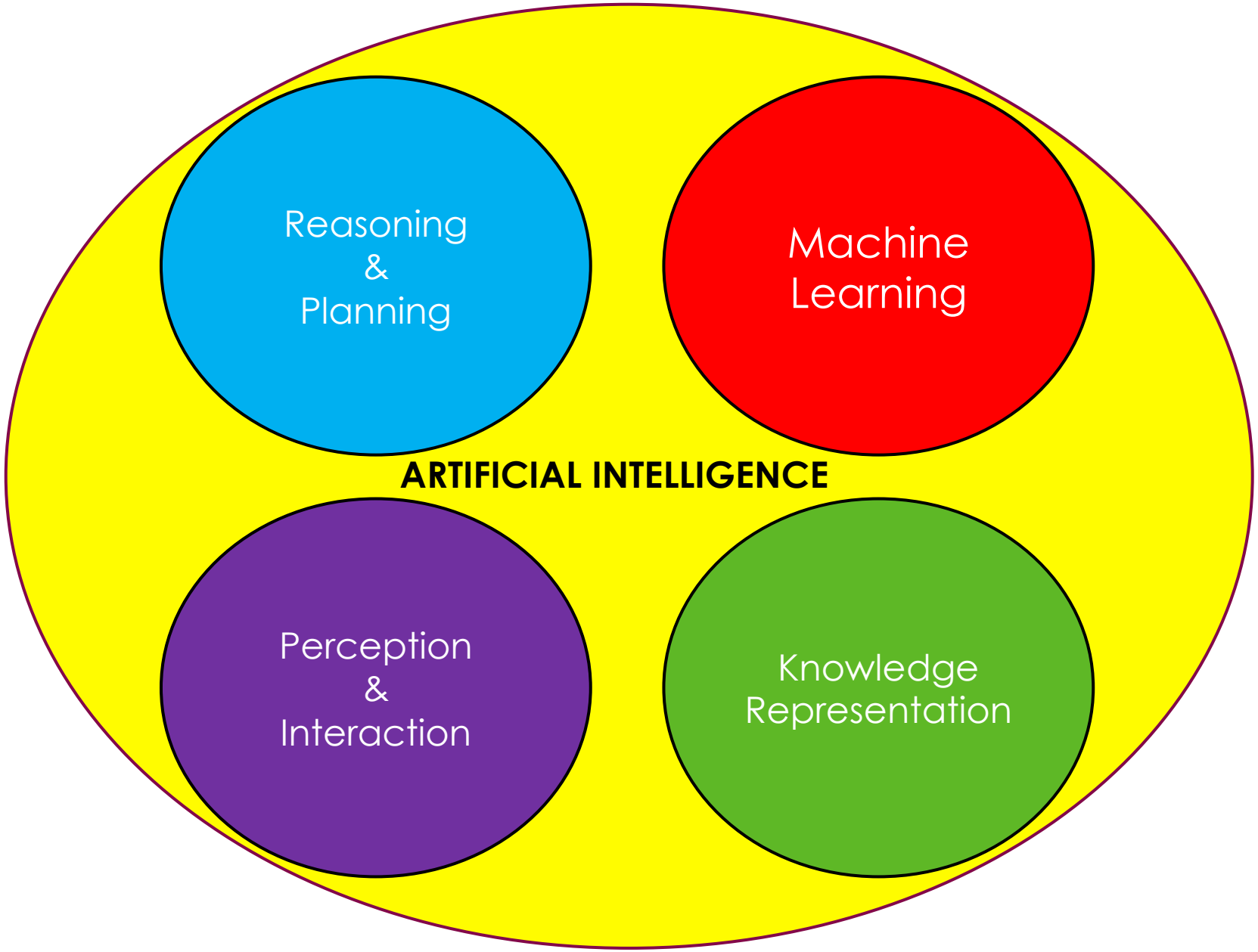
Aprendizado por Reforço: conceitos, aplicações e desafios

ANNA HELENA REALI COSTA
UNIVERSIDADE DE SÃO PAULO
anna.reali@usp.br

Parte 1


Reasoning
&
Planning
Machine
Learning
ARTIFICIAL INTELLIGENCE
Perception
&
Interaction
Knowledge
Representation

Learning

- Learning modifies the agent's decision mechanisms to improve performance
- Learning is essential for unknown environments
  - i.e., when designer lacks omniscience
- Learning is useful as a system construction method
  - i.e., expose the agent to reality

Learning Agent

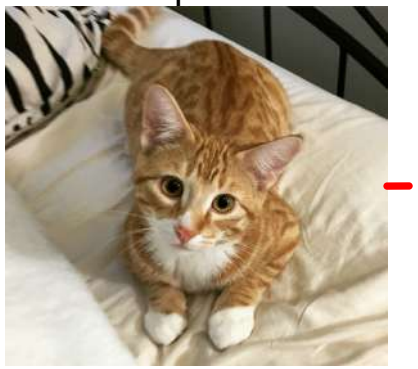
- Design of a learning agent depends on what feedback is available
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning

Supervised Learning

- **Supervised learning** is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.

Training Data

input



desired output

CAT

Execution

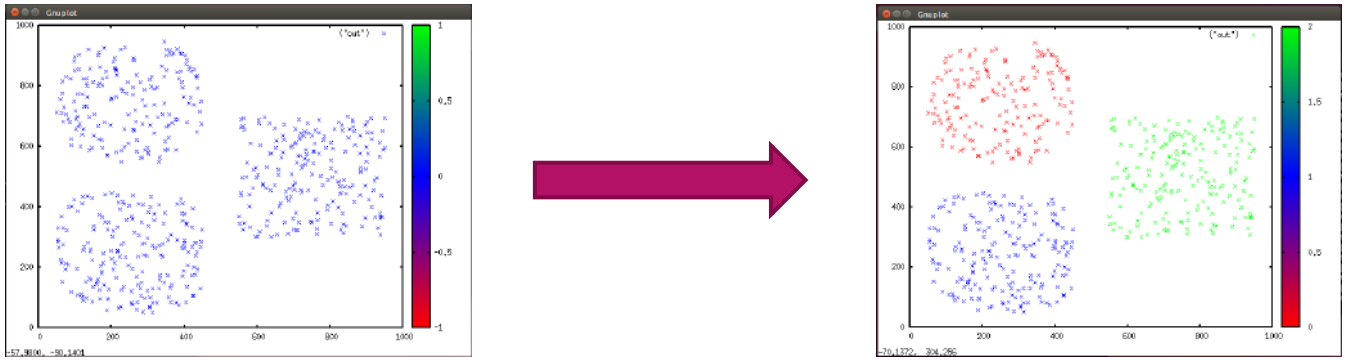
system

CAT

https://en.wikipedia.org/wiki/Supervised_learning

Unsupervised Learning

- **Unsupervised learning** is a type of machine learning that looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision.



https://en.wikipedia.org/wiki/Unsupervised_learning

Reinforcement Learning

- RL addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to get as much reward as it can over the long run.
- Applied to Sequential Decision Problems

Reinforcement Learning

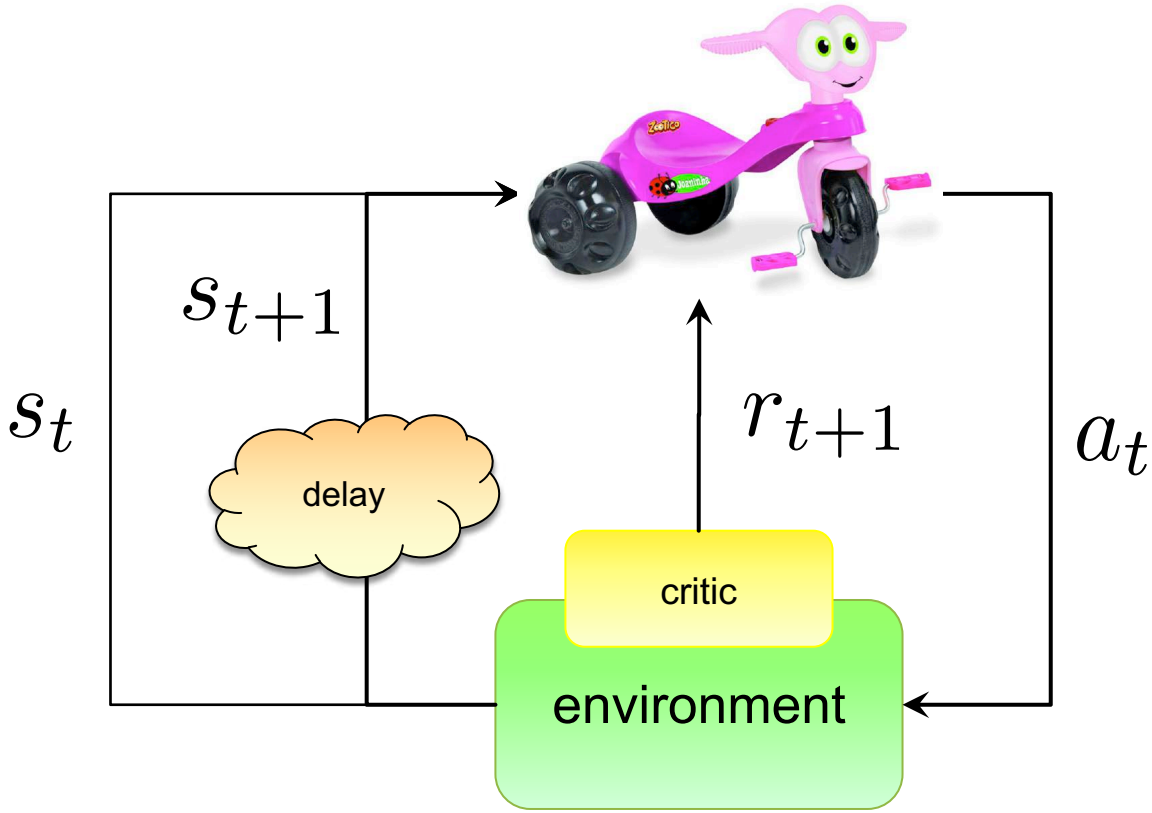
- Feedback is delayed, occasional
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives
- Trial and error learning (via experiences)

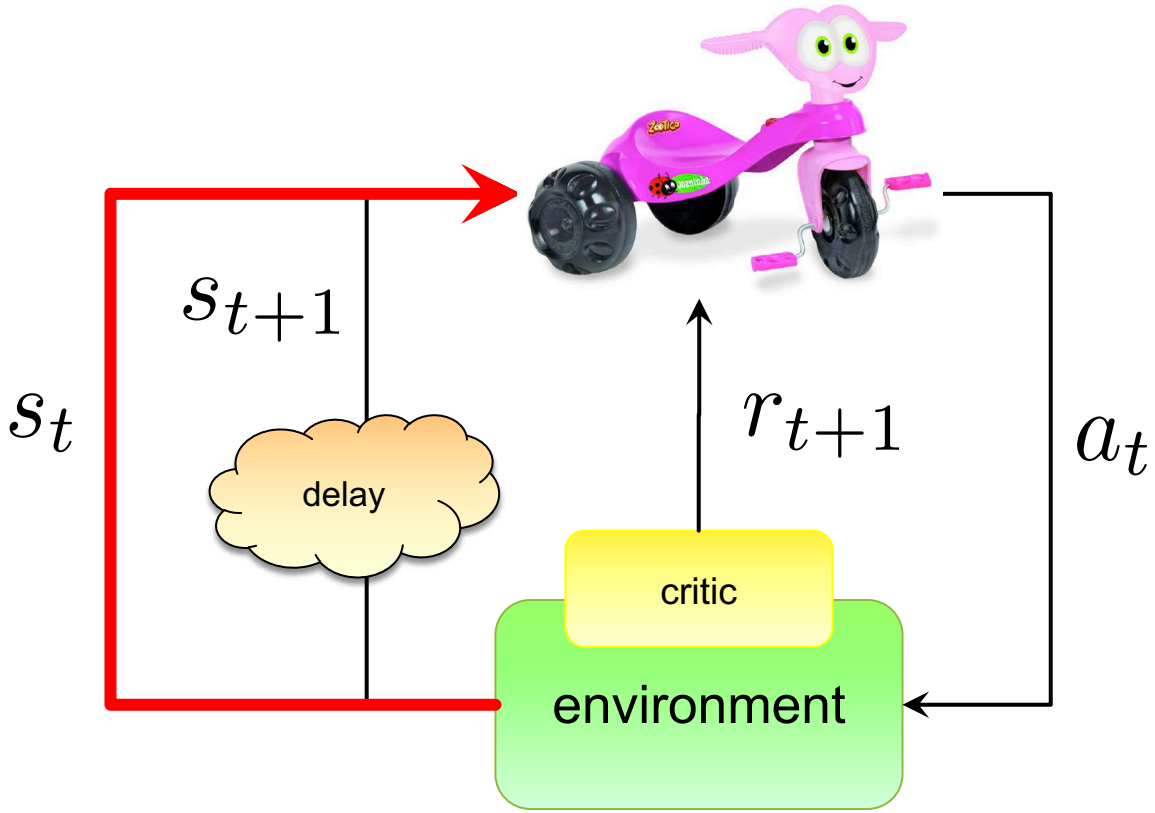
Task of

RL allows us to deal with sequential decision problems

yed reward to choose **sequences of**
produce the **greatest cumulative reward**

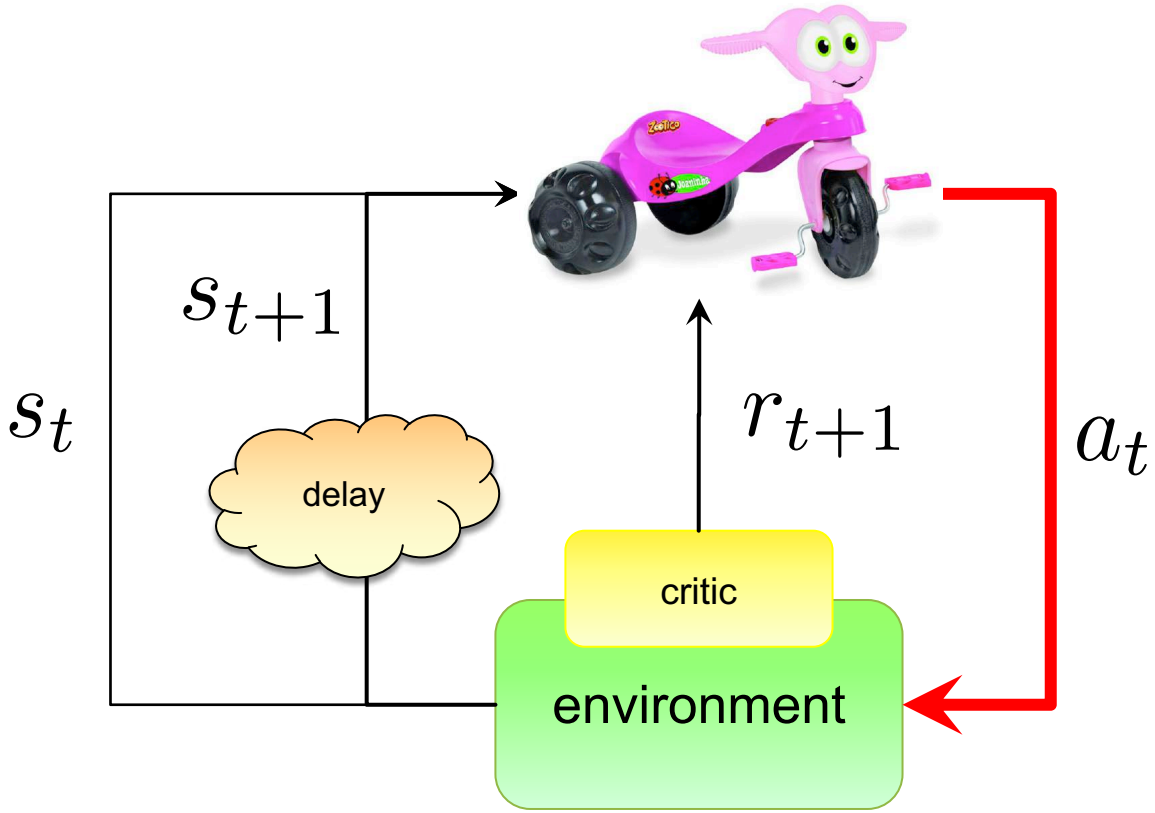
Consider an agent that wants to learn to ride a tricycle


s_{t+1}
s_t
delay
r_{t+1}
a_t
critic
environment



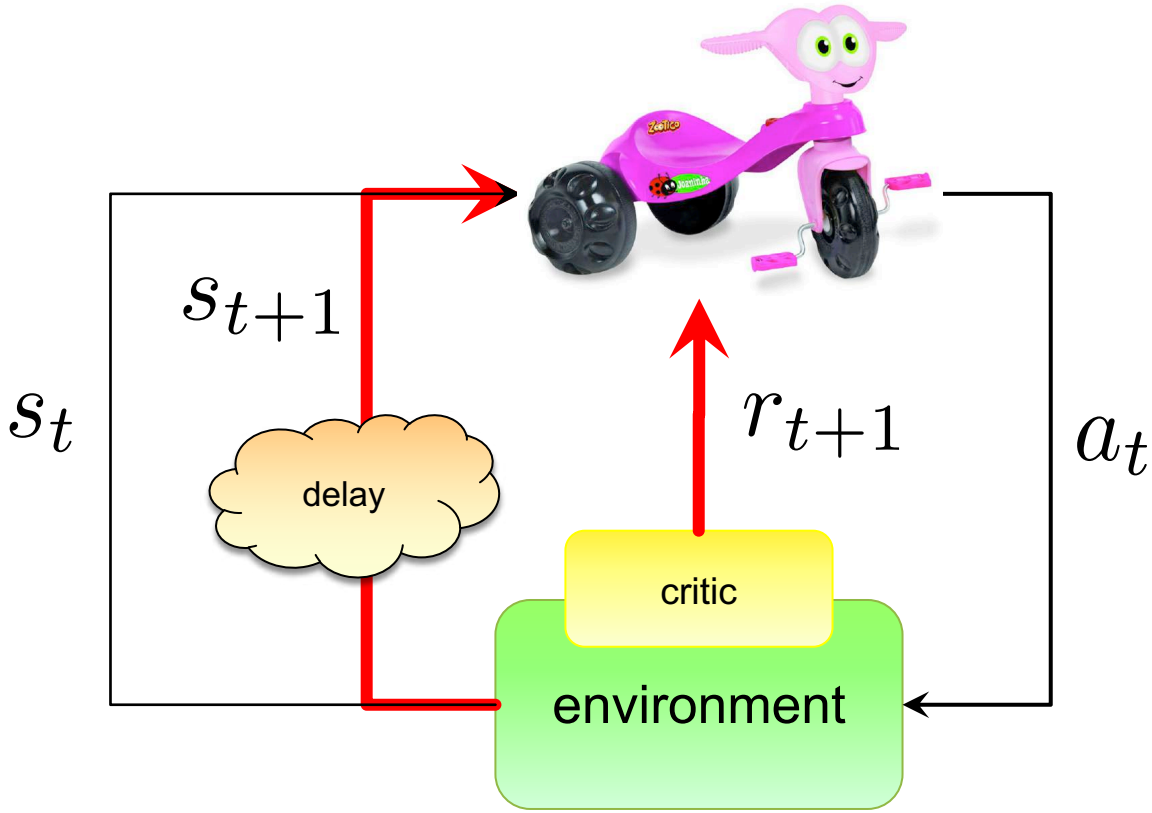
<*position, speed*>

S_t



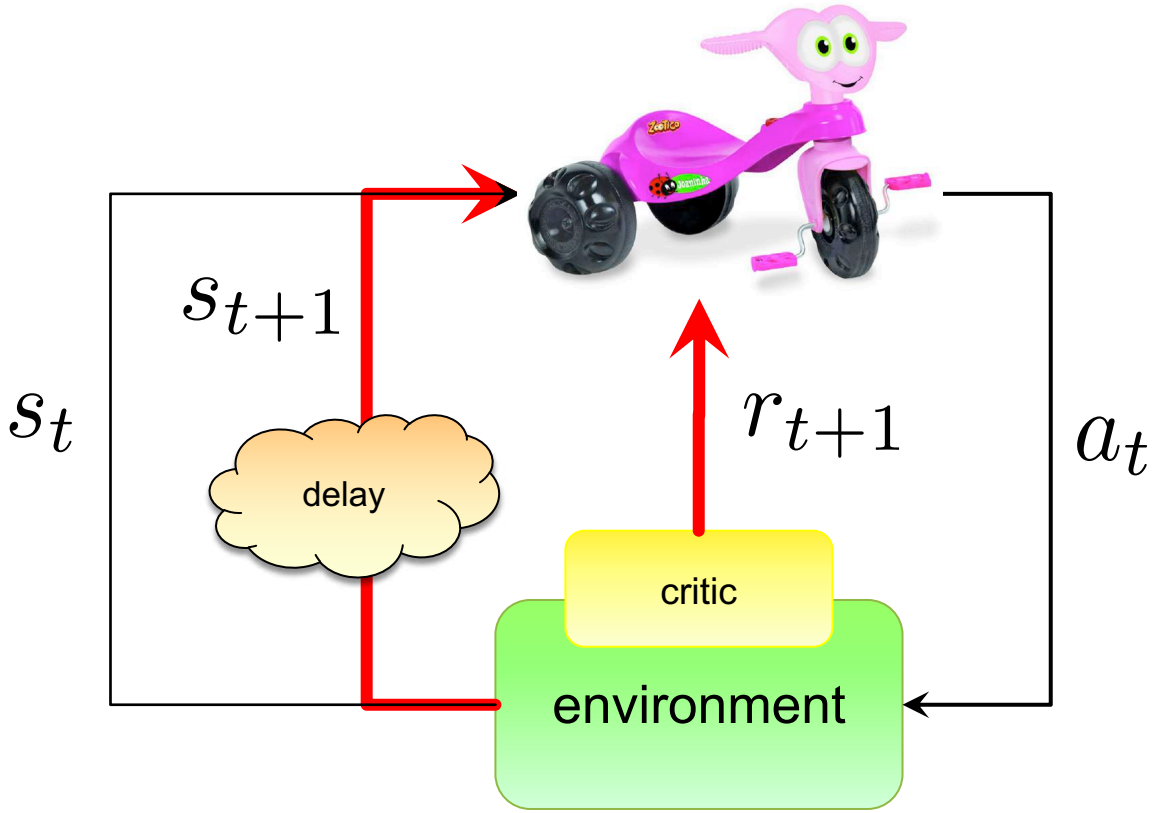
<position, speed> *<handlebar, pedals>*

s_t a_t



<position, speed> *<handlebar, pedals>* ⇨ *<new position, new speed>,* *advancement*

s_t a_t s_{t+1} r_{t+1}



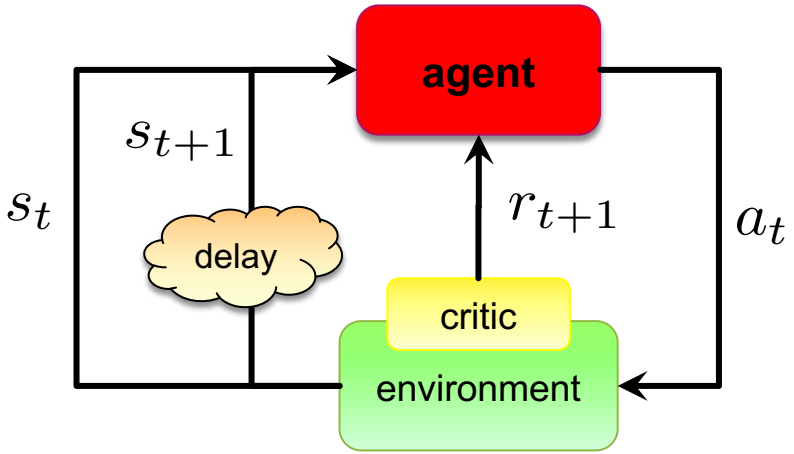
Experience:

$\langle s, a, s', r \rangle_1$

<position, speed> *<handlebar, pedals>* ⇨ *<new position, new speed>,* *advancement*

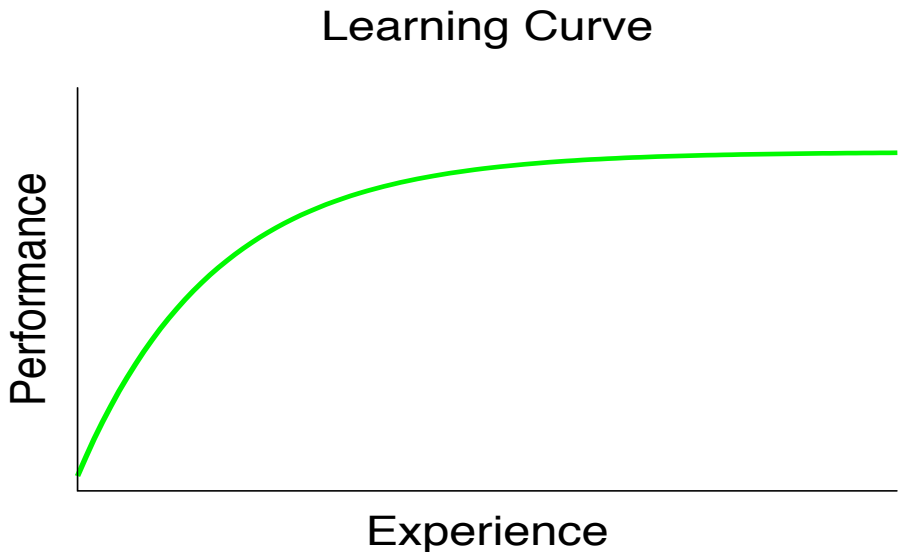
s_t a_t s_{t+1} r_{t+1}

Reinforcement Learning



Many experiences:

$$\langle s, a, s', r \rangle_1$$
$$\langle s, a, s', r \rangle_2$$
$$\cdots$$
$$\cdots$$
$$\cdots$$
$$\langle s, a, s', r \rangle_n$$



MDP – Model Formulation

An MDP is defined as $\langle S, A, T, R, \gamma \rangle$:

- **S** is the set of possible system states (arbitrary finite set);
- **A** is the set of allowable actions (arbitrary finite set);
- **T**: $S \times A \times S \rightarrow [0,1]$ is the **transition probability function**;
- **R**: $S \times A \rightarrow \Re$ is the **reinforcement function**;
- γ: discount factor that balances the trade-off between immediate rewards and future rewards, $\gamma \in [0,1]$.

MDP and RL

- In ***Reinforcement Learning***, we would like an agent to **learn** to behave well in an **MDP** world, but **without knowing** anything about **T** or **R** when it starts out
- The goal is to learn, for each state, the action that produces the **highest expected accumulated reward**

RL solution

- **Solution**: a policy, $\pi: S \longrightarrow A$, that maximizes:

$$Q^*(s,a) = E\left[\sum_{t=0}^{\infty} \gamma^t r(s,a)\right]$$

Optimal solution: $\pi^*(s) = \arg\max_a Q^*(s,a)$

Value learning: Q-learning

- Q-learning is a **value-based method** that estimates Q*(s,a) with experiences $< s, a, s', r >$:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha\left[\left(r + \gamma \max_{a'} \hat{Q}(s',a')\right) - \hat{Q}(s,a)\right]$$

$\alpha \in [0,1]$ is the learning rate.

Q values

- Q(s,a) is the expected discounted future reward for starting in state s, taking a as our first action, and then continuing optimally.

$$\pi^* = arg\ max_a\ Q^*(s,a)$$

- The agent only needs to consider each available action a in its current state s and chooses the action that maximizes Q(s,a).
- **Q(s,a) summarizes all the information** needed to determine the discounted cumulative reward that will be gained in the future if a is selected in s.

Exploration x Exploitation

Which experimentation strategy produces most effective learning?

- The agent has to **exploit** what it has already experienced in order to obtain reward, but ….
- It also has to **explore** in order to gather new information.

Trade-off: Exploration vs. Exploitation

- **ε-Greedy**:

$$a_t = \begin{cases} a_t^* & \text{with probability } 1-\varepsilon \quad \textbf{exploitation} \\ \text{random action with probability } \varepsilon & \textbf{exploration} \end{cases}$$

- **Softmax** action selection (Boltzmann distribution):

Choose *a* on *t* with probability: $\dfrac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$,

τ is the "computational temperature"

Q-learning Algorithm

Initialize Q(s,a) arbitrarily
Observe the current state $\boldsymbol{s}_t$
do forever (stop criterium)
select an action $\boldsymbol{a}_t$ and execute it in $\boldsymbol{s}_t$
receive immediate reward $\boldsymbol{r(s_t,a_t)}$
observe the new state $\boldsymbol{s}_{t+1}$
update Q(s,a) as follows:
$Q_{t+1}(s_t,a_t) \leftarrow \mathbf{(1-\alpha)}\, Q_t(s_t,a_t) + \alpha\, [r(s_t,a_t) + \gamma \max_a Q_t(s_{t+1},a)]$
$\boldsymbol{s}_t \leftarrow s_{t+1}$

Q-learning Algorithm

Initialize Q(s,a) arbitrarily
Observe the current state s_t
do forever (stop criterium)
select an action a_t and execute it in s_t
receive immediate reward $r(s_t,a_t)$
observe the new state s_{t+1}
update Q(s,a) as follows:
$Q_{t+1}(s_t,a_t) \leftarrow (1-\alpha)\, Q_t(s_t,a_t) + \alpha\, [r(s_t,a_t) + \gamma \max_a Q_t(s_{t+1},a)]$
$s_t \leftarrow s_{t+1}$

Exploration
x
Exploitation

Q-learning Algorithm

Initialize Q(s,a) arbitrarily
Observe the current state $\boldsymbol{s}_t$
do forever (stop criterium)
select an action $\boldsymbol{a}_t$ and execute it in $\boldsymbol{s}_t$
receive immediate reward $\boldsymbol{r(s_t,a_t)}$
observe the new state $\boldsymbol{s}_{t+1}$
update Q(s,a) as follows:
$Q_{t+1}(s_t,a_t) \leftarrow \mathbf{(1-\alpha)}\, Q_t(s_t,a_t) + \alpha[r(s_t,a_t) + \gamma \max_a Q_t(s_{t+1},a)]$
$\boldsymbol{s}_t \leftarrow s_{t+1}$

Learning rate

- The basic form of the update looks like this:

$$X_{t+1} \leftarrow (1 - \alpha) X_t + \alpha New_t$$

α is a learning rate; usually it is 0.1 or 0.2. It is quite typical (and, in fact, **required for convergence**), to start with a large alpha, and then decrease it over time.

- So, we are updating our estimate of X to be mostly like our **old** value of X, but adding in a new term ***New***
  - This kind of update is essentially a **running average** of the new terms received on each step.

Q-learning Algorithm

There are two iterative processes going on:

- One is the usual kind of **averaging** we do, when we collect a lot of samples and try to estimate their mean (using the learning rate)
- The other is the **dynamic programming** iteration done by value iteration, updating the value of a state based on the estimated values of its successors.

Q-learning notes

- **Session**: at the beginning of each session, start the Q values with zeros or random values
- **Episode**: each learning restart, without restarting the Q values (it stops when reaching the target or timeout).
  - Typically, one presents the result as the average of sessions (each session comprising a number of episodes) in different iterations.