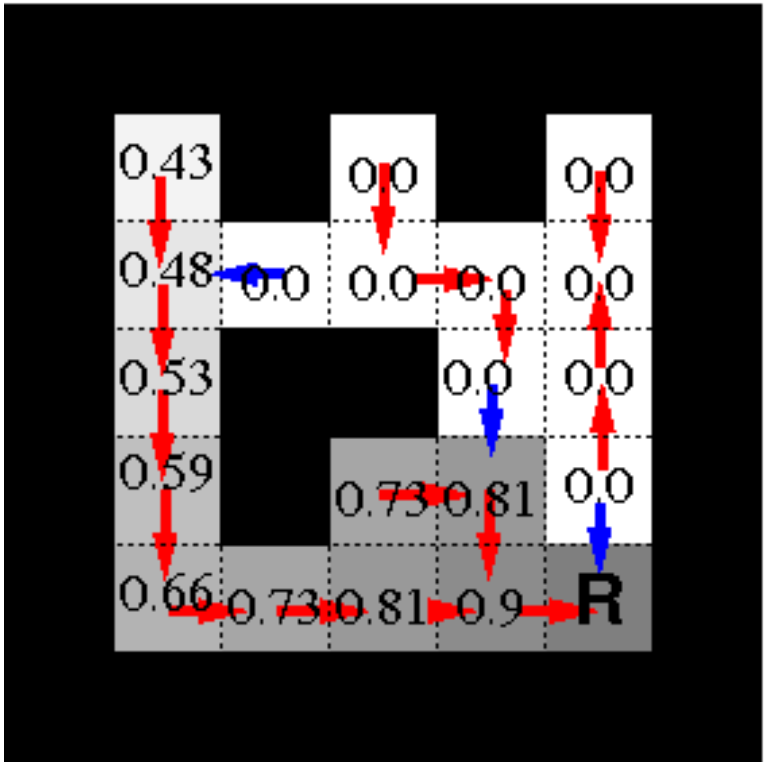
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



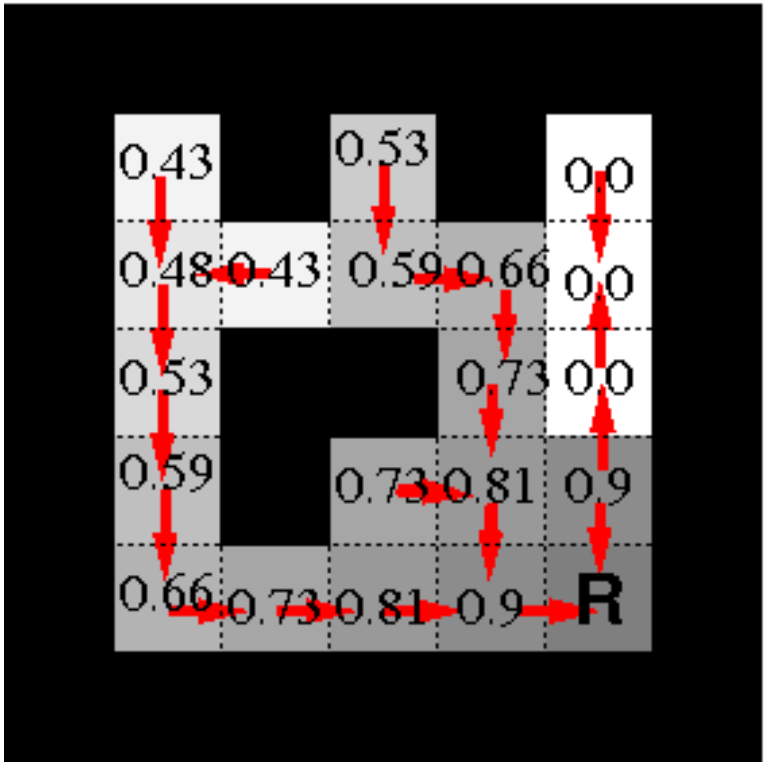
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



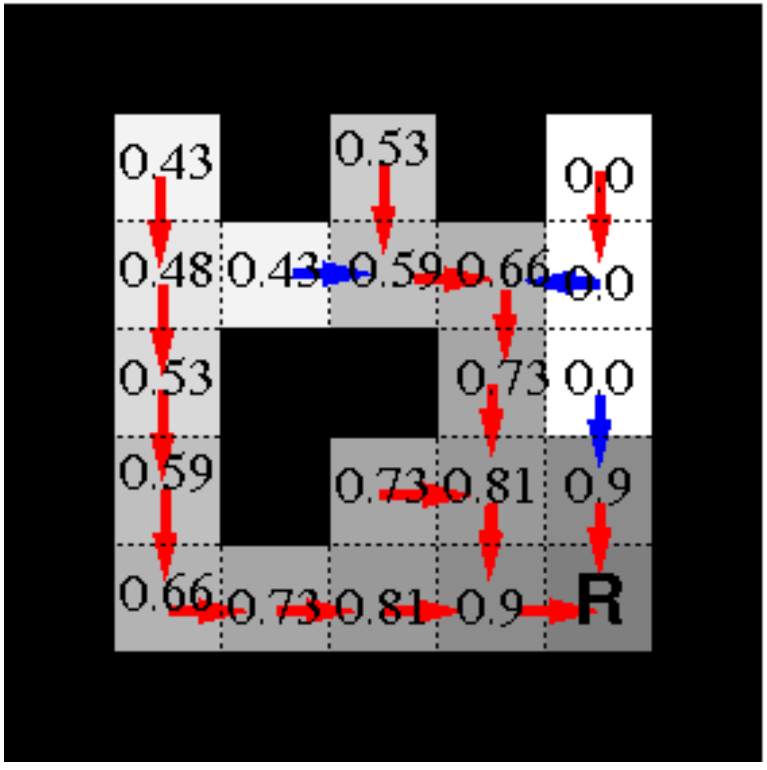
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



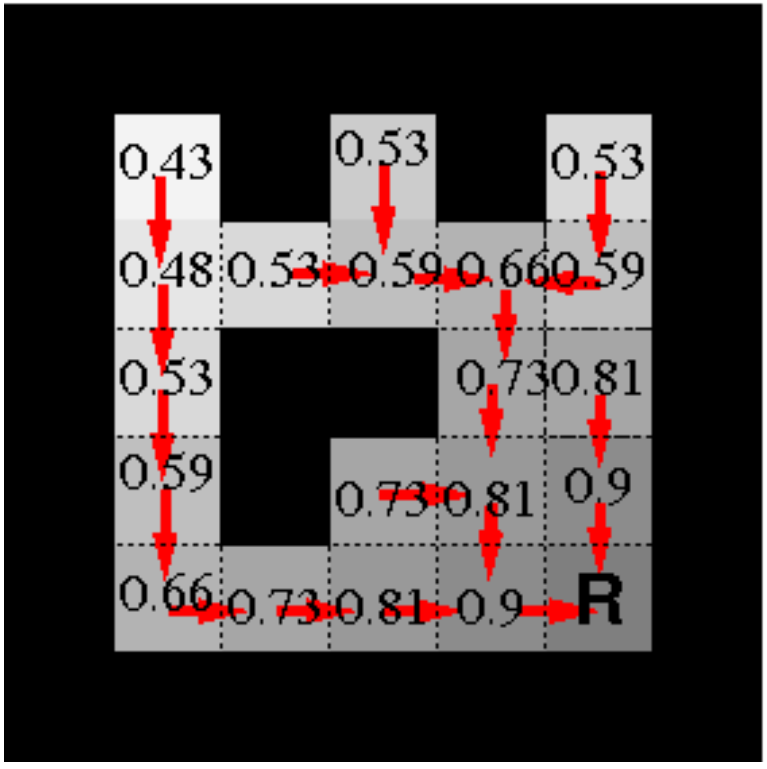
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



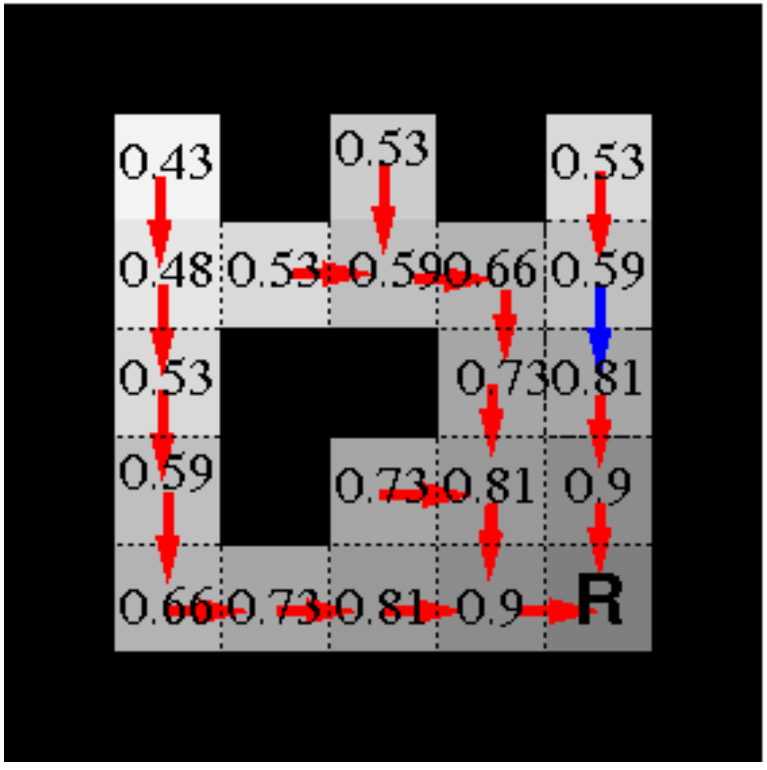
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



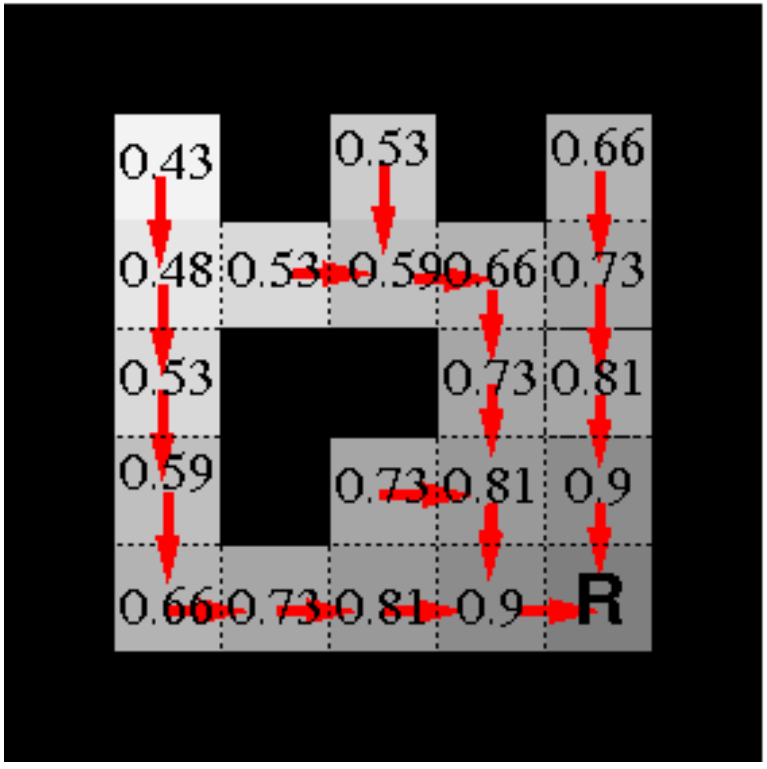
$$\forall s \in S, \pi_{i+1}(s) \leftarrow improve(\pi_i(s), V_i(s))$$

Policy Iteration in practice



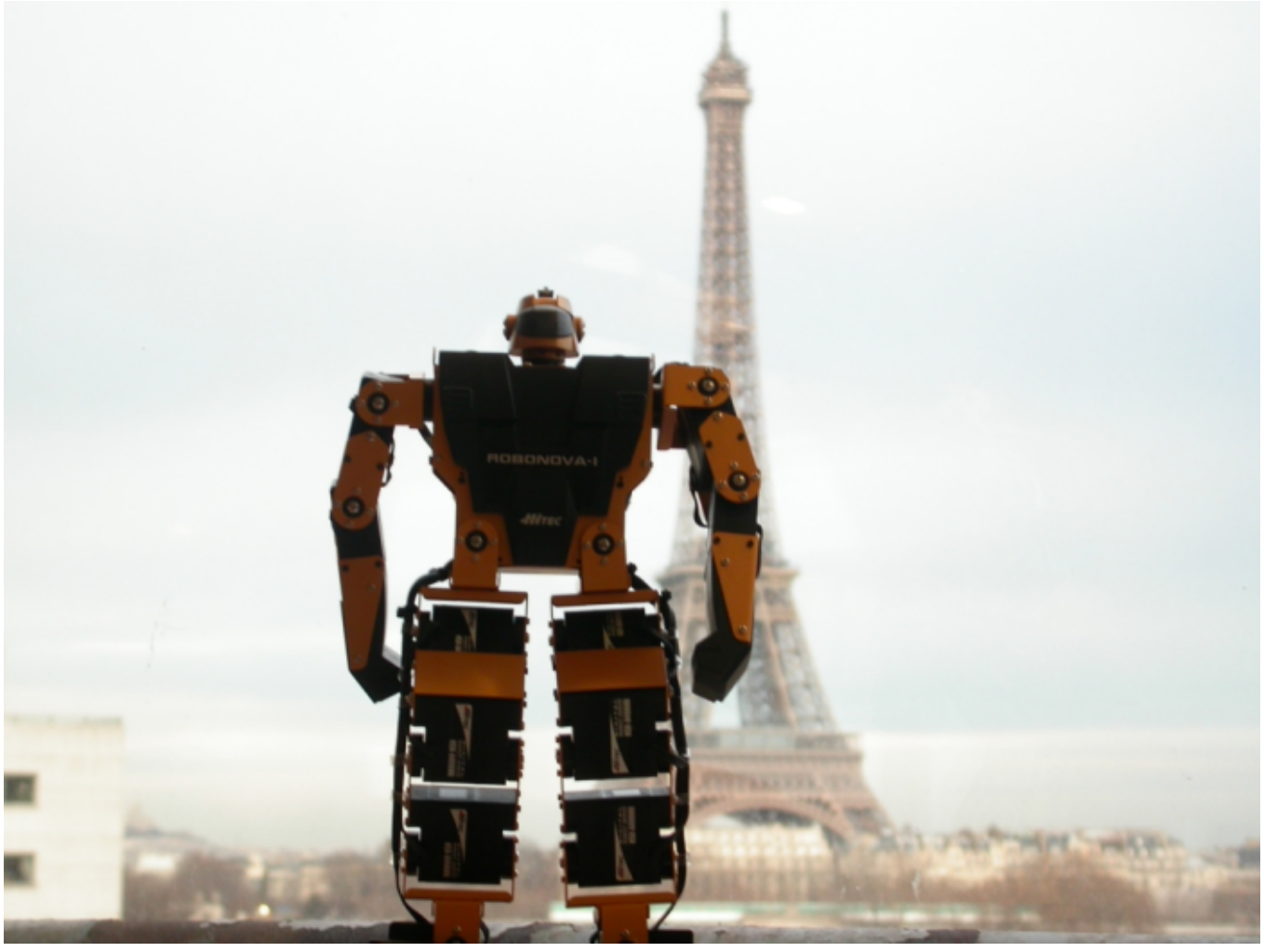
$$\forall s \in S, V_i(s) \leftarrow evaluate(\pi_i(s))$$

Policy Iteration in practice



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

Any question?



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

Families of methods

- Critic : (action) value function $\rightarrow$ evaluation of the policy
- Actor: the policy itself
- Critic-only methods: iterates on the value function up to convergence without storing policy, then computes optimal policy. Typical examples: value iteration, Q-learning, Sarsa
- Actor-only methods: explore the space of policy parameters. Typical example: CMA-ES
- Actor-critic methods: update in parallel one structure for the actor and one for the critic. Typical examples: policy iteration, many AC algorithms
- Q-learning and Sarsa look for a global optimum, AC looks for a local one

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:
- $V(s_t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ...$
- $V(s_{t+1}) = r_{t+2} + \gamma(r_{t+3} + \gamma^2 r_{t+4} + ...$
- Thus $V(s_t) = r_{t+1} + \gamma V(s_{t+1})$
- $\delta_k = r_{k+1} + \gamma V(s_{k+1}) - V(s_k)$: measures the error between current values of V and the values they should have

Monte Carlo methods

- Much used in games (Go...) to evaluate a state
- Generate a lot of trajectories: $s_0, s_1, \ldots, s_N$ with observed rewards $r_0, r_1, \ldots, r_N$
- Update state values $V(s_k)$, $k = 0, \ldots, N-1$ with:

$$V(s_k) \leftarrow V(s_k) + \alpha(s_k)(r_k + r_{k+1} + \cdots + r_N - V(s_k))$$

- It uses the average estimation method (1)

Temporal Difference (TD) Methods

- Temporal Difference (TD) methods combine the properties of DP methods and Monte Carlo methods:
- in Monte Carlo, T and r are unknown, but the value update is global, trajectories are needed
- in DP, T and r are known, but the value update is local
- TD: as in DP, $V(s_t)$ is updated locally given an estimate of $V(s_{t+1})$ and T and r are unknown
- Note: Monte Carlo can be reformulated incrementally using the temporal difference δ_k update

Policy evaluation: TD(0)

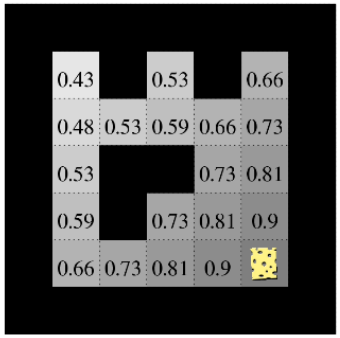
- Given a policy π, the agent performs a sequence $s_0, a_0, r_1, \cdots, s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, \cdots$
- $V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$
- Combines the TD update (propagation from $V(s_{t+1})$ to $V(s_t)$) from DP and the incremental estimation method from Monte Carlo
- Updates are local from s_t, s_{t+1} and r_{t+1}
- Proved in 1994

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

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:
  - Work with $Q(s, a)$ rather than $V(s)$.
  - Learn a model of T: model-based (or indirect) reinforcement learning
  - Actor-critic methods (simultaneously learn V and update π)

Value function and Action Value function



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

- 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.

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 proved given infinite exploration [Dayan & Sejnowski, 1994]

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

Q-Learning in practice

(Q-learning: the movie)

- 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

From $Q(s, a)$ to Actor-Critic (1)

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

- In $Q - \textit{learning}$, given a $Q - \textit{Table}$, one must determine the max at each step
- This becomes expensive if there are numerous actions

From $Q(s, a)$ to Actor-Critic (2)

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

- One can store the best value for each state
- Then one can update the max by just comparing the changed value and the max
- No more maximum over actions (only in one case)

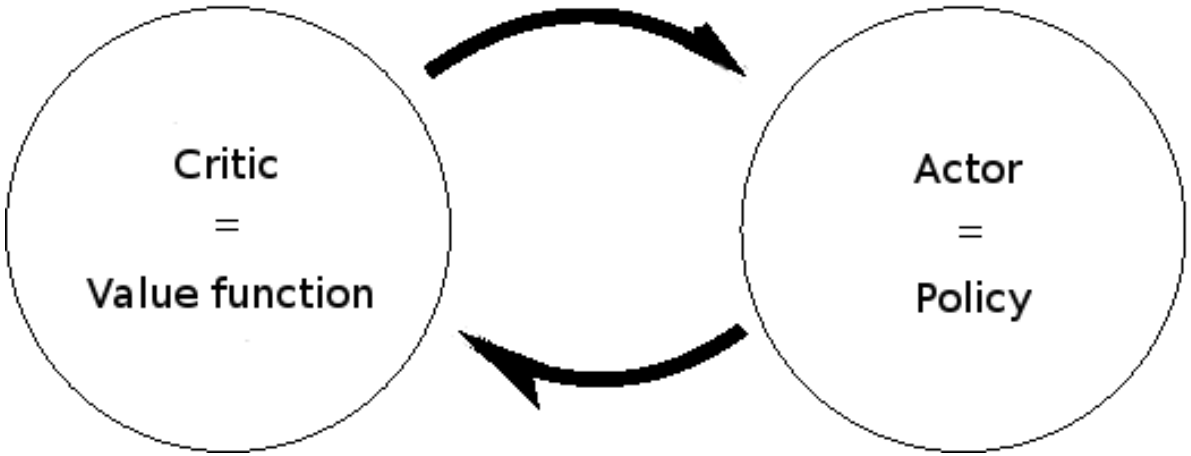
From $Q(s, a)$ to Actor-Critic (3)

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

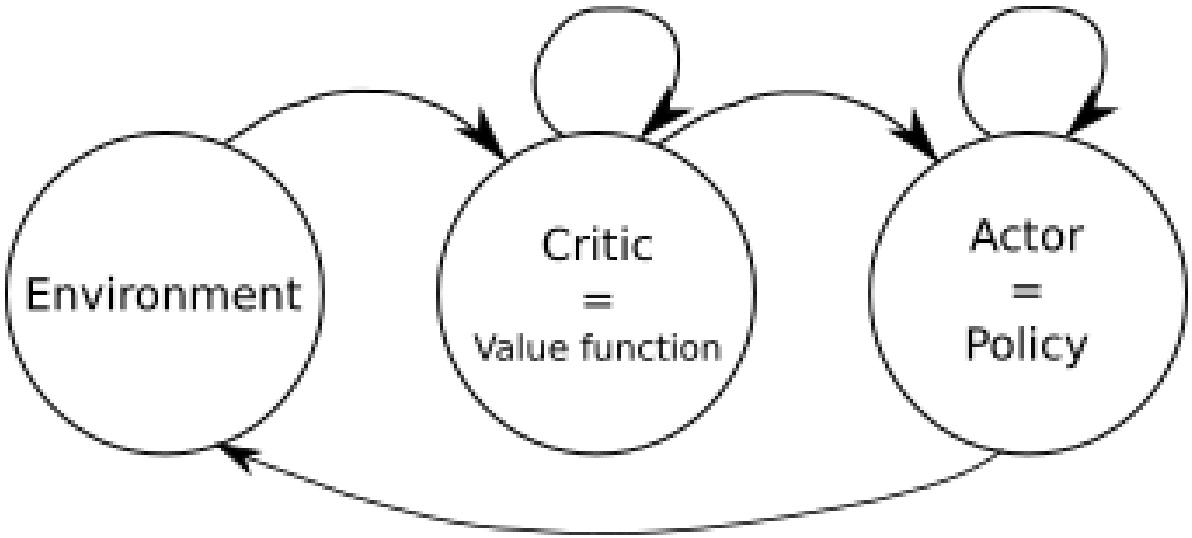
- Storing the max is equivalent to storing the policy
- Update the policy as a function of value updates
- Basic actor-critic scheme

Dynamic Programming and Actor-Critic (1)



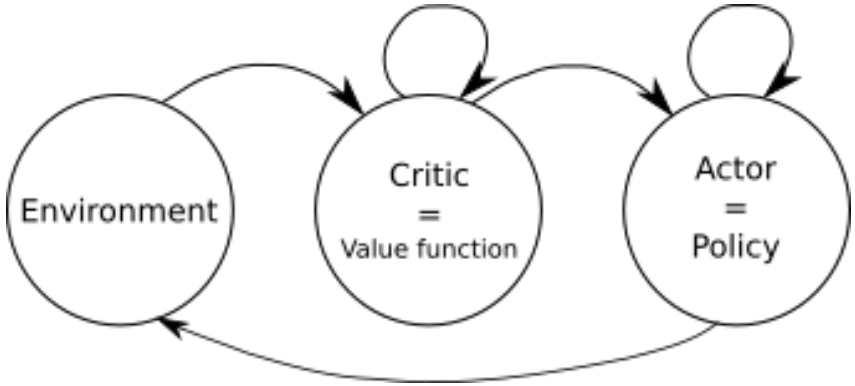
- In both PI and AC, the architecture contains a representation of the value function (the critic) and the policy (the actor)
- In PI, the MDP (T and r) is known
- PI alternates two stages:
  1. Policy evaluation: update ($V(s)$) or ($Q(s,a)$) given the current policy
  2. Policy improvement: follow the value gradient

Dynamic Programming and Actor-Critic (2)



- 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

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 $V_k(s) \leftarrow V_k(s) + \alpha_k \delta_k$
- Actor: $P^\pi(a|s) = P^\pi(a|s) + \alpha_k \prime \delta_k$
- 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)

Reminder: TD 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:
- $V(s_t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ...$
- $V(s_{t+1}) = r_{t+2} + \gamma(r_{t+3} + \gamma^2 r_{t+4} + ...$
- Thus $V(s_t) = r_{t+1} + \gamma V(s_{t+1})$
- $\delta_k = r_{k+1} + \gamma V(s_{k+1}) - V(s_k)$: measures the error between current values of V and the values they should have

Monte Carlo (MC) methods

- Much used in games (Go...) to evaluate a state
- Generate a lot of trajectories: $s_0, s_1, \ldots, s_N$ with observed rewards $r_0, r_1, \ldots, r_N$
- Update state values $V(s_k)$, $k = 0, \ldots, N-1$ with:

$$V(s_k) \leftarrow V(s_k) + \alpha(s_k)(r_k + r_{k+1} + \cdots + r_N - V(s_k))$$

- It uses the average estimation method (1)

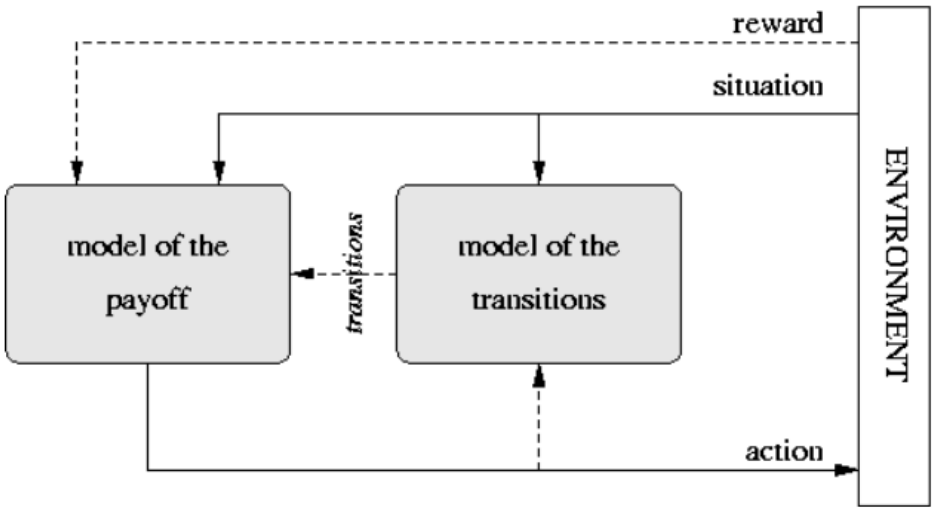
TD vs MC

- Temporal Difference (TD) methods combine the properties of DP methods and Monte Carlo methods:
- In Monte Carlo, T and r are unknown, but the value update is global, trajectories are needed
- In DP, T and r are known, but the value update is local
- TD: as in DP, $V(s_t)$ is updated locally given an estimate of $V(s_{t+1})$ and T and r are unknown
- Note: Monte Carlo can be reformulated incrementally using the temporal difference δ_k update

Eligibility traces

- To improve over Q-learning
- Naive approach: store all (s, a) pair and back-propagate values
- Limited to finite horizon trajectories
- Speed/memory trade-off
- TD(λ), SARSA (λ) and Q(λ): more sophisticated approach to deal with infinite horizon trajectories
- A variable $e(s)$ is decayed with a factor λ after s was visited and reinitialized each time s is visited again
- TD(λ): $V(s) \leftarrow V(s) + \alpha\delta e(s)$, (similar for SARSA ($\lambda$) and Q($\lambda$)),
- If $\lambda = 0$, $e(s)$ goes to 0 immediately, thus we get TD(0), SARSA or Q-learning
- TD(1) = Monte-Carlo...

Model-based Reinforcement Learning



- General idea: planning with a learnt model of T and r is performing back-ups "in the agent's head" ([Sutton, 1990, Sutton, 1991])
- Learning T and r is an incremental self-supervised learning problem
- Several approaches:
  - Draw random transition in the model and apply TD back-ups
  - Dyna-PI, Dyna-Q, Dyna-AC
  - Better propagation: Prioritized Sweeping

Moore, A. W. & Atkeson, C. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130.

Dyna architecture and generalization

(Dyna-like video (good model))
(Dyna-like video (bad model))

- Thanks to the model of transitions, Dyna can propagate values more often
- Problem: in the stochastic case, the model of transitions is in $card(S) \times card(S) \times card(A)$
- Usefulness of compact models
- MACS: Dyna with generalisation (Learning Classifier Systems)
- SPITI: Dyna with generalisation (Factored MDPs)

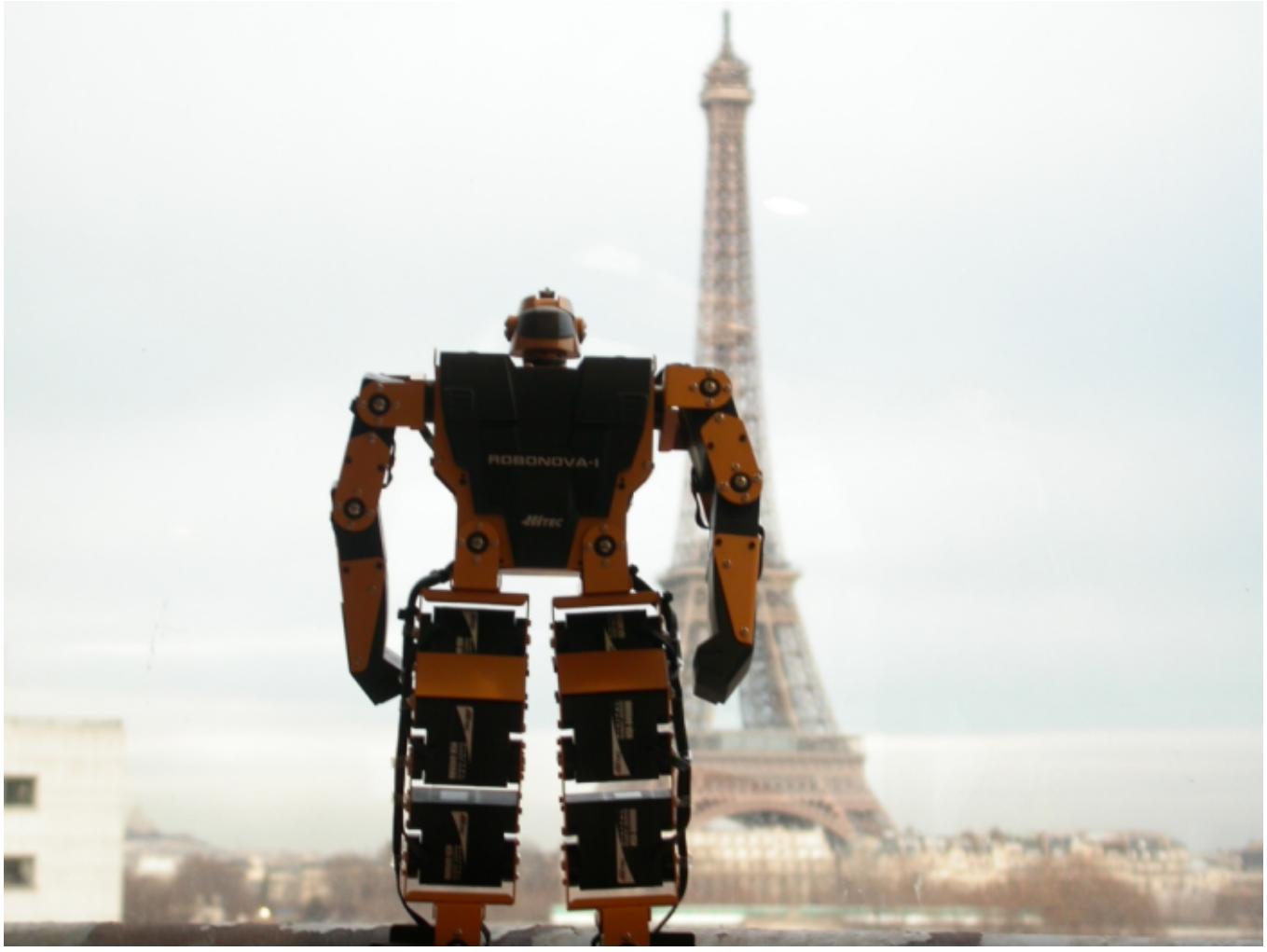
Gérard, P., Meyer, J.-A., & Sigaud, O. (2005) Combining latent learning with dynamic programming in MACS. *European Journal of Operational Research*, 160:614–637.

Degris, T., Sigaud, O., & Wuillemin, P.-H. (2006) Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems. *Proceedings of the 23rd International Conference on Machine Learning (ICML'2006)*, pages 257–264

A few messages

- Dynamic programming and reinforcement learning methods can be split into pure actor, pure critic and actor-critic methods
- Dynamic programming, value iteration, policy iteration are when you know the transition and reward functions
- Actor critic RL is a model-free, PI-like algorithm
- Model-based RL combines dynamic programming and model learning

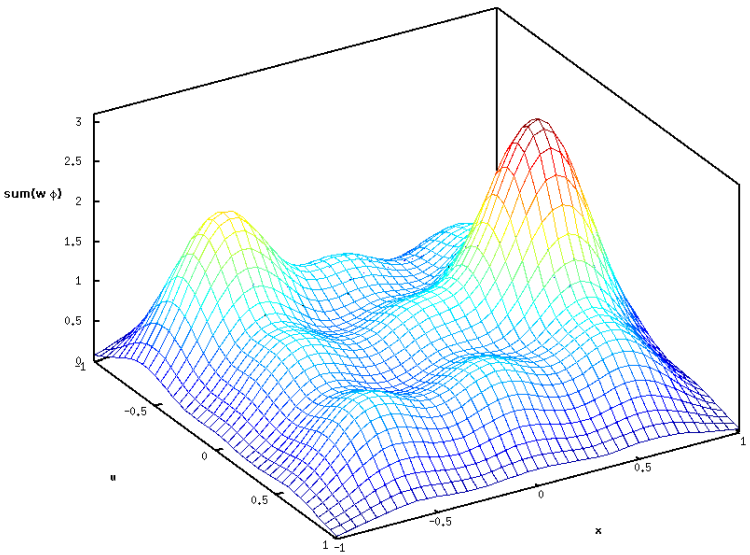
Any question?



Questions

- SARSA is on-policy and Q-learning is off-policy
  Right or Wrong ?
- The actor-critic approach is model-based
  Right or Wrong ?
- In SARSA, the policy is represented implicitly through the critic
  Right or Wrong ?

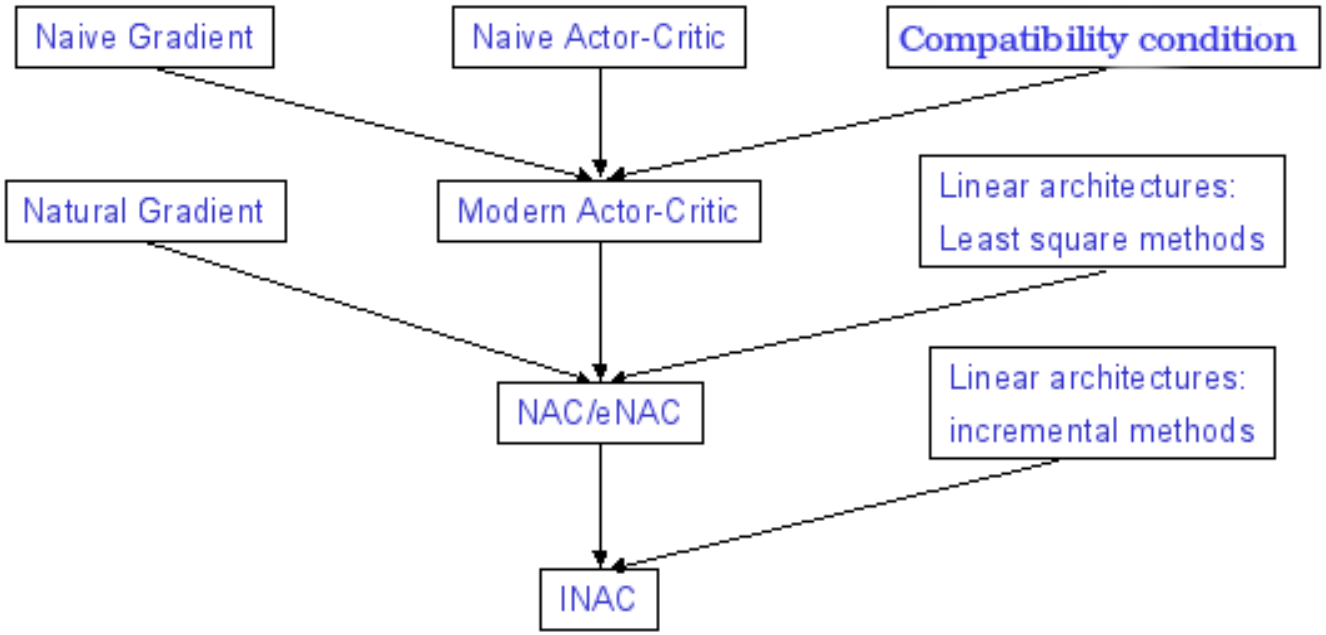
Parametrized representations



- To represent a continuous function, use features and a vector of parameters
- Learning tunes the weights
- Linear architecture: linear combination of features

- A deep neural network is not a linear architectures: deep layer parameters tune the features
- Parametrized representations:
  - In critic-based methods, like DQN: of the critic $Q(s_t, a_t|\theta)$
  - In policy gradient methods: of the policy $\pi_{\mathbf{w}}(a_t|s_t)$
  - In actor-critic methods: both

Quick history of previous attempts (J. Peters' and Sutton's groups)



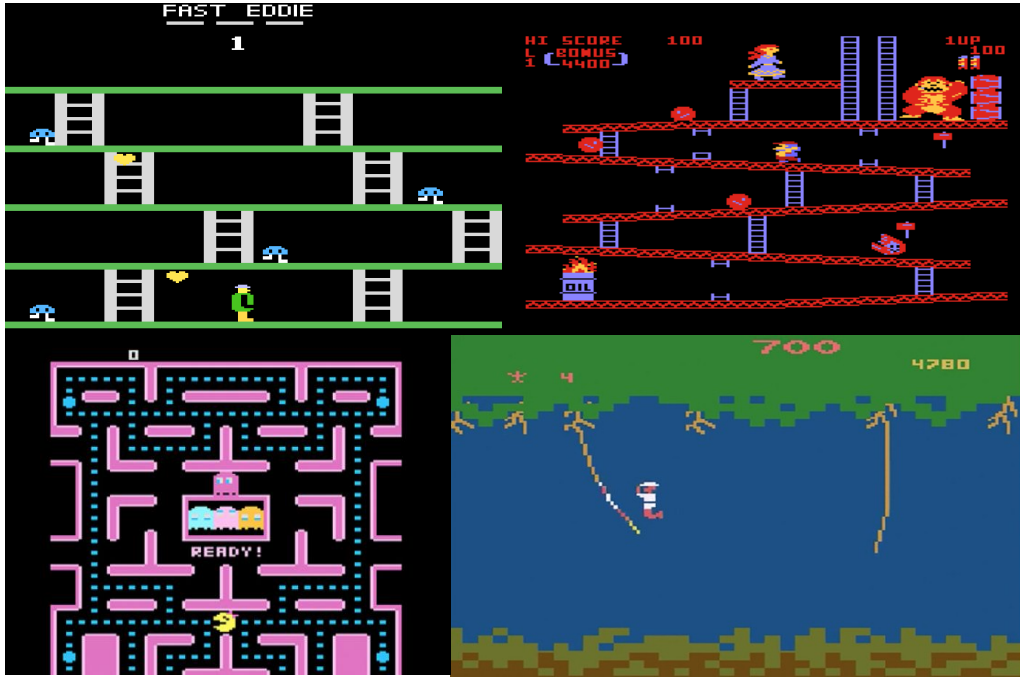
- Those methods proved inefficient for robot RL
- Keys issues: value function estimation based on linear regression is too inaccurate, tuning the stepsize is critical

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000) Policy gradient methods for reinforcement learning with function approximation. In NIPS 12 (pp. 1057–1063).: MIT Press.

General motivations for Deep RL

- Approximation with deep networks provided enough computational power can be very accurate
- Discover the adequate features of the state in a large observation space
- All the processes rely on efficient backpropagation in deep networks
- Available in CPU/GPU libraries: TensorFlow, theano, caffe, Torch... (RProp, RMSProp, Adagrad, Adam...)

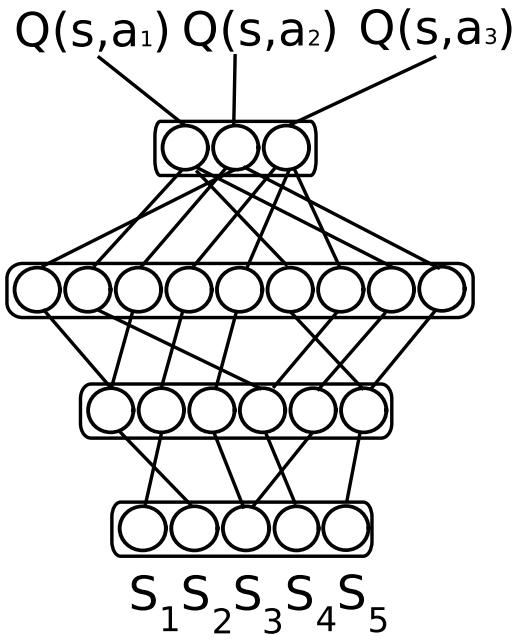
DQN: the breakthrough



- DQN: Atari domain, Nature paper, small discrete actions set
- Learned very different representations with the same tuning

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.

The Q-network in DQN



- Limitation: requires one output neuron per action
- Select action by finding the max (as in Q-learning)
- Q-network parameterized by θ

Learning the Q-function

- Supervised learning: minimize a loss-function, often the squared error w.r.t. the output:

$$L(s,a) = (y^*(s,a) - Q(s,a|\theta))^2 \tag{2}$$

  by backprop on critic weights θ
- For each sample i, the Q-network should minimize the RPE:

$$\delta_i = r_i + \gamma \max_a Q(s_{i+1}, a|\theta) - Q(s_i, a_i|\theta)$$

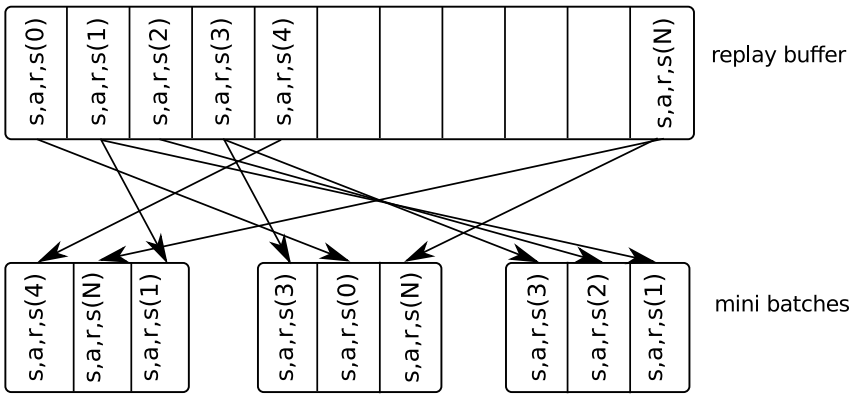
- Thus, given a minibatch of N samples $\{s_i, a_i, r_i, s_{i+1}\}$, compute $y_i = r_i + \gamma \max_a Q(s_{i+1}, a|\theta')$
- So update θ by minimizing the loss function

$$L = 1/N \sum_i (y_i - Q(s_i, a_i|\theta))^2 \tag{3}$$

Trick 1: Stable Target Q-function

- The target $y_i = r_i + \gamma \max_a Q(s_{i+1}, a)|\theta)$ is itself a function of Q
- Thus this is not truly supervised learning, and this is unstable
- Key idea: "periods of supervised learning"
- Compute the loss function from a separate *target network* $Q'(...|\theta')$
- So rather compute $y_i = r_i + \gamma \max_a Q'(s_{i+1}, a|\theta')$
- θ' is updated to θ only each K iterations

Trick 2: Replay buffer shuffling



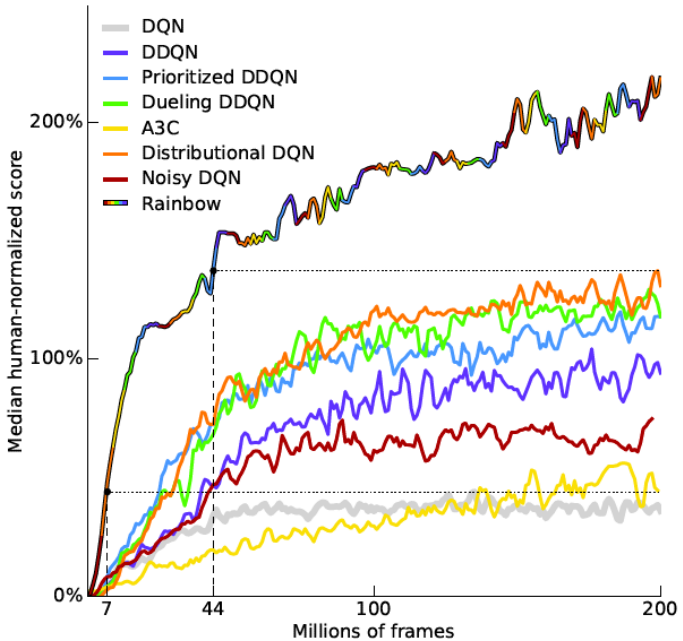
- In most learning algorithms, samples are assumed independently and identically distributed (iid)
- Obviously, this is not the case of behavioral samples (s_i, a_i, r_i, s_{i+1})
- Idea: put the samples into a buffer, and extract them randomly
- Use training minibatches (make profit of GPU when the input is images)
- The replay buffer management policy is an issue

Lin, L.-J. (1992) Self-Improving Reactive Agents based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8(3/4), 293–321

de Bruin, T., Kober, J., Tuyls, K., & Babuška, R. (2015) The importance of experience replay database composition in deep reinforcement learning. In *Deep RL workshop at NIPS 2015*

Zhang, S. & Sutton, R. S. (2017) A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*

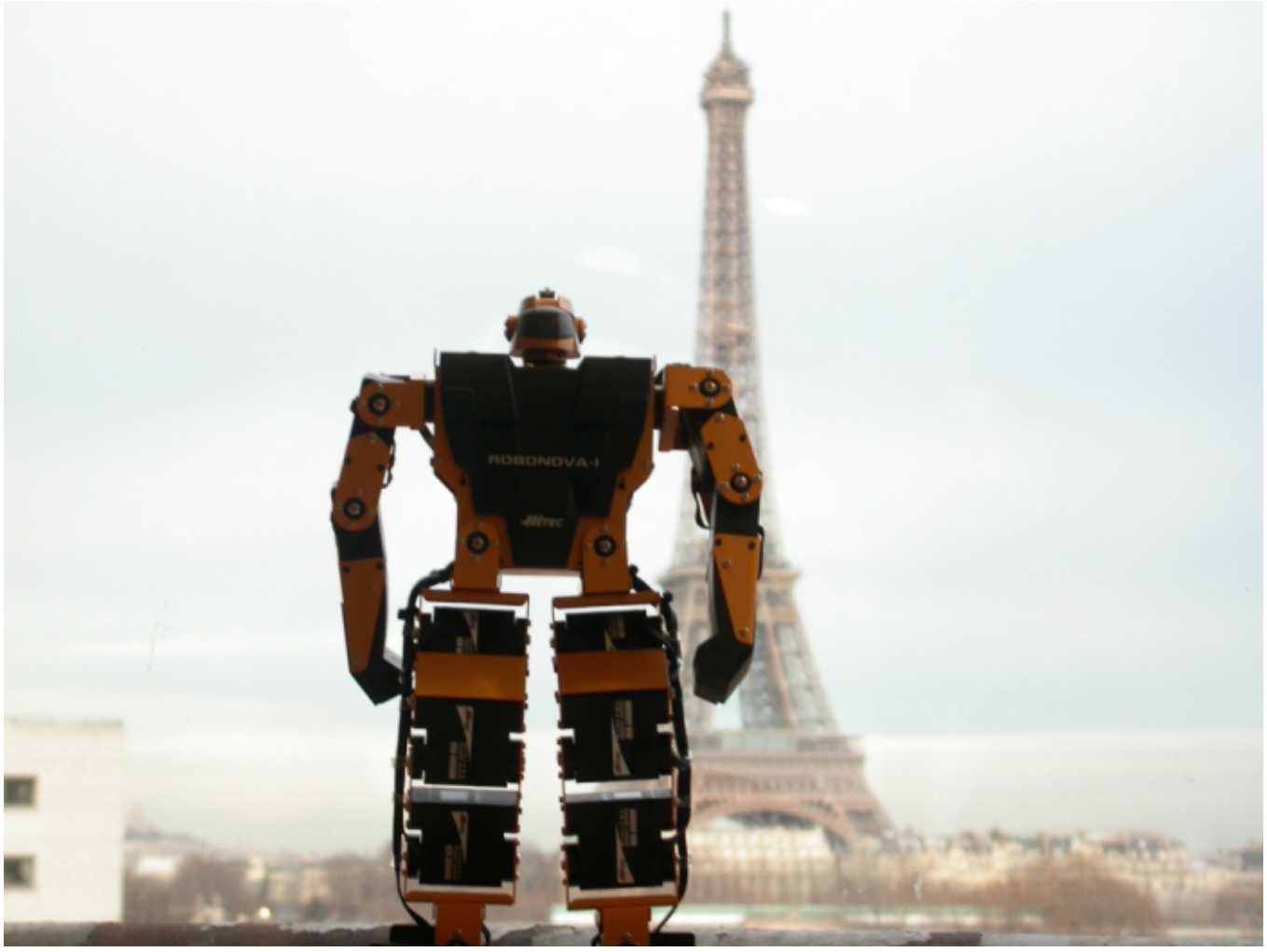
Rainbow



- A3C, distributional DQN and Noisy DQN presented later
- Combining all local improvements

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2017) Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*

Any question?

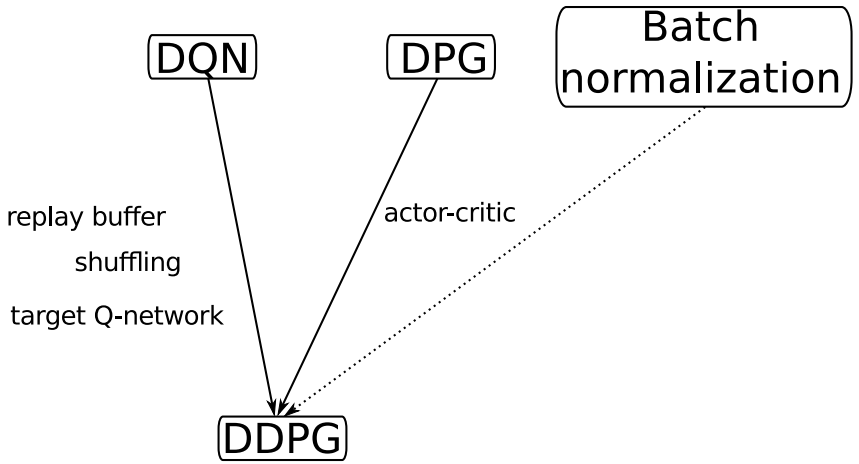


Deep Deterministic Policy Gradient

- Continuous control with deep reinforcement learning
- Works well on "more than 20" (27-32) domains coded with MuJoCo (Todorov) / TORCS
- End-to-end policies (from pixels to control)

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015) Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971 7/9/15*

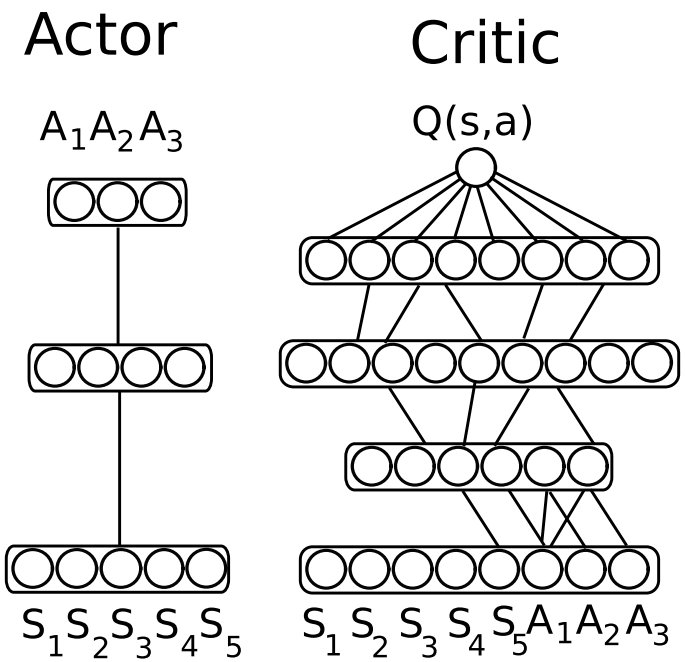
DDPG: ancestors



- Most of the actor-critic theory for continuous problem is for stochastic policies (policy gradient theorem, compatible features, etc.)
- DPG: an efficient gradient computation for deterministic policies, with proof of convergence
- Batch norm: inconclusive studies about importance

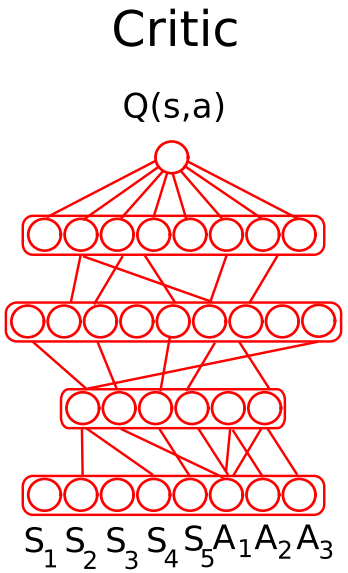
Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014) Deterministic policy gradient algorithms. In *ICML*

General architecture



- Actor parametrized by μ, critic by θ
- All updates based on SGD (as in most deep RL algorithms)

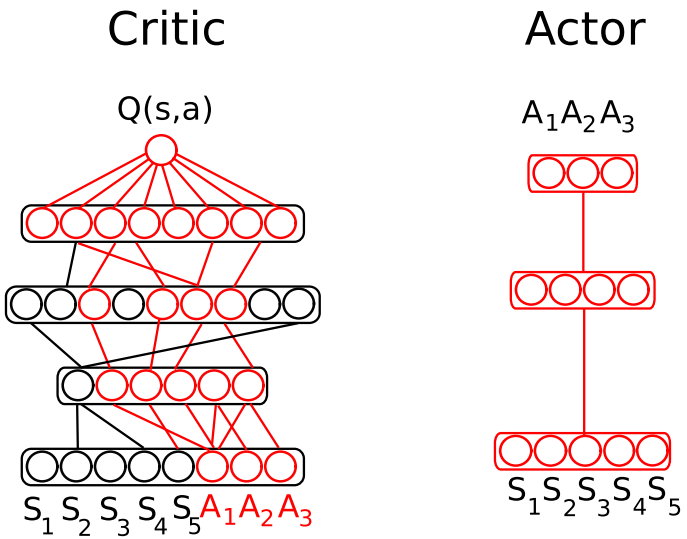
Training the critic



- Same idea as in DQN, but for actor-critic rather than Q-learning
- Minimize the RPE: $\delta_t = r_t + \gamma Q(s_{t+1}, \pi(s_t)|\theta) - Q(s_t, a_t|\theta)$
- Given a minibatch of N samples $\{s_i, a_i, r_i, s_{i+1}\}$ and a target network Q', compute $y_i = r_i + \gamma Q'(s_{i+1}, \pi(s_{i+1})|\theta')$
- And update θ by minimizing the loss function

$$L = 1/N \sum_i (y_i - Q(s_i, a_i|\theta))^2 \quad (4)$$

Training the actor



- Deterministic policy gradient theorem: the true policy gradient is

$$\nabla_\mu \pi(s,a) = \mathbb{E}_{\rho(s)}[\nabla_a Q(s,a|\theta)\nabla_\mu \pi(s|\mu)] \quad (5)$$

- $\nabla_a Q(s,a|\theta)$ is used as error signal to update the actor weights.
- Comes from NFQCA
- $\nabla_a Q(s,a|\theta)$ is a gradient over actions
- $y = f(w.x + b)$ (symmetric roles of weights and inputs)
- Gradient over actions $\sim$ gradient over weights

Hafner, R. & Riedmiller, M. (2011) Reinforcement learning in feedback control. *Machine learning*, 84(1-2), 137–169.

Subtleties

- The actor update rule is

$$\nabla_{\mathbf{w}}\pi(s_i) \approx 1/N \sum_i \nabla_a Q(s, a|\theta)|_{s=s_i, a=\pi(s_i)} \nabla_{\mathbf{w}}\pi(s)|_{s=s_i}$$

- Thus we do not use the action in the samples to update the actor
- Could it be

$$\nabla_{\mathbf{w}}\pi(s_i) \approx 1/N \sum_i \nabla_a Q(s, a|\theta)|_{s=s_i, a=a_i} \nabla_{\mathbf{w}}\pi(s)|_{s=s_i}?$$

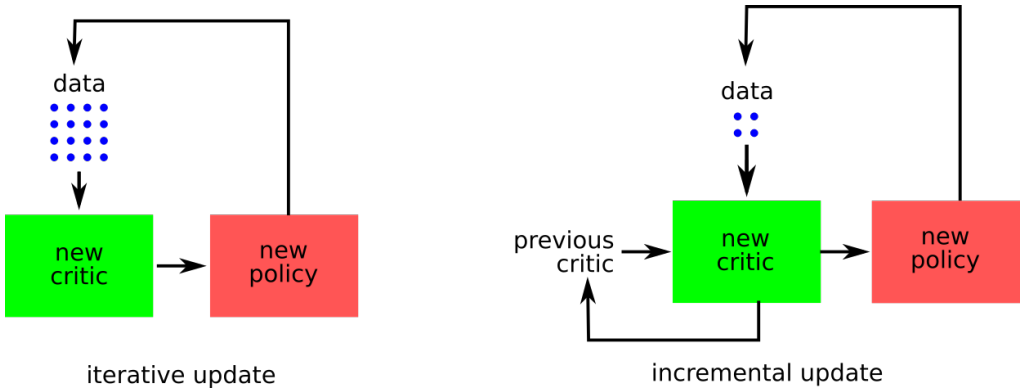
- Work on $\pi(s_i)$ instead of a_i
- Does this make the algorithm on-policy instead of off-policy?
- Does this make a difference?

Stability issue: TD3

- Very recent breakthrough
- Several ways to act against an overestimation bias
- Have two critics, always consider the min, to prevent overestimation
- Less problem knowledge than critic value clipping
- Gives a justification for target actor: slow update of policy is necessary

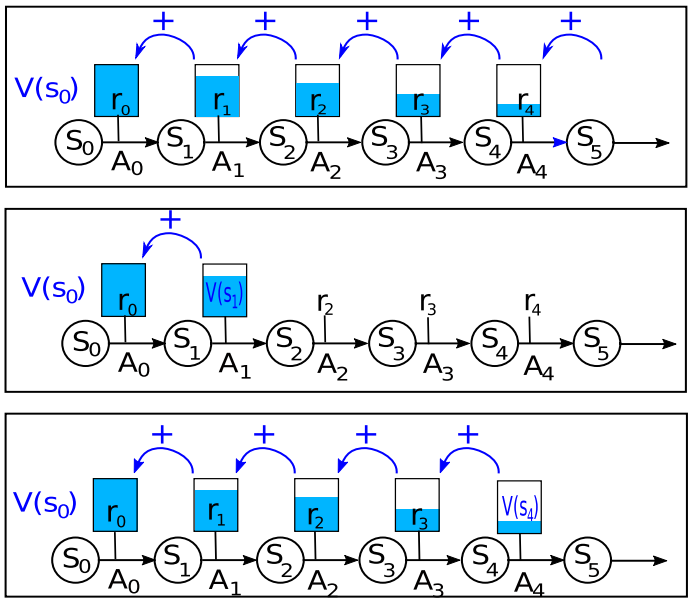
Fujimoto, S., van Hoof, H., & Meger, D. (2018) Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*

Iterative versus Incremental



- Iterative: the critic is recomputed each time (Monte Carlo, e.g. TRPO)
- But still provides a value for each state, thus different from episode-based methods
- Incremental: the critic is updated with new data (TD, e.g. DDPG, or N-step TD, e.g. PPO, D4PG, A3C...)
- Incremental gives more sample reuse

Monte Carlo, One-step TD and N-step TD

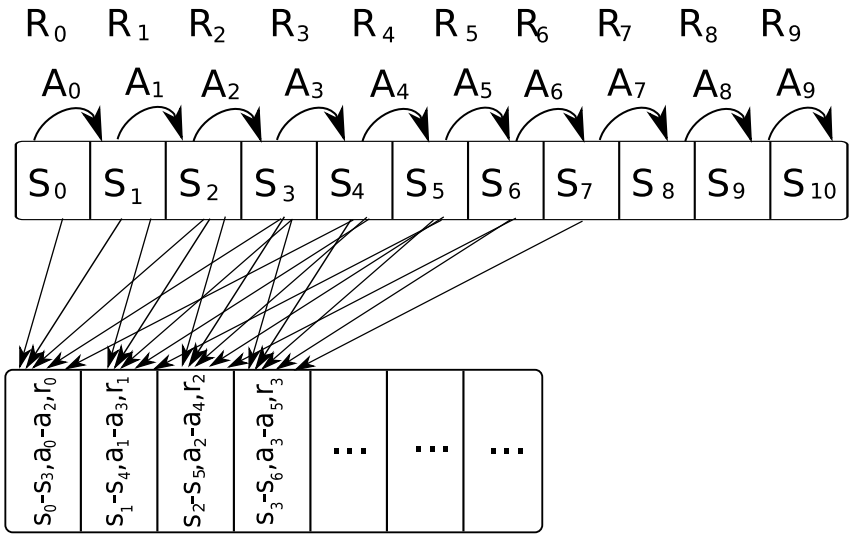


- MC suffers from variance due to exploration (+ stochastic trajectories)
- MC is on-policy → less sample efficient
- One step TD suffers from bias
- N-step TD: tuning N to control the bias variance compromize

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015b) High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*

Sharma, S., Ramesh, S., Ravindran, B., et al. (2017) Learning to mix N-step returns: Generalizing λ-returns for deep reinforcement learning. *arXiv preprint arXiv:1705.07445*

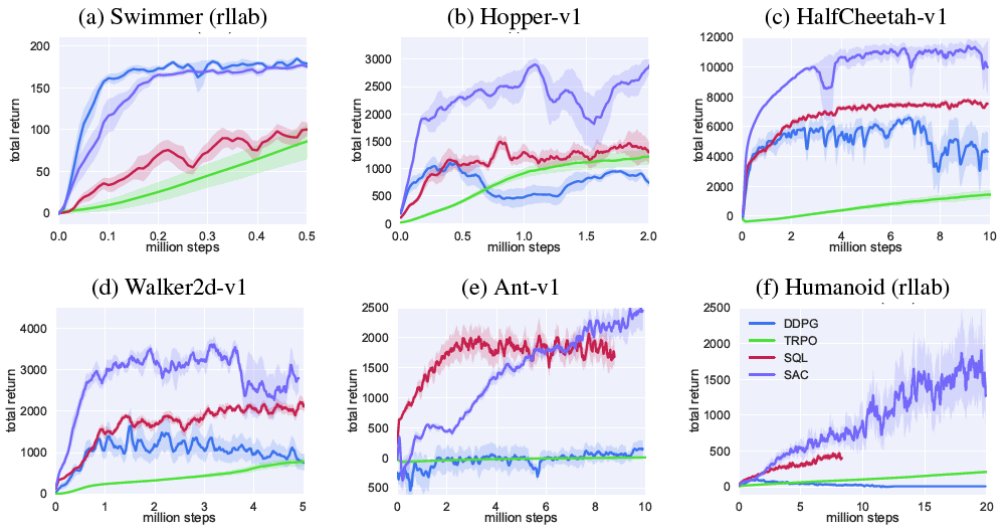
Combining N-step return and replay buffer



- N-step return introduced in A2C, A3C, but without a replay buffer
- Compatibility with shuffling and stochasticity: samples contain N+1 states and N actions
- A bit "less off-policy"?
- Most reliable improvement factor in D4PG, used in PPO and SAC

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016) Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*

Soft Actor-Critic (SAC)



- Comes from PPO, DDPG, A3C...
- Actor-critic with stochastic actor, off-policy
- Adds entropy regularization to favor exploration (follow-up of several papers)
- No annealing of regularization term, effect of entropy not much studied

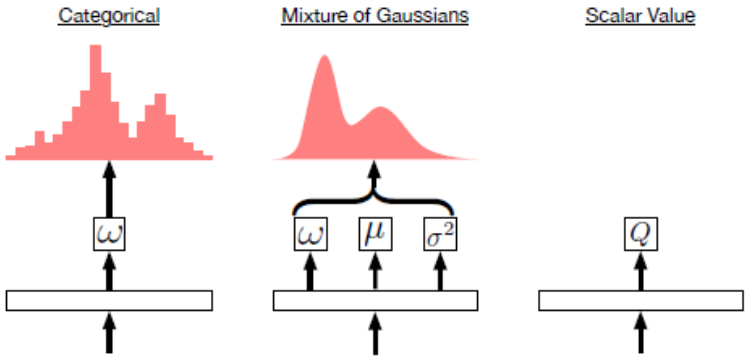
Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018) Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*

ACKTR

- K-FAC: Kronecker Factored Approximated Curvature: efficient estimate of natural gradient
- ACKTR: TRPO with K-FAC natural gradient calculation
- The per-update cost of ACKTR is only 10% to 25% higher than SGD
- Improves sample efficiency (more actor-critic)
- Not much excitement: does the natural gradient really matter?

Wu, Y., Mansimov, E., Liao, S., Grosse, R., & Ba, J. (2017) Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. *arXiv preprint arXiv:1708.05144*

D4PG

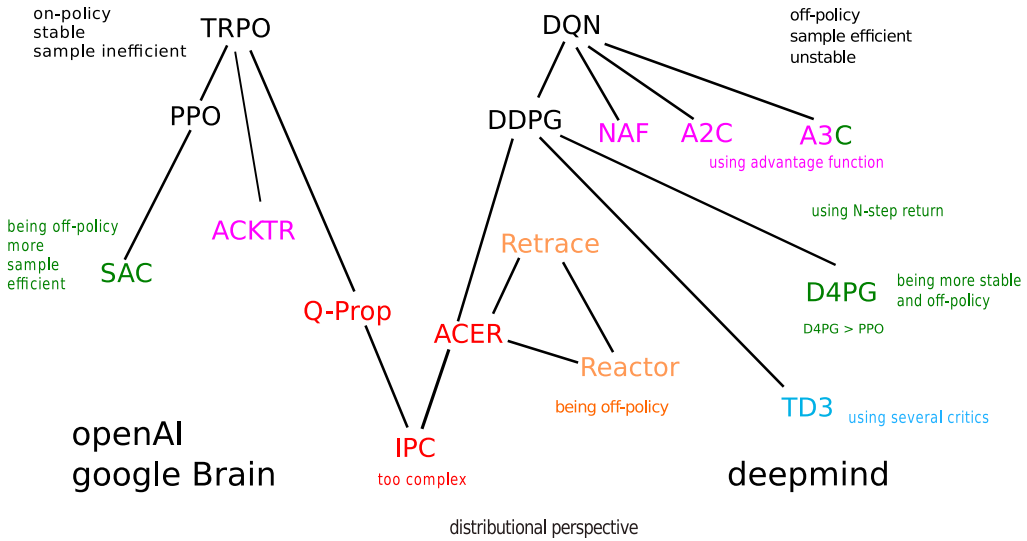


- Distributional policy gradient
- Uses a distribution over returns, and a deterministic policy
- Combined with Prioritized Experience Replay and N-step return
- One of the hotest topics

Barth-maron, G., Hoffman, M., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N., & Lillicrap, T. P. (2018) Distributional policy gradient. In *ICLR* (pp. 1–16).

Bellemare, M. G., Dabney, W., & Munos, R. (2017) A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*

The big picture



- State-of-the-art: SAC, D4PG, Reactor, TD3?...

Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2016) Benchmarking deep reinforcement learning for continuous control. *arXiv preprint arXiv:1604.06778*

Take home messages

- Off-policy Temporal Difference (TD) methods reuse samples but suffer from bias and may run unstable
- MC methods suffer from variance, reuse less samples, but are more stable
- N-step return methods offer a compromize
- Stochastic policies are used for exploration, but combining deterministic policies with dedicated exploration mechanisms may be preferred

Status

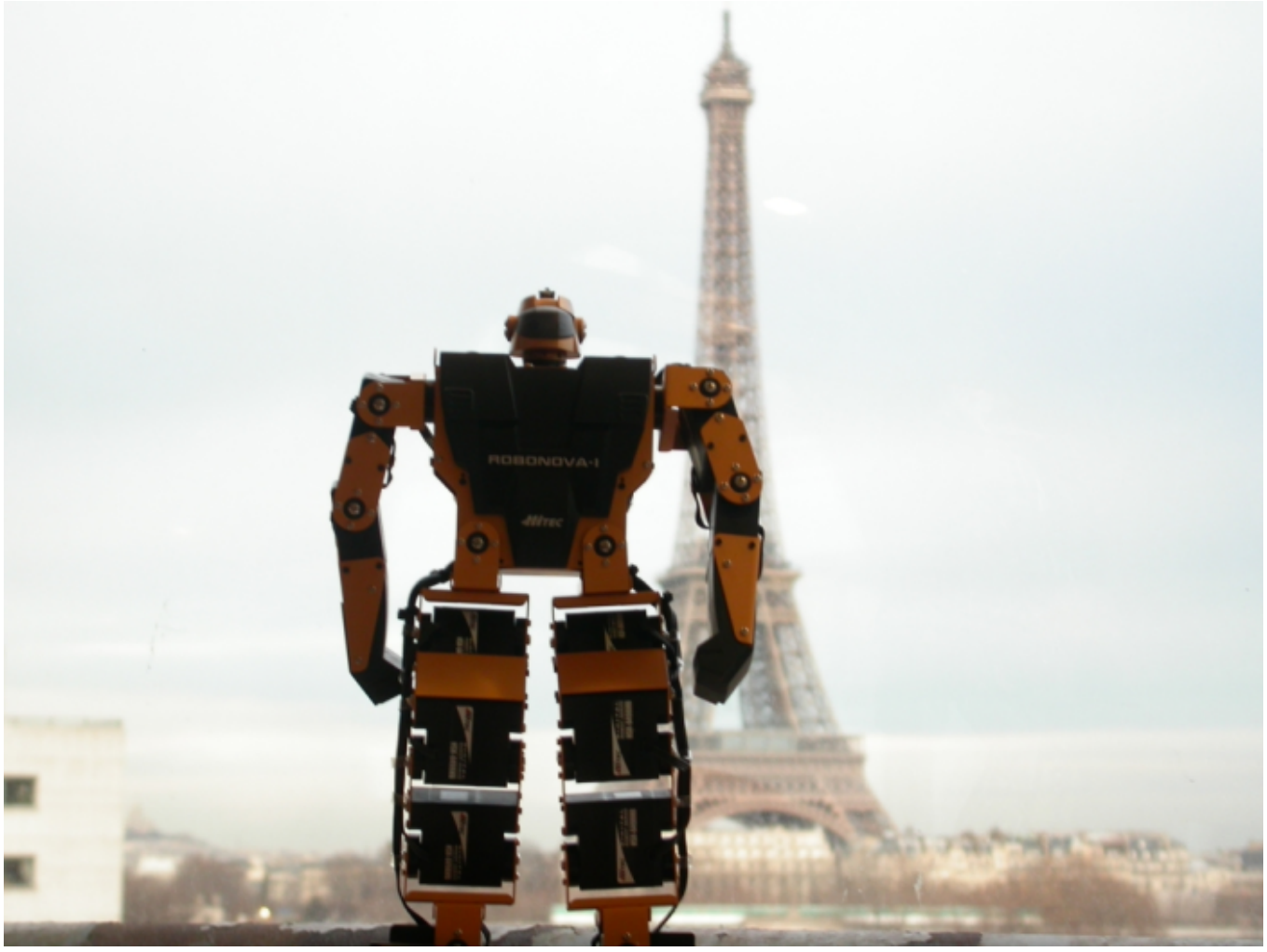
- Big companies are ruling the game, focus on performance
- Deep RL that matters: instabilities, hard to compare, sensitivity to hyper-parameters
- Empirical comparisons based mostly on openAI, mujoco, deepmind control suite
- Lack of controlled experiments (e.g. [Amiranashvili et al., 2018])
- Still fast performance progress, but progress is now more in exploration, multitask learning, curriculum learning, etc.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2017) Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*

General conclusion

- Learning is required for controlling autonomous humanoids
- The Reinforcement Learning framework provides algorithms for autonomous agents.
- It can also help explain neural activity in the brain.
- Such a pluridisciplinary approach can contribute both to a better understanding of the brain and to the design of algorithms for autonomous decision-making
- Deep reinforcement learning is bringing new efficient tools into the story

Any question?




Amiranashvili, A., Dosovitskiy, A., Koltun, V., & Brox, T. (2018).
Td or not td: Analyzing the role of temporal differencing in deep reinforcement learning.
Edité dans *International Conference on Learning Representations (ICLR)*.


Barth-maron, G., Hoffman, M., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N., & Lillicrap, T. P. (2018).
Distributional policy gradient.
Edité dans *ICLR*, pages 1–16.


Bellemare, M. G., Dabney, W., & Munos, R. (2017).
A distributional perspective on reinforcement learning.
arXiv preprint arXiv:1707.06887.


Buffet, O. & Sigaud, O. (2008).
Processus décisionnels de Markov en intelligence artificielle.
Lavoisier.


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


de Bruin, T., Kober, J., Tuyls, K., & Babuška, R. (2015).
The importance of experience replay database composition in deep reinforcement learning.
Edité dans *Deep RL workshop at NIPS 2015*.


Degris, T., Sigaud, O., & Wuillemin, P.-H. (2006).
Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems.
Edité dans *Proceedings of the 23rd International Conference on Machine Learning*, pages 257–264, CMU, Pennsylvania.


Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2016).
Benchmarking deep reinforcement learning for continuous control.
arXiv preprint arXiv:1604.06778.


Fujimoto, S., van Hoof, H., & Meger, D. (2018).

Addressing function approximation error in actor-critic methods.
arXiv preprint arXiv:1802.09477.

Gérard, P., Meyer, J.-A., & Sigaud, O. (2005).

Combining latent learning with dynamic programming in MACS.
European Journal of Operational Research, 160:614–637.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018).

Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.
arXiv preprint arXiv:1801.01290.

Hafner, R. & Riedmiller, M. (2011).

Reinforcement learning in feedback control.
Machine learning, 84(1-2):137–169.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2017).

Deep reinforcement learning that matters.
arXiv preprint arXiv:1709.06560.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2017).
Rainbow: Combining improvements in deep reinforcement learning.
arXiv preprint arXiv:1710.02298.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015).

Continuous control with deep reinforcement learning.
arXiv preprint arXiv:1509.02971.

Lin, L.-J. (1992).

Self-Improving Reactive Agents based on Reinforcement Learning, Planning and Teaching.
Machine Learning, 8(3/4):293–321.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016).

Asynchronous methods for deep reinforcement learning.

arXiv preprint arXiv:1602.01783.

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.

Moore, A. W. & Atkeson, C. (1993).
Prioritized sweeping: Reinforcement learning with less data and less real time.
Machine Learning, 13:103–130.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., & Abbeel, P. (2015).
High-dimensional continuous control using generalized advantage estimation.
arXiv preprint arXiv:1506.02438.

Sharma, S., Ramesh, S., Ravindran, B., et al. (2017).
Learning to mix n-step returns: Generalizing lambda-returns for deep reinforcement learning.
arXiv preprint arXiv:1705.07445.

Sigaud, O. & Buffet, O. (2010).
Markov Decision Processes in Artificial Intelligence.
iSTE - Wiley.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014).
Deterministic policy gradient algorithms.
Edité dans *Proceedings of the 30th International Conference in Machine Learning*.

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.

Sutton, R. S. (1990).
Integrating architectures for learning, planning, and reacting based on approximating dynamic programming.

Edité dans *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, San Mateo, CA. Morgan Kaufmann.

Sutton, R. S. (1991).

DYNA, an integrated architecture for learning, planning and reacting.
SIGART Bulletin, 2:160–163.

Sutton, R. S. & Barto, A. G. (1998).

Reinforcement Learning: An Introduction.
MIT Press.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000).

Policy gradient methods for reinforcement learning with function approximation.
Edité dans *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press.

Watkins, C. J. C. H. (1989).

Learning with Delayed Rewards.
Thèse de doctorat, Psychology Department, University of Cambridge, England.

Wu, Y., Mansimov, E., Liao, S., Grosse, R., & Ba, J. (2017).

Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation.
arXiv preprint arXiv:1708.05144.

Zhang, S. & Sutton, R. S. (2017).

A deeper look at experience replay.
arXiv preprint arXiv:1712.01275.