TÉCNICO LISBOA

INTRODUCTION TO ROBOTICS

Learning and Decision Making

Adapted from 2022 course handouts

Pedro U. Lima

Instituto Superior Técnico/Instituto de Sistemas e Robótica

October 2024

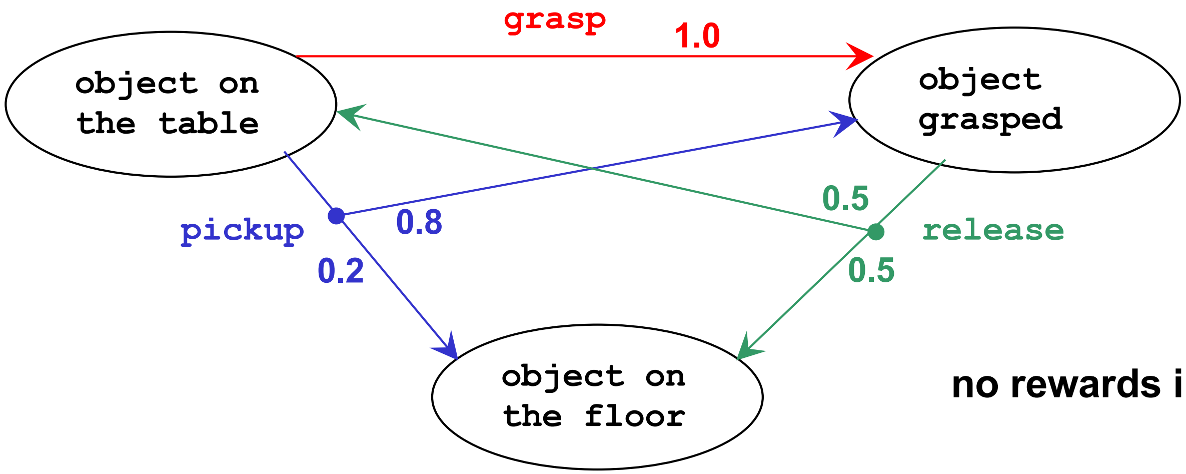
Course handouts

Markov Chains and Markov Decision Processes

A **Markov Chain** (which by definition satisfies the **Markov Property**) with transition probabilities dependent on actions, and with added rewards, is known as **Markov Decision Process (MDP)**

Example: conditional joint probability of state and reward:

$$\Pr\left\{x_{t+1} = x', r_{t+1} = r \middle| x_t, u_t, r_t, x_{t-1}, u_{t-1}, \ldots, r_1, x_0, u_0\right\} = \Pr\left\{x_{t+1} = x', r_{t+1} = r \middle| x_t, u_t\right\}$$



no rewards included in the diagram

Markov Decision Process (MDP)

Given:

- States *x*
- Actions *u*
- Transition probabilities *p(x'|u,x)*
- Reward function *r(x,u)*
- Initial state x_0
- *Discount factor* γ
- Horizon *T*

Goal:

- Find policy $\pi(x)$ that maximizes the expected sum of discounted rewards accumulated over time

$$\max_{\pi} \mathbb{E}\left(\sum_{\tau=0}^{T} \gamma^{\tau} r_{\tau} \mid \pi \right)$$

Markov Decision Process (MDP)

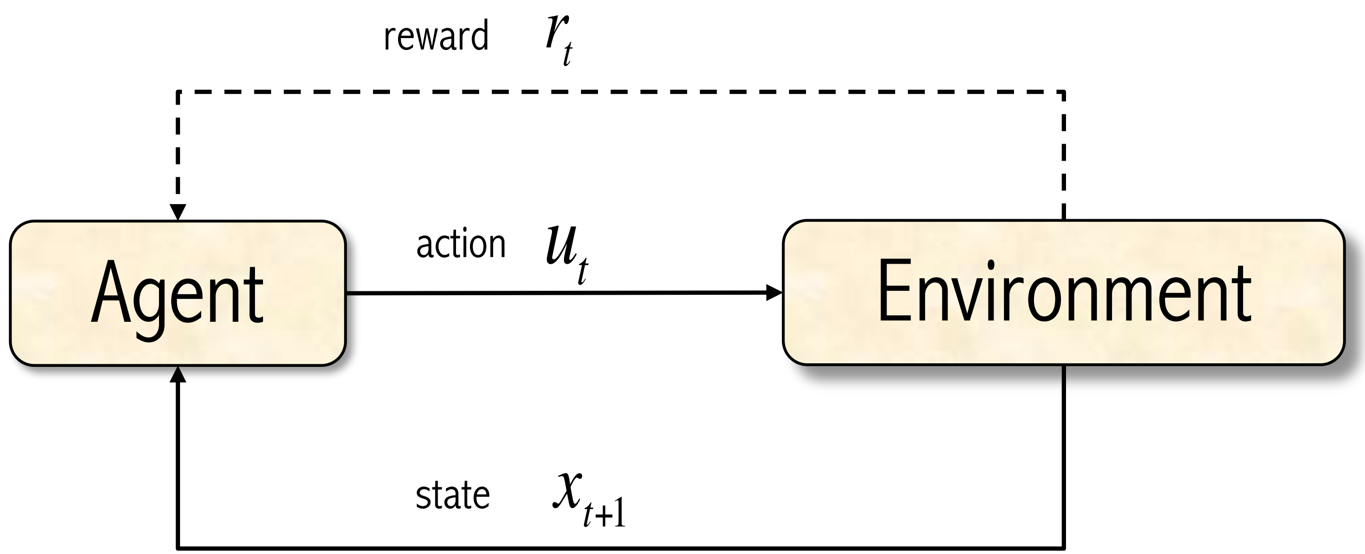
$x_t \in X$

$u_t \in U(x_t)$

$r_{t+1} \in \Re$

Probabilistic Policy function

$$\pi(x_t, u_t) = P(u = u_t | x = x_t), \forall u_t \in U(x_t)$$



Goal: choose the policy that maximizes the

Value function $$V_\pi(x) = \mathbb{E}\left(\sum_{\tau=0}^{T} \gamma^\tau r_\tau \,|\pi, x_0 = x\right)$$

T may go to infinity, as long as $\gamma \neq 1$

States, Actions, and Rewards

- State: complete description of the state of the world.
  - whole chess board information in a chess game.
  - position, velocity, angle, angular velocity of a cart-pole system.
- Action: possible actions in the environment.
  - discrete: left, right, up, down.
  - continuous: robot wheel velocities.
- Reward: $r(x_t, a_t)$ measure how “good” an action for a particular state is.
  - angle of the cart-pole close to zero.
- Discounted cumulative reward:

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

MDP Rewards and Policies

Policy (fully observable case) is a map of states onto actions:

$$\pi : \quad x_t \rightarrow \quad u_t$$

Expected discounted cumulative reward / payoff:

$$R_T = E \quad \left[\sum_{\tau=0}^{T} \gamma^{\tau} r_{t+\tau+1} \right], \quad 0 < \gamma \leq 1$$

– T=0: greedy policy
– T>0: finite horizon case, typically no discount
– T=∞: infinite-horizon case, finite reward
if discount γ < 1

Exact methods

- **Goal:** Find policy $\pi(x)$ that maximizes the expected cumulative reward
- Two methods for finding the optimal policy:
  - value iteration
  - policy iteration

Policies cont'd.

- Expected cumulative payoff of policy π, for a given state x:

$$R_T^\pi(x) = E\left[\sum_{t=0}^{T} \gamma^t r_t \middle|\ x_0 = x, \pi\right]$$

- Optimal value function:

$$V^*(x) = \max_\pi R_T^\pi(x) = \max_\pi E\left[\sum_{t=0}^{T} \gamma^t r_t \middle|\ x_0 = x, \pi\right]$$

- Optimal policy:

$$\pi^* = \arg\max_\pi R_T^\pi(x)$$

Value Iteration

- 1-step optimal value function and policy

$$V_1^*(x) = \max_u \sum_{x'} P(x'|x,u) r(x,u,x')$$

$$\pi_1^*(x) = \arg\max_u \sum_{x'} P(x'|x,u) r(x,u,x')$$

- 2-step optimal value function and policy:

$$V_2^*(x) = \max_u \sum_{x'} P(x'|x,u)(r(x,u,x') + \gamma V_1^*(x'))$$

$$\pi_2^*(x) = \arg\max_u \sum_{x'} P(x'|x,u)(r(x,u,x') + \gamma V_2^*(x'))$$

k-step Value Iteration

- Optimal Value function:

$$V_k^*(x) = \max_u \sum_{x'} P(x'|x,u)(r(x,u,x') + \gamma V_{k-1}^*(x'))$$

- Optimal Policy:

$$\pi_k^*(x) = \arg\max_u \sum_{x'} P(x'|x,u)(r(x,u,x') + \gamma V_{k-1}^*(x'))$$

Infinite Horizon

- Optimal policy, infinite horizon:

$$V_\infty^*(x) = \max_u \sum_{x'} P(x'|x,u)(r(x,u,x') + \gamma V_\infty^*(x'))$$

- Bellman equation

- Fixed point is optimal policy

- Necessary and sufficient condition:

induced policy is optimal iff
value function satisfies the above condition

Value Iteration

Algorithm

For all *x*

$$V_0^*(x) = 0$$

For k = 1, 2, … until convergence

For all *x*

$$V_k^*(x) = \max_u \sum_{x'} P(x'|x,u)(r(x,u,x') + \gamma V_{k-1}^*(x'))$$

Value Iteration

Value iteration converges to the optimal value function, which satisfies the Bellman equation.

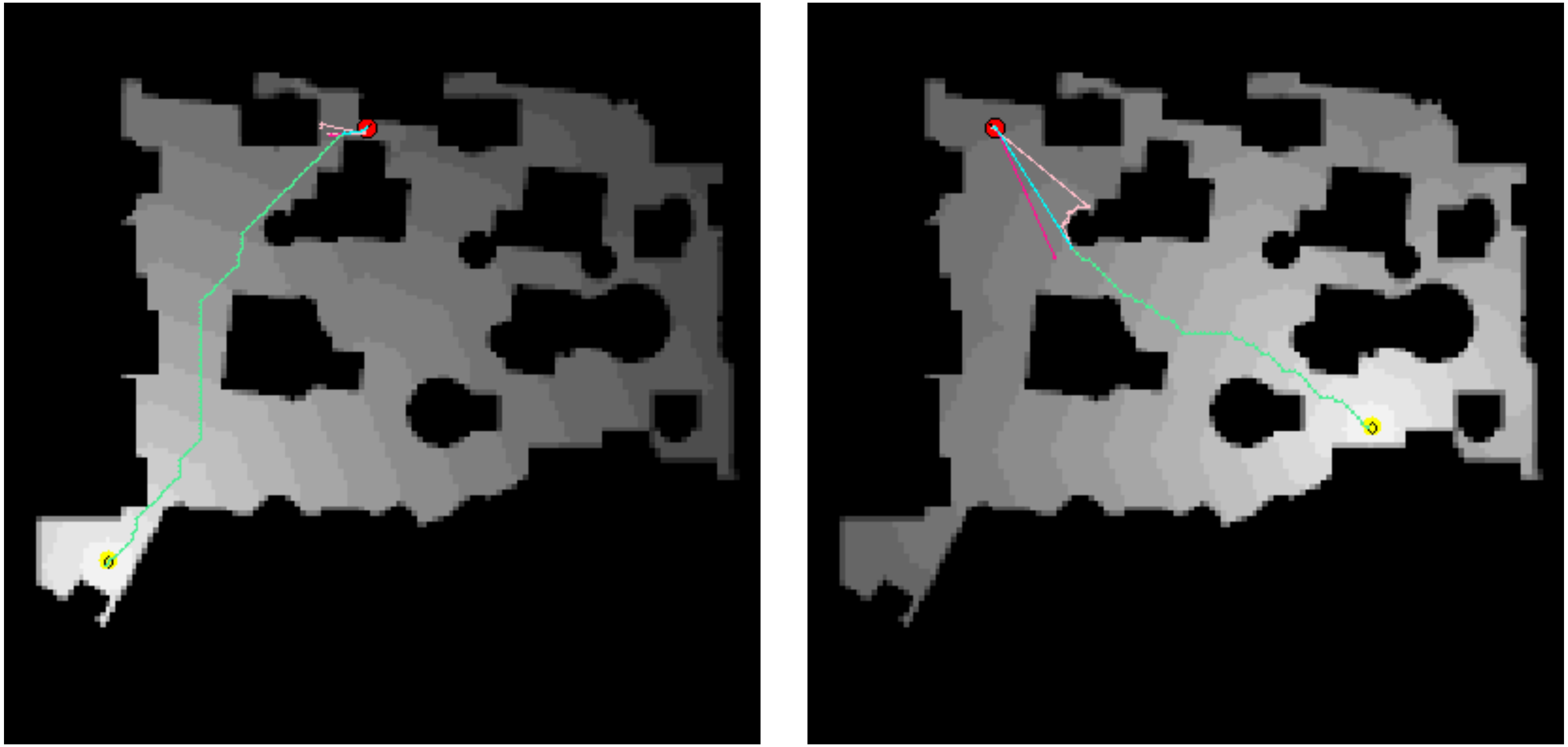
$$V_k^* \rightarrow V^*(x)$$

$$V^*(x) = \max_u \sum_{x'} P(x'|x,u)(r(x,u,x') + \gamma V^*(x'))$$

The optimal policy is given by

$$\pi^*(x) = \arg\max_u \sum_{x'} P(x'|x,u)(r(x,u,x') + \gamma V^*(x'))$$

Value Iteration for Motion Planning



Value Function and Policy Iteration

- Often the optimal policy has been reached long before the value function has converged.
- Policy iteration calculates a new policy based on the current value function and then calculates a new value function based on this policy.
- This process often converges faster to the optimal policy.

Policy Iteration

repeat

repeat (keeping the same policy)

for all x

$$V_{i+1}^{\pi_k}(x) = \sum_{x'} P(x'|x, \pi_k(x))(r(x, \pi_k(x), x') + \gamma V_i^{\pi_k}(x'))$$

policy evaluation

until value function has converged

update policy

for all x

$$\pi_{k+1}(x) = \arg\max_u \sum_{x'} P(x'|x, u)(r(x, u, x') + \gamma V^{\pi_k}(x'))$$

policy improvement

until policy has converged

Reinforcement Learning

Previous (DP) methods to solve MDPs assume full knowledge of $p(x'|u,x)$ and $r(u,x)$

Dynamic Programming (DP)

- To determine V for $|X| = N$, a system of N non-linear equations must be solved.
- Well-established mathematical method.
- A complete model of the environment is required (P *and* R known).
- Often faces the "*curse of dimensionality*" [Bellman, 1957]

Reinforcement Learning

Alternative approaches, if we do not know $p(x'|u,x)$ and $r(u,x)$

Monte Carlo

- Similar to DP, but P and R_s unknown.
- P and R determined from the average of several trial-and-error trials.
- Unappropriate for a step-by-step incremental approximation of V^*.

Temporal Differences

- Knowledge of P *e* R is not required
- Step-by-step incremental approximation of V.
- Mathematical analysis more complex.
- examples: *Q-learning, SARSA, ...*

Reinforcement Learning

Should one learn V*(x)?

- The agent should prefer a state with higher V, because the future cumulative reward will be greater
- But the agent chooses actions, not states
- All fine, then

$$\pi^*(x) = argmax_u\{r(x,u) + E[\gamma V^*(\delta(x,u))\ ]\}$$

Unknown!

Value Functions

state value for policy π:

$$V_{\infty}^{\pi}(x) = \mathrm{E}_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \middle| x_t = x \right\}$$

Expected value of starting in state *x* and following policy π thereafter.

NOTE: value of final state, if any, is always zero.

(state, action) value for policy π:

$$Q_{\infty}^{\pi}(x,u) = \mathrm{E}_{\pi}\left\{ \sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \middle| x_t = x, u_t = u \right\}$$

Expected value of starting in state *x, carrying out action u*, and following policy π thereafter.

Learn Q*(x,u) instead!

Value Functions cont'd

relation between state value and Q function for policy π:

Q is such that its value is the maximum discounted cumulative reward that can be achieved starting from state x and applying action u as the first action

$$Q(x,u) = E[r_{t+1} + \gamma V^*(x_{t+1}) | x_t = x, u_t = u]$$
$$\pi^*(x) = argmax_u \ Q(x,u)$$

$$V^*(x) = max_u Q(x,u)$$
$$\therefore \ Q(x,u) = E[r_{t+1} + \gamma max_{u'} Q(x',u') | x_t = x, u_t = u]$$

Value Functions cont'd

Bellman equation for *V* and *Q*
(discrete action and state spaces, deterministic policy)

$$V^*(x) = \max_u \sum_{x'} P(x'|x,u)(r(x,u,x') + \gamma V^*(x'))$$

$$Q^*(x,u) = \sum_{x'} P(x'|x,u)(r(x,u,x') + \max_{u'} \gamma Q^*(x',u'))$$

Value iteration

$$V_k^*(x) = \max_u \sum_{x'} P(x'|x,u)(r(x,u,x') + \gamma V_{k-1}^*(x'))$$

Q-value iteration

$$Q_{k+1}(x,u) = \sum_{x'} P(x'|x,u)(r(x,u,x') + \max_{u'} \gamma Q_k(x',u'))$$

Value Functions cont'd

Q-value iteration

$$Q_{k+1}(x,u) = \sum_{x'} P(x'|x,u)(r(x,u,x') + \max_{u'} \gamma Q_k(x',u'))$$

Rewritten as an expectation

$$Q_{k+1}(x,u) = E_{x' \sim P(x'|x,u)}[r(x,u,x') + \max_{u'} \gamma Q_k(x',u')]$$

Tabular Q-learning:

- Replace expectation by samples

$$x' \sim P(x'|x,u)$$

- Compute error w.r.t Bellman equation and iterate Q-value

$$Q_{k+1}(x,u) = Q_k(x,u) + \alpha \left[r(x,u,x') + \max_{u'} \gamma Q_k(x',u') - Q_k(x,u) \right]$$

Learning rate

Tabular Q-Learning - Algorithm

```
Algorithm:
     Initialize Q0(x,u) for all x and u
     Initialize current state x
     For k=1,2,... until convergence
          Sample action u
          Execute action u, get r and x'
          Compute Qk+1(x,u)
```

$$Q_{k+1}(x,u) = Q_k(x,u) +$$

$$\alpha \left[r(x,u,x') + \max_{u'} \gamma Q_k(x',u') - Q_k(x,u) \right]$$

Update current state $x \leftarrow x'$

α_n constant allows adaptability to slow environment changes but it does not guarantee convergence – only possible with a temporal decay under given circumstances.

Algorithm Convergence

- All states and actions are visited infinitely often
- Learning rate is such that

$$0 < \alpha_k < 1$$

$$\sum_{k=0}^{\infty} \alpha_k(x,u) = \infty$$

$$\sum_{k=0}^{\infty} \alpha_k(x,u)^2 < \infty$$

Then $\forall x, u \; P[\lim_{k \to \infty} Q_k(x,u) = Q^*(x,u)] = 1$

Action Selection: Exploration *vs* Exploitation

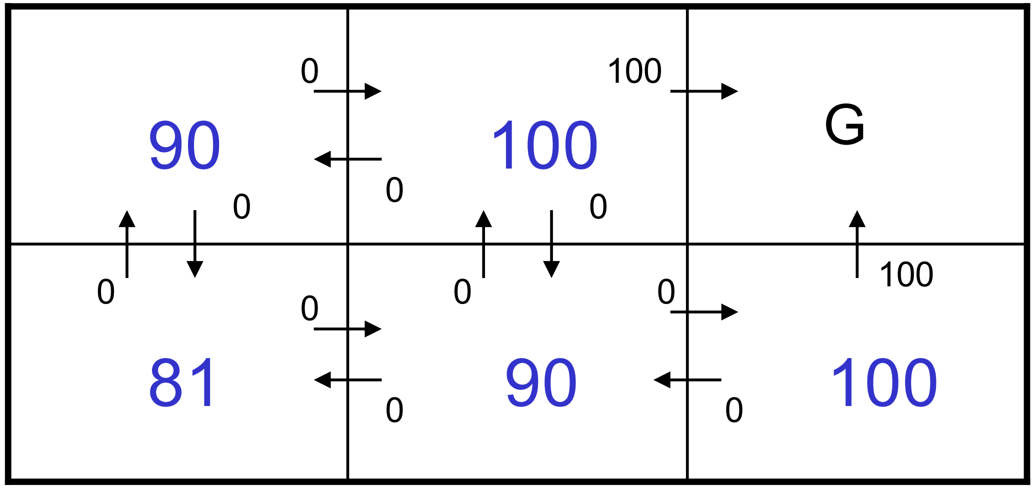
Exploration: less promising actions, which may lead to good results, are tested.
Exploitation: takes advantage of tested actions which are more promising, i.e., which have a larger *Q(x,u)*.

- ε- *greedy:* at each step *n*, picks the best action so far with probability 1-ε , for small ε, but can also pick with probability ε , in an uniformly distributed random fashion, one of the other actions.

- *softmax:* at each step *n*, picks the action to be executed according to a Gibbs or Boltzmann distribution:

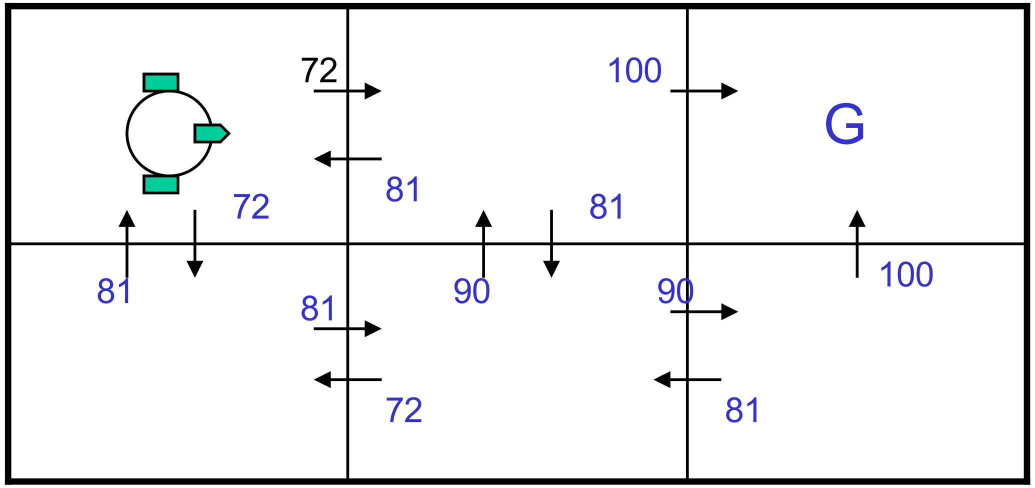
$$\pi_n(x,u) = \frac{e^{Q_n(x,u)/\tau}}{\sum_{u'(x)} e^{Q_n(x,u')/\tau}}$$

Q-Learning – an Example



$r(x,u)$

$V^*(x)$

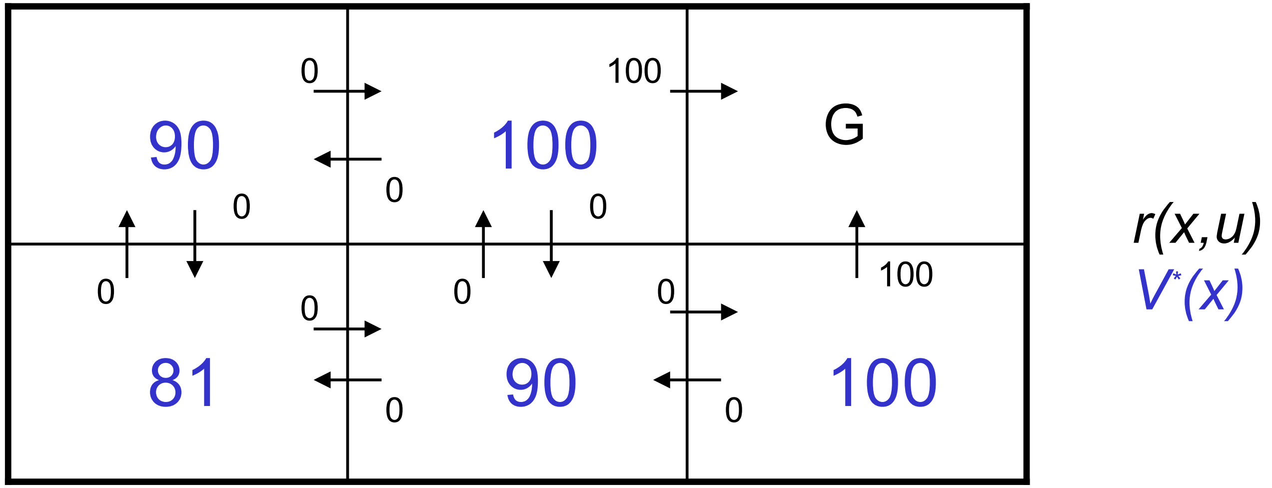


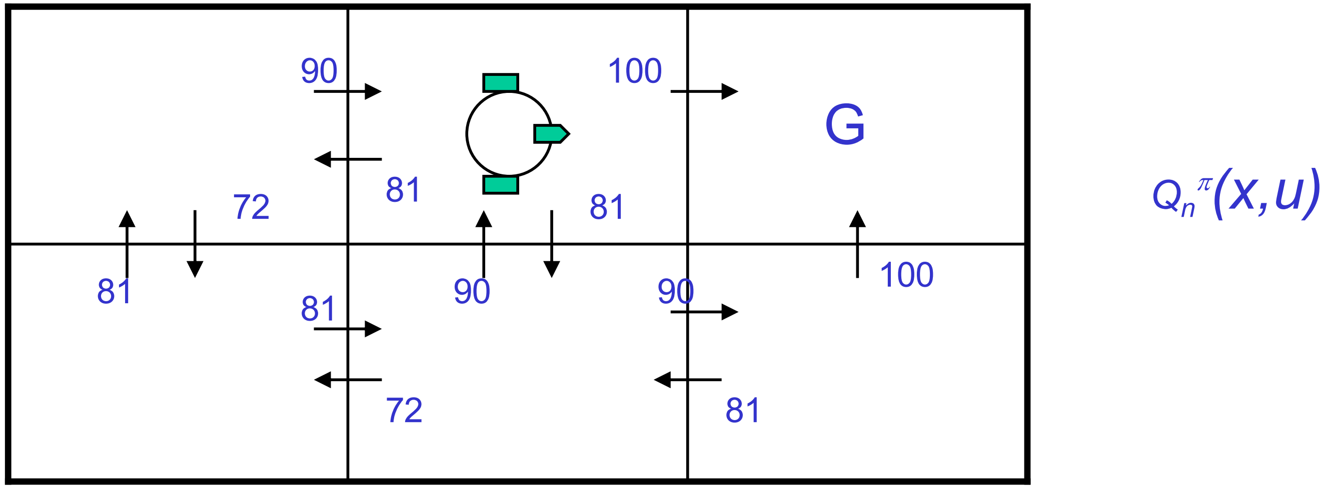
$Q_n^{\pi}(x,u)$

$\alpha_n = 1$

$\gamma = 0.9$

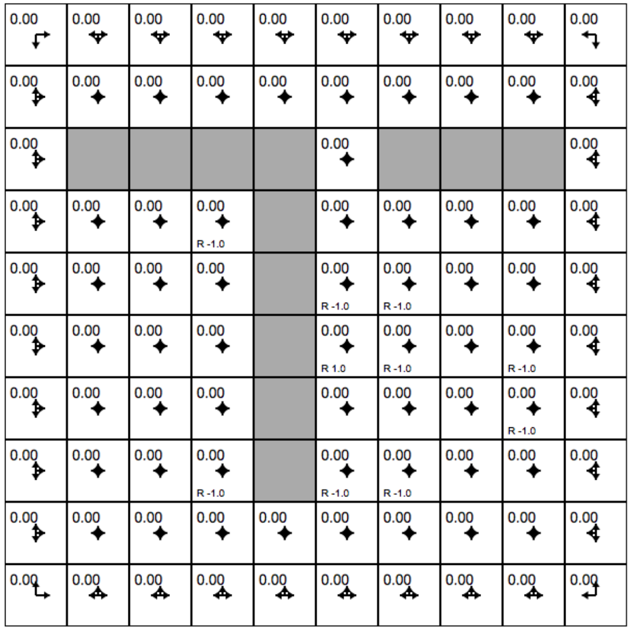
Q-Learning – an Example



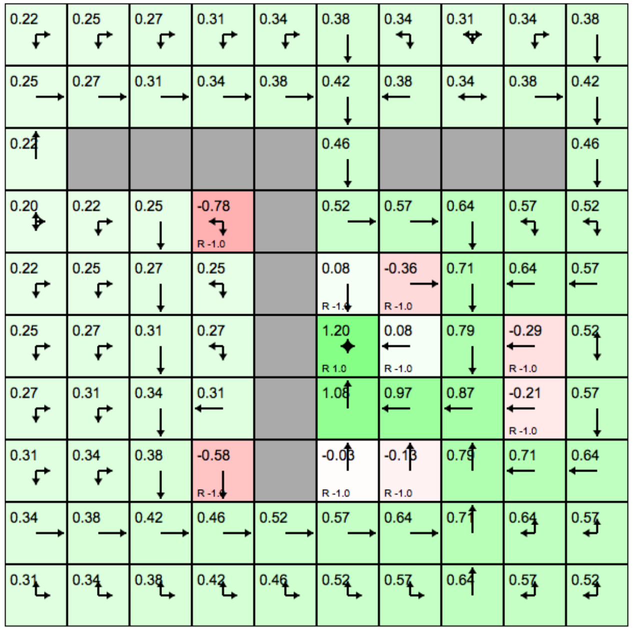


Q-Learning – Grid World

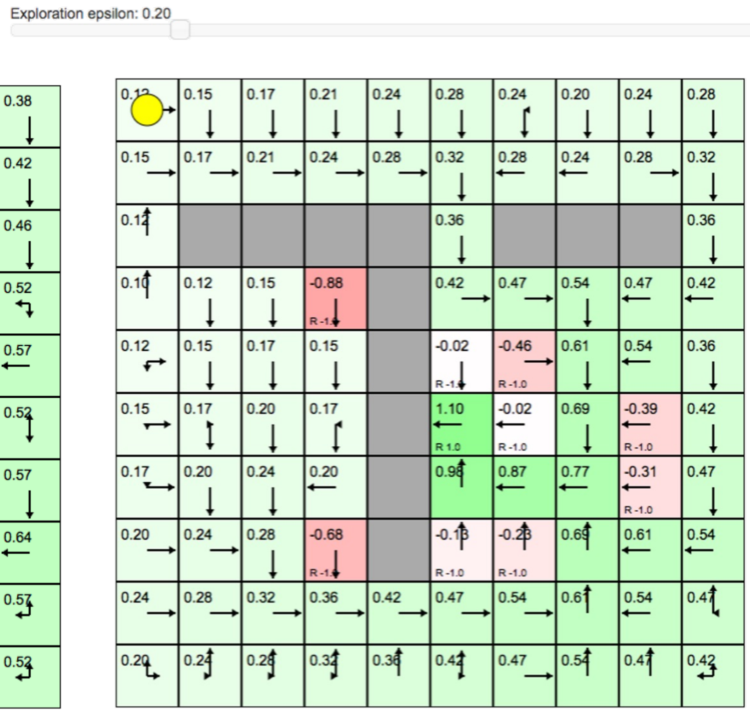
http://cs.stanford.edu/people/karpathy/reinforcejs/index.html



Setup



Dynamic Programming



Q-Learning

Tabular methods do not scale

Discrete environments (number of states)

- Tetris: 10^60
- Atari games: 10^308

Discretized continuous environments

- Inverted pendulum: 10^2
- Hopper: 10^10
- Humanoid: 10^100

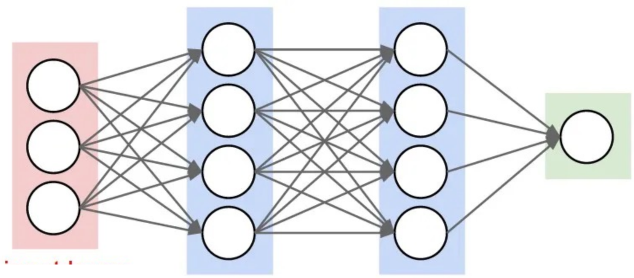
Alternative to tabular representation:

- Parametrized Q function $Q_\theta(x, u)$
  Typically a neural network.

From tabular Q-learning to DQN

Alternative to tabular representation:

- Parametrized Q function $Q_\theta(x,u)$
  Typically a neural network.



- New update rule on the parameters (gradient-based)

$$\theta_{k+1} = \theta_k - \alpha \Delta_\theta \left[\frac{1}{2} (Q_{\theta_k}(x,u) - r(x,u,x') - \gamma \max_{u'} Q_{\theta_k}(x',u'))^2 \right]$$

Alternative to sampling once and updating the Q-function

– Sample and store several actions in a replay memory
– Select a small batch from the replay memory and perform gradient descent using that batch

Many methods out there

- Deep Q-learning (DQN)
  - Learn the Q function (parametrized by θ)
  - Policy π is generated directly from Q
- Policy gradient methods
  - Directly learn π (parametrized by θ)
  - TRPO (Trust Region Policy Optimization)
  - PPO (Proximal Policy Optimization)
- Actor Critic methods
  - Neural nets for the value function and the policy
  - DDPG (Deep Deterministic Policy Gradient)
  - SAC (Soft Actor Critic)

Real Robot RL (Q-learning)

TÉCNICO LISBOA

Real Robot System (MDP + RL)

TÉCNICO LISBOA

References:

- Sutton, Richard S., and Andrew G. Barto. Introduction to reinforcement learning. Vol. 135. Cambridge: MIT Press, 1998.
- Mitchell, Thomas M. "Machine Learning." (1997).
- Sebastian Thrun, Wolfram Burgard and Dieter Fox, *Probabilistic Robotics*, 2005 The MIT Press
- M. T. J. Spaan, “Partially Observable Markov Decision Processes”, in Reinforcement Learning: State of the Art, M. A. Wiering and M. van Otterlo, editors, Springer Verlag, 2012.