Practical Introduction to Reinforcement Learning with Gym in Python

21st IEEE Latin American Robotics Symposium

Material available at www.miguelsolis.info

Dr. Miguel A. Solis
Universidad Andrés Bello
November 12, 2024

LARS 2024

Robotics

Disciplines

- Mobile Robotics
- Bio-inspired Robotics
- Intelligent Robotics
- Cognitive Robotics
- Evolutive Robotics
- Human-Robot Interaction
- Micro-Robotics
- Nano-Robotics
- Tele-Operated Robotics
- Swarm Robotics

Action and Perception

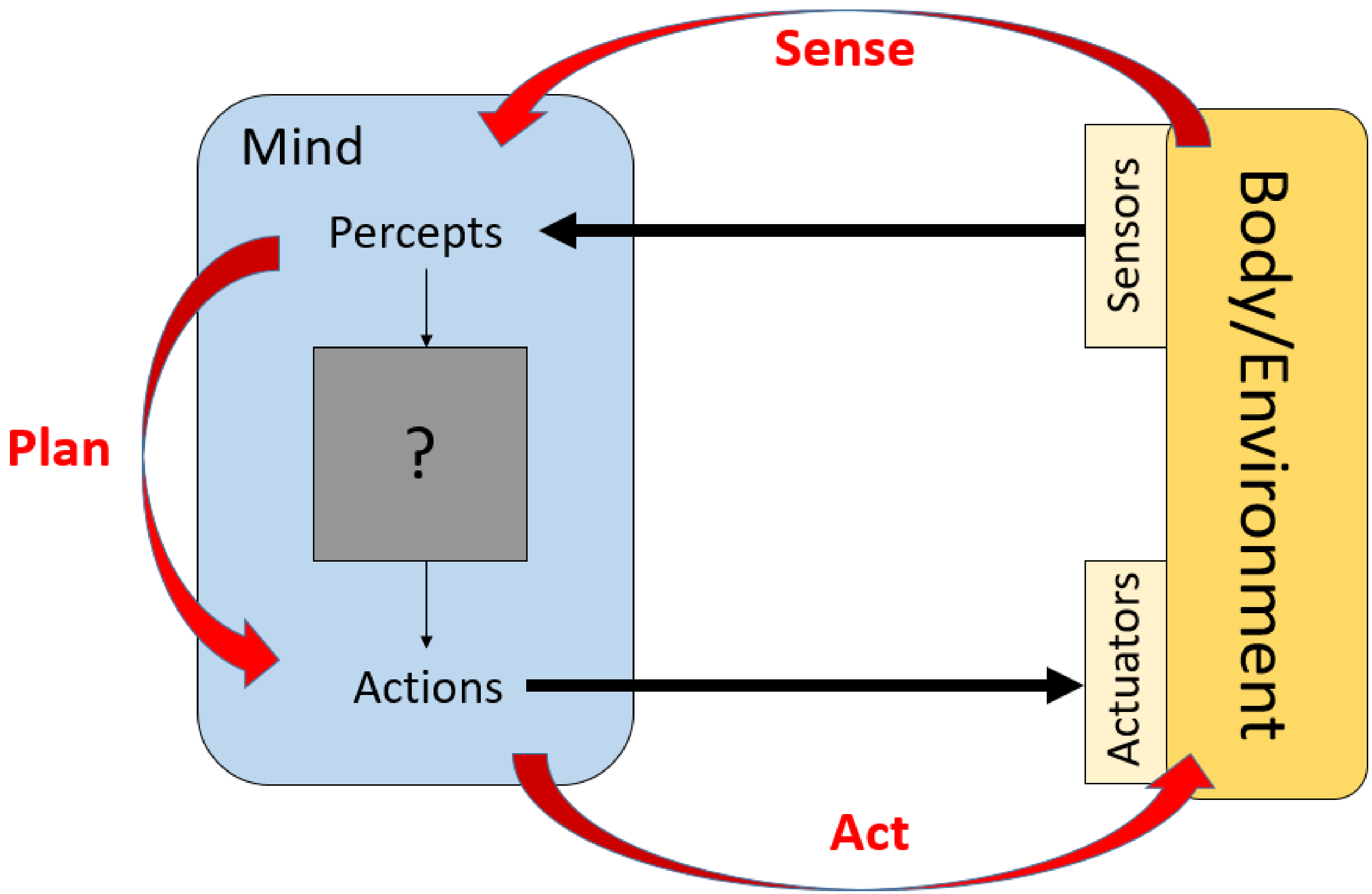


Image extracted from “Robotic Systems”, Kris Hauser, University of Illinois at Urbana-Champaign.

Decision Making

Ways of interacting between agent/environment:

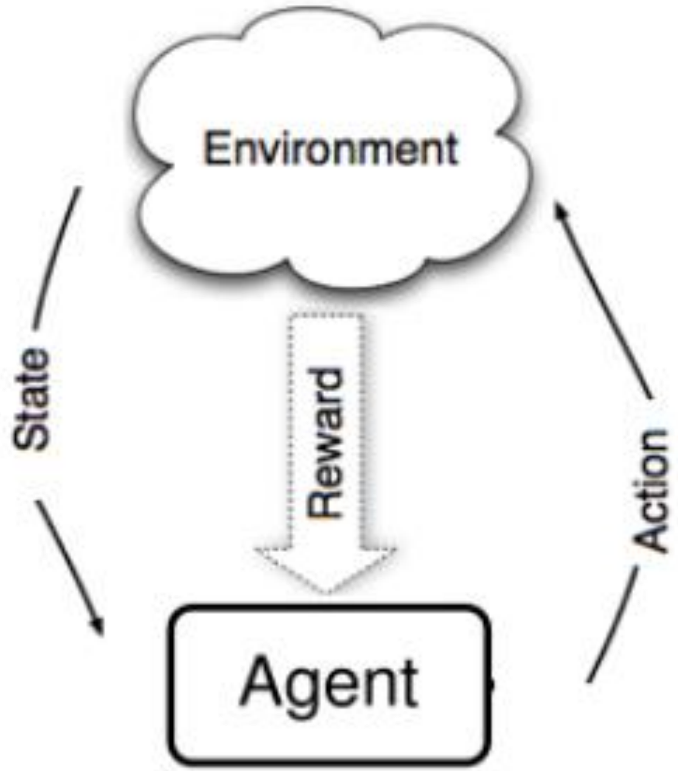
- reactive: simplest way of decisión making through finite states machine (if A when X, then execute B leaving on Y)

Decision Making

Ways of interacting between agent/environment:

- intelligent: behavior shaping through heuristics
- incremental learning: behavior shaping through patterns recognition and prediction of outcomes from the environment

Reinforcement learning



- Learning through interactions with environment.
- Applications

  Sequential decisión problems

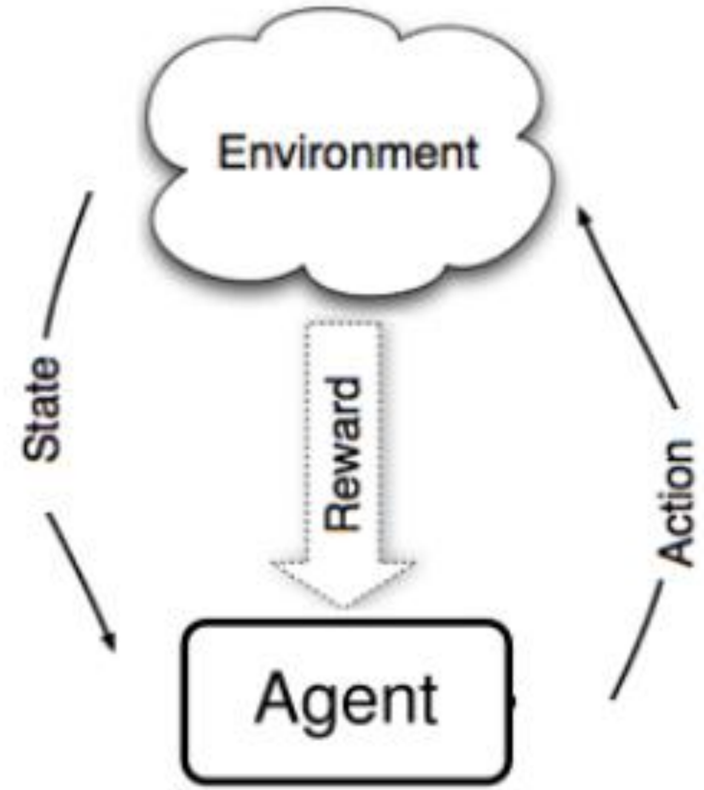
  Adaptive systems

Reinforcement learning and conductism

Edward L. Thorndike – Animal Intelligence: An experimental study of the associate processes in animals (1898).

Animal modifies its behavior according to trial-and-error interactions with the environment.

Reinforcement learning and conductism



- Agent perceives current state, s.
- Agent executes action, a, as output.
- Agent receives a reward, r, as reinforcement signal.

LARS 2024

Markov Process

- A reinforcement learning problema is formally formulated through an MDP.

- MDP: Markov Decision Process.

A given state s_k comes from a (first-order) Markov Process if and only if:

$$Pr\{s_{k+1}|s_k\} = Pr\{s_{k+1}|s_1, ..., s_k\}.$$

Reinforcement learning elements

A reinforcement learning problema, formulated as an MDP is given by the tuple (S, A, T, R) where:

- S: set of states
- A: set of actions
- T: S x A x S → [0, 1] (transition function, unknown)
- R: S x A x S → $\mathbb{R}$ (rewards function)
- π : S → A (policy)

Reinforcement learning elements

State on time step k:

$$s_k \in \mathcal{S}$$

Action executed on time step k:

$$a_k \in \mathcal{A}$$

State-transition function:

$$s_{k+1} = T(s_k, a_k)$$

Rewards

$$r_k = R(s_k, a_k, s_{k+1})$$

- $r_k > 0$
- $r_k = 0$
- $r_k < 0$

Policy $\pi : \mathcal{S} \to \mathcal{A}$

A policy is said to be optimal, if it maximizes the long-term reward

Return

Value function:
$$V^{\pi}(s_k) = r_k + \gamma r_{k+1} + \gamma^2 r_{k+2} + \dots$$
$$= \sum_{i=0}^{\infty} \gamma^i r_{k+i}.$$

Constraints:

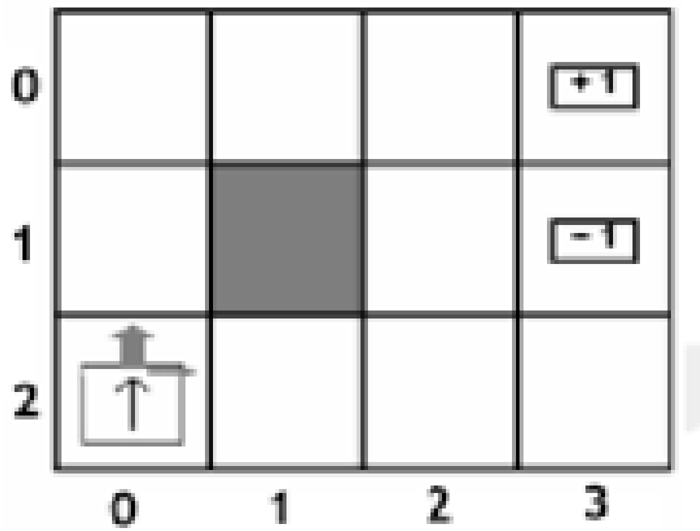
$$0 \le \gamma < 1.$$

r_k bounded

A policy π^* is optimal if value function for that policy is optimal:

$$V^*(s) = V^{\pi^*}(s) \ge V^{\pi}(s) \quad \forall s, \pi$$

Toy example: Grid World



- States: location within the grid.
- Actions: up, left, right, down.
- Rewards: +1 , -1 , -0.1

Classical algorithms

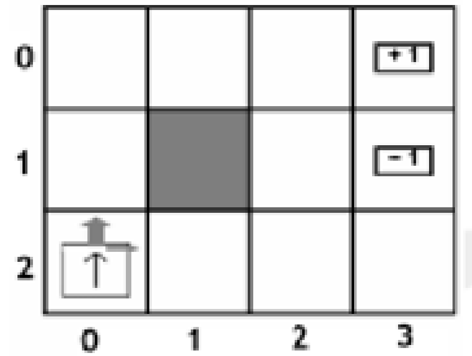
- Value Iteration
- Policy Iteration
- Temporal Difference
    - Q-Learning
    - SARSA

Value Iteration

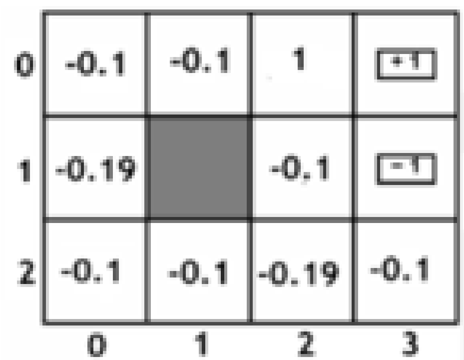
It consists on k iterations, until $V_k(s) - V_{k-1}(s)$ is small enough (according to a given tolerance), with updates given by:

$$V_k(s) = \max_a \sum_{s'} Pr\{s', s, a\} \cdot (R(s', s, a) + \gamma V_{k-1}(s'))$$

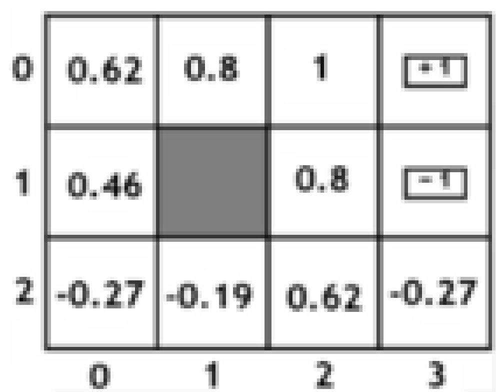
Toy example



- After 1 iteration:



- After 2 iterations:



Toy example

- After 2 iterations:

	0	1	2	3
0	0.62	0.8	1	+1
1	0.46		0.8	-1
2	0.31	0.46	0.62	0.46

- On convergence:

	0	1	2	3
0	0.62	0.8	1	+1
1	0.46		0.8	-1
2	0.31	0.46	0.62	0.46

Q values

Just as V(s) gives some value related to state, Q(s,a) gives some value to taking a certain action on such state.

Optimal policy π^* satisfies:

$$Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a) \quad \forall s, a, \pi$$

Q-learning

Consists on iterating over every (state,action) pair, for given hyperparameters (alpha and gamma).

Update expression:

$$Q(s_k, a_k) \leftarrow (1-\alpha)Q(s_k, a_k) + \alpha \left(r_{k+1} + \gamma \max_a Q(s_{k+1,a}) \right)$$

Assuming $\hat{Q} = Q^*$, then optimal action for every state could be obtained by means of maximizing:

$$\pi^*(s_k) = arg \max_{a_k} Q^*(s_k, a_k)$$

Image extracted from “Introduction to Q-learning with OpenAI Gym”, Gelana Tostaeva, Medium.

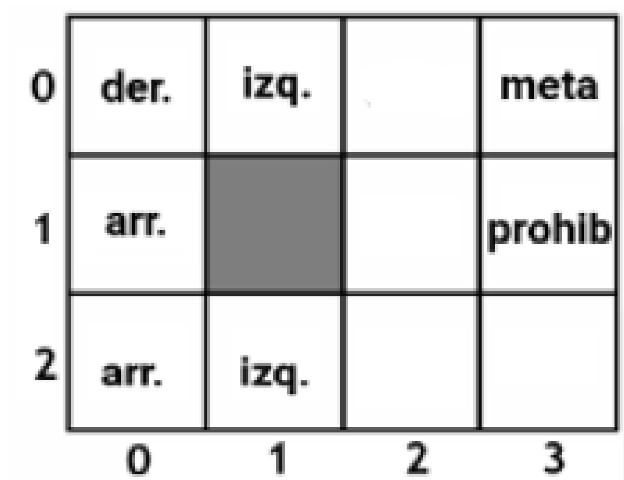
Toy example: Grid World

Universidad Andrés Bello

Algorithm does not iterate over the whole states space, just those visited states.

This (toy) example is episodic (when agent gets to final state, episode finish and a new episode starts with the agent starting again).

Must pay attention to local optimum.



LARS 2024

Toy example: Grid World

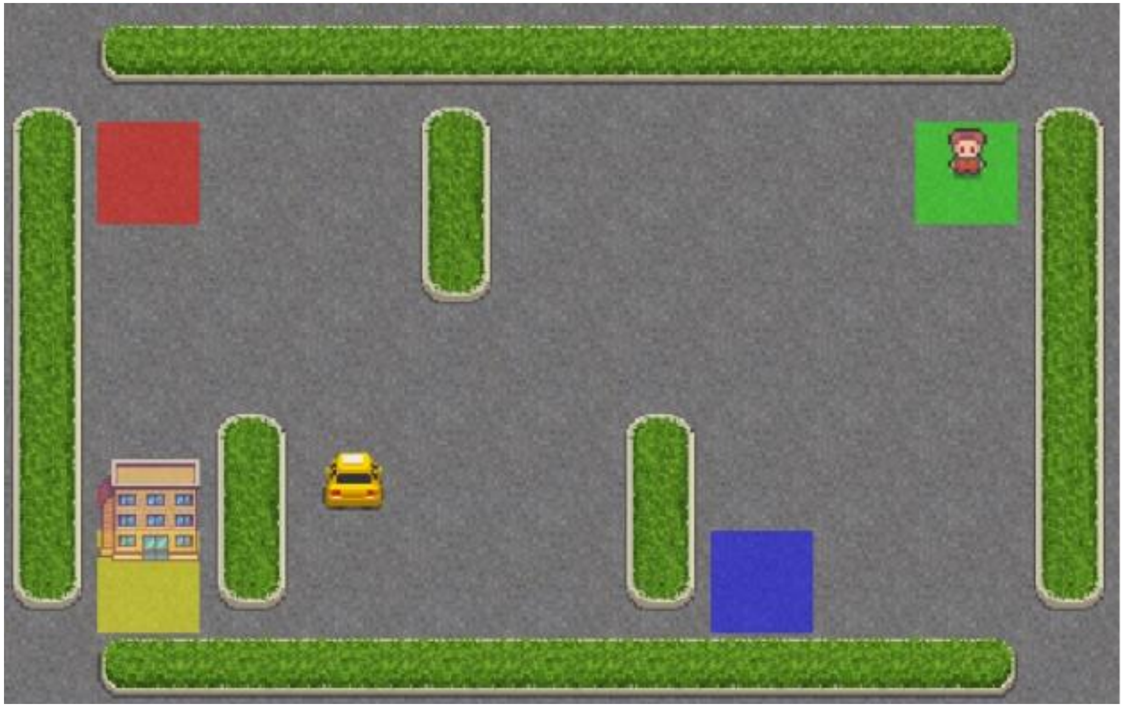
Local optimum stuck:

	0	1	2	3
0	der.	izq.		meta
1	arr.			prohib
2	arr.	izq.		

Including exploration

	0	1	2	3
0	der.	der.	der.	meta
1	arr.			prohib
2	arr.	izq.		

Taxi problem (Gym)



Actions

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

0: south
1: north
2: east (right)
3: west (left)
4: pick up passenger
5: leave passenger

States and Actions Space

Actions: 6

States: 500 (passenger_location, taxi_location, destination)

4 possible destinations
passenger_location: 4 possible locations for origin, or same location as the taxi
taxi_location: 25 possible locations according to the map

Gym verification:

env.action_space
env.observation_space

Rewards

- Passenger left on correct location: 20 points

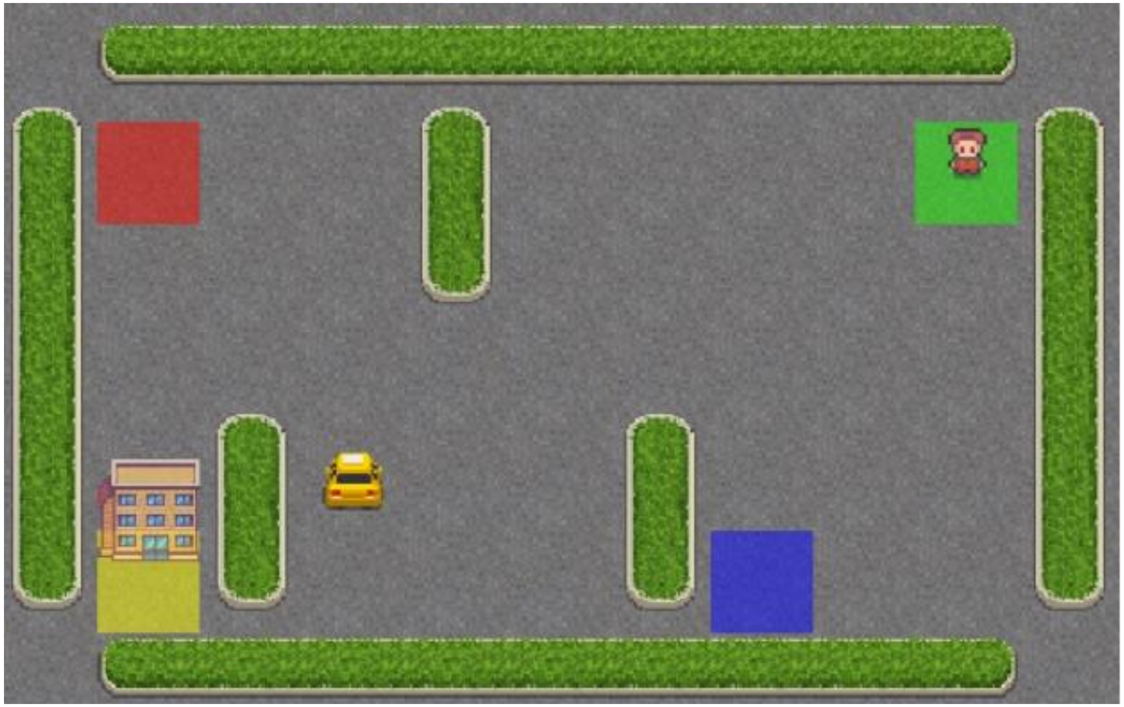
- Discount of 1 point anytime taxi moves with passenger without arriving to destination

- Discount of 10 points for leaving passenger on illegal location

Python Notebook

Universidad Andrés Bello

12112024_Tutorial_MSolis.ipynb

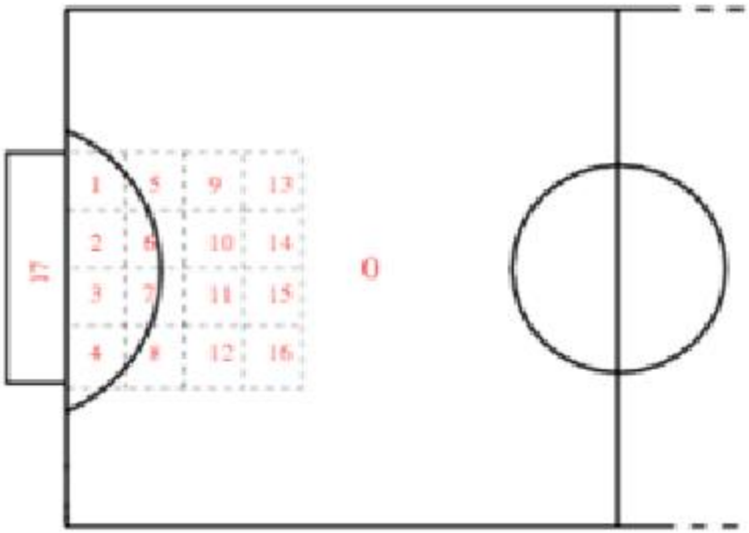


Open with any Python Notebook Interpreter, or colab.research.google.com

LARS 2024

More examples

- G.A. Ahumada, C.J. Nettle and M.A. Solis, ‘Accelerating Q-learning through Kalman Filter Estimations applied in a RoboCup SSL Simulation’, **Proceedings** of the 10th IEEE Latin American Robotics Symposium, 2013.



More examples

Universidad Andrés Bello

Deffensive strategy generation

State is composed by

- dist(Keeper, ball) , dist(Taker, ball)
- Dist(Keeper_Team_A, Keeper_Team_B)
- Dist(Keeper_Team_A, Taker_Team_B)
- Angle(Keeper_Team_A, Taker_Team_B)

Ollino, F., Solis, M. A., & Allende, H. (2018). Batch reinforcement learning on a RoboCup Small Size League keepaway strategy learning problem. In 4th Congress on Robotics and Neuroscience, CRoNe 2018. CEUR-WS.

LARS 2024

Important challenges

Rewards design

Delay on actions execution

Tabular representation

Some current topics

- affordances

(Object, Action, Effect)

Given an object and a certain action, what effect does it have?

Given an object and a desired effect, what is the required action?

- continual reinforcement learning (open-ended)

Final questions?

(material available at www.miguelsolis.info)