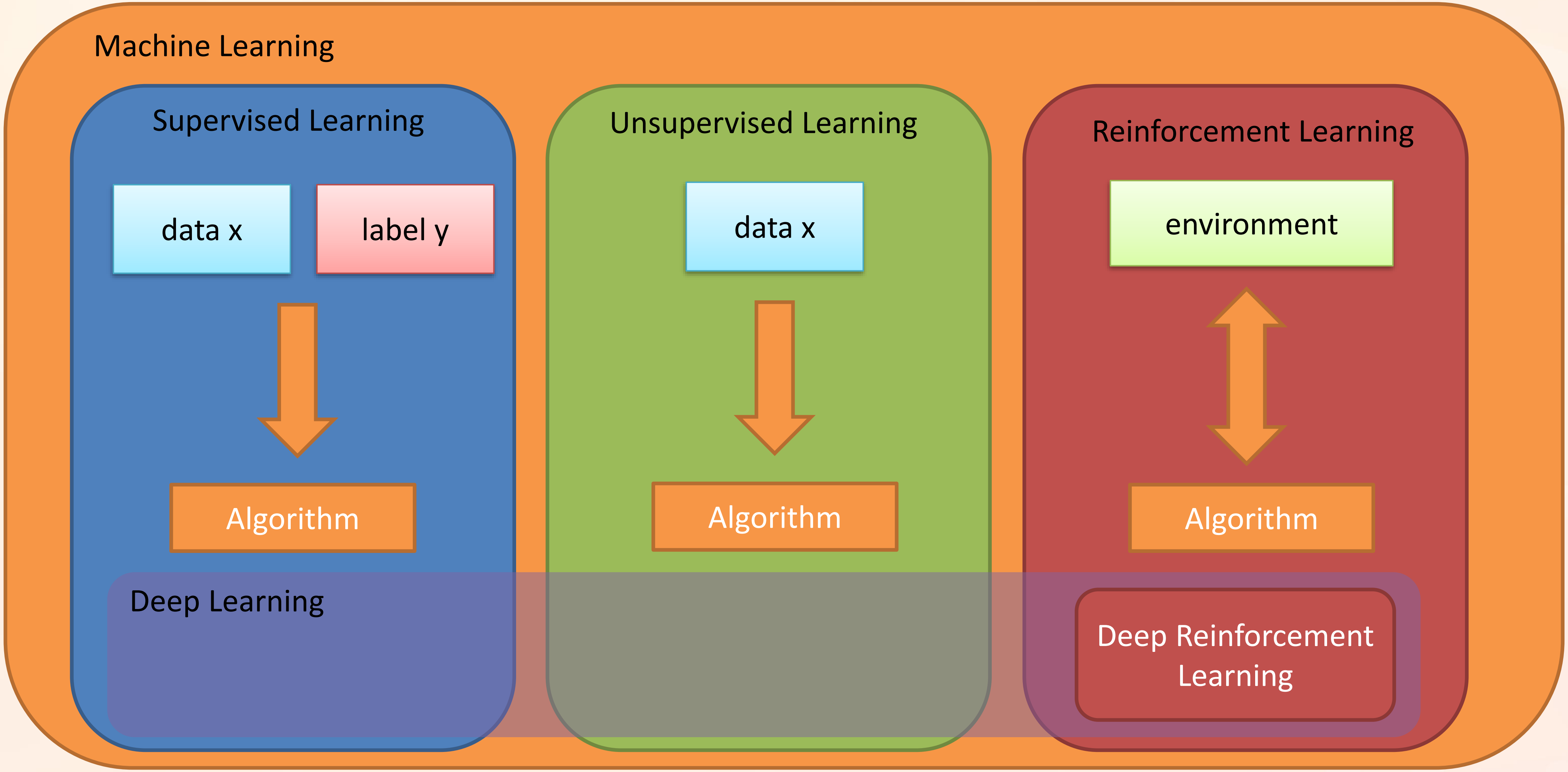
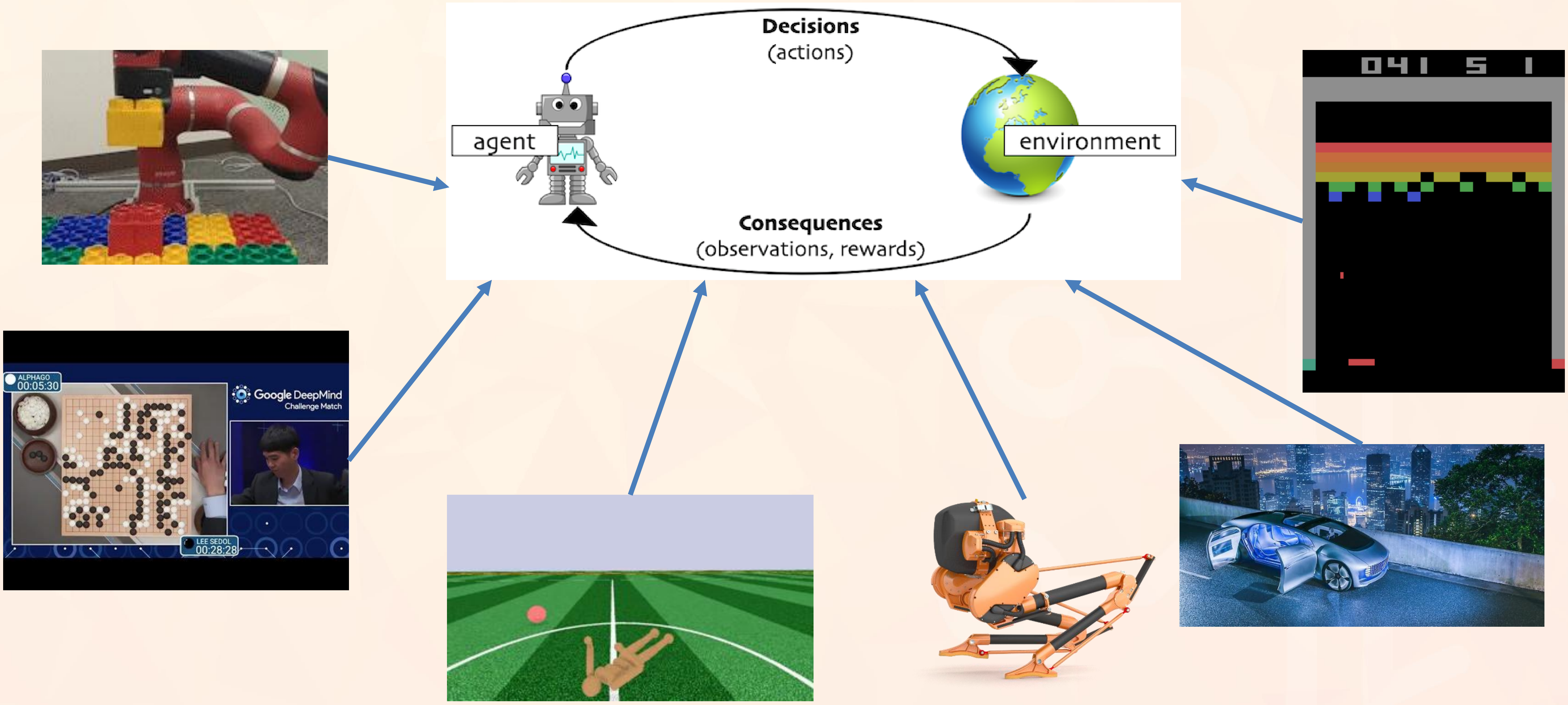
Reinforcement Learning course

By Timothé Boulet

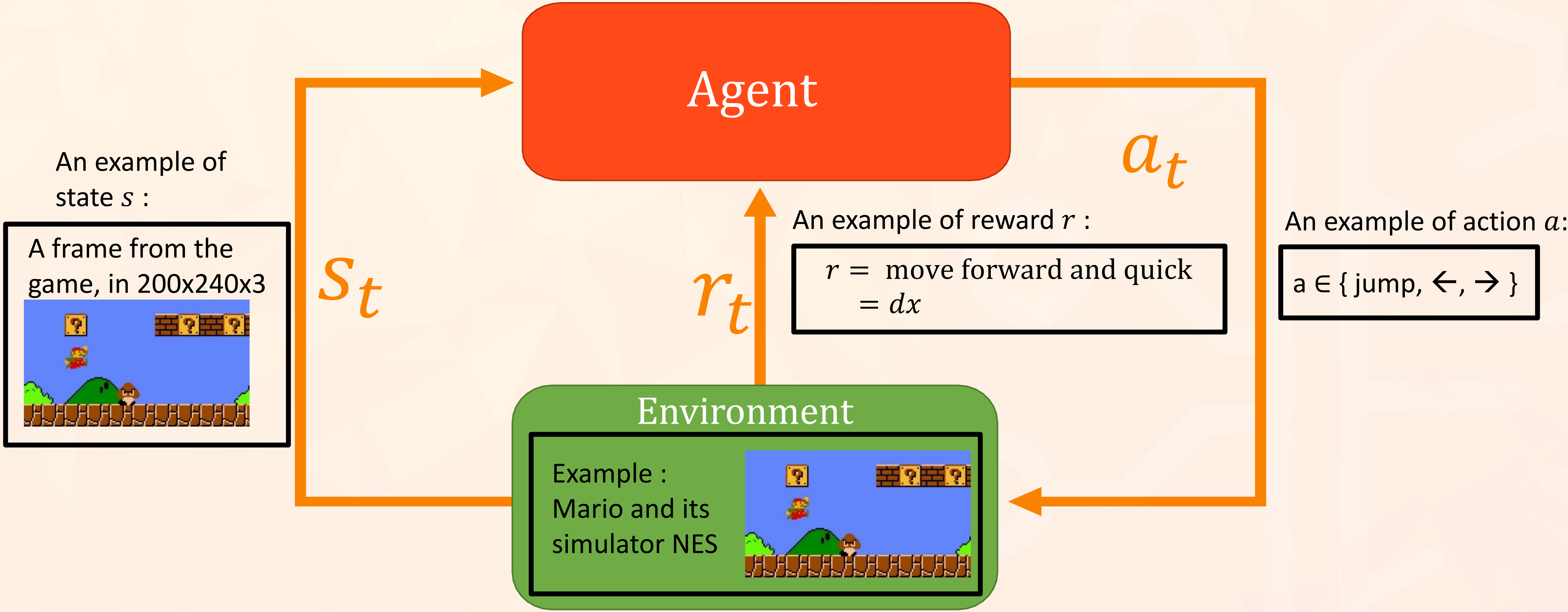
3 main subfield of Machine Learning



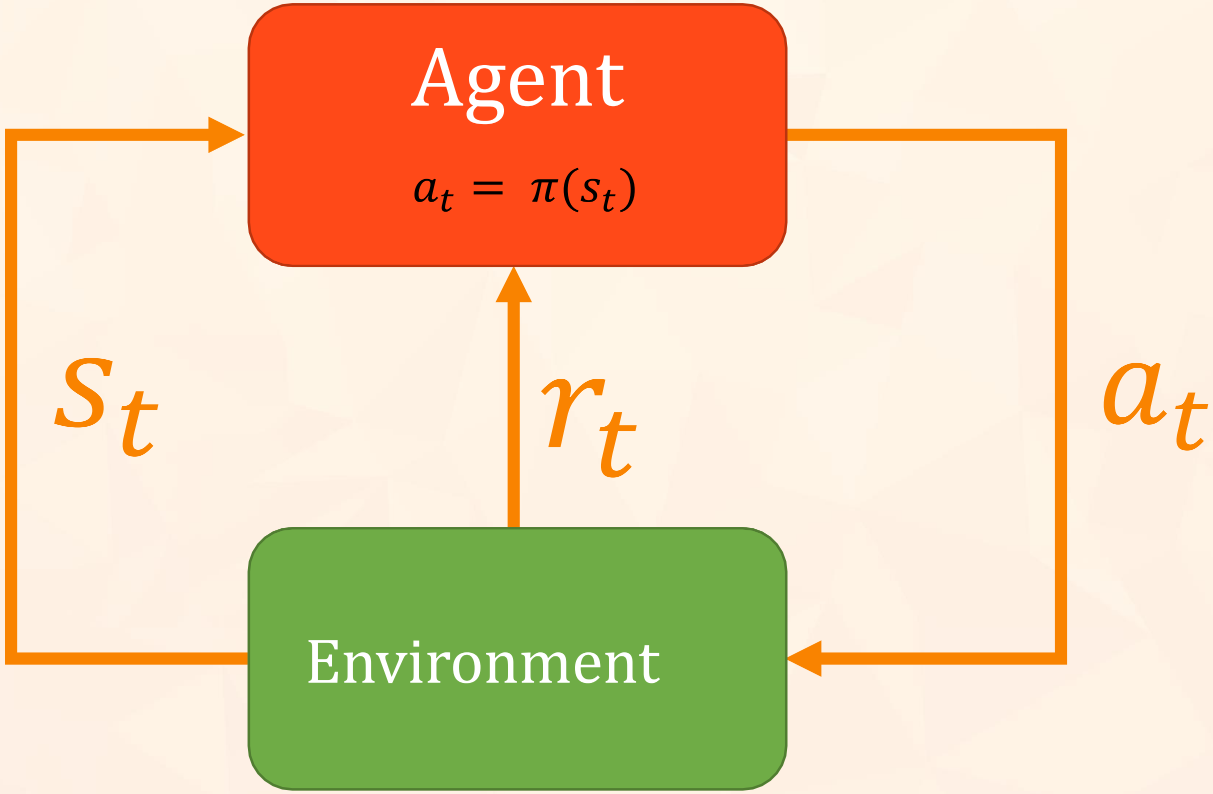
RL applications



The RL framework : env. & agent interaction



RL Framework : the policy π



Agent has a policy π

$$\pi(s) = a$$

or

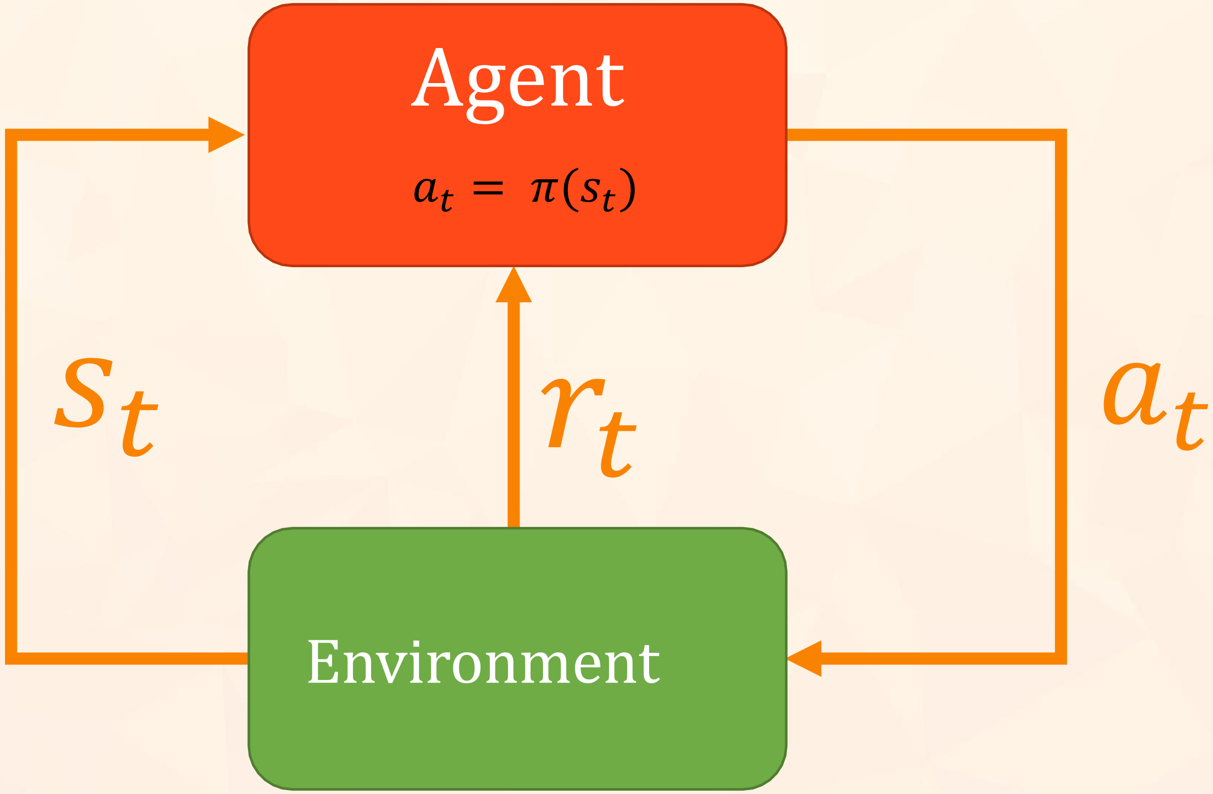
$$\pi(a|s) = P(A_t = a|S_t = s)$$

or

$$\begin{cases} \pi(s) = \text{loi L} \\ a \sim L \end{cases}$$

$\pi($  $) = \begin{cases} \rightarrow & 90\% \\ \leftarrow & 5\% \\ \uparrow & 5\% \end{cases}$

RL Framework : transitions & episodes τ



An episode τ :

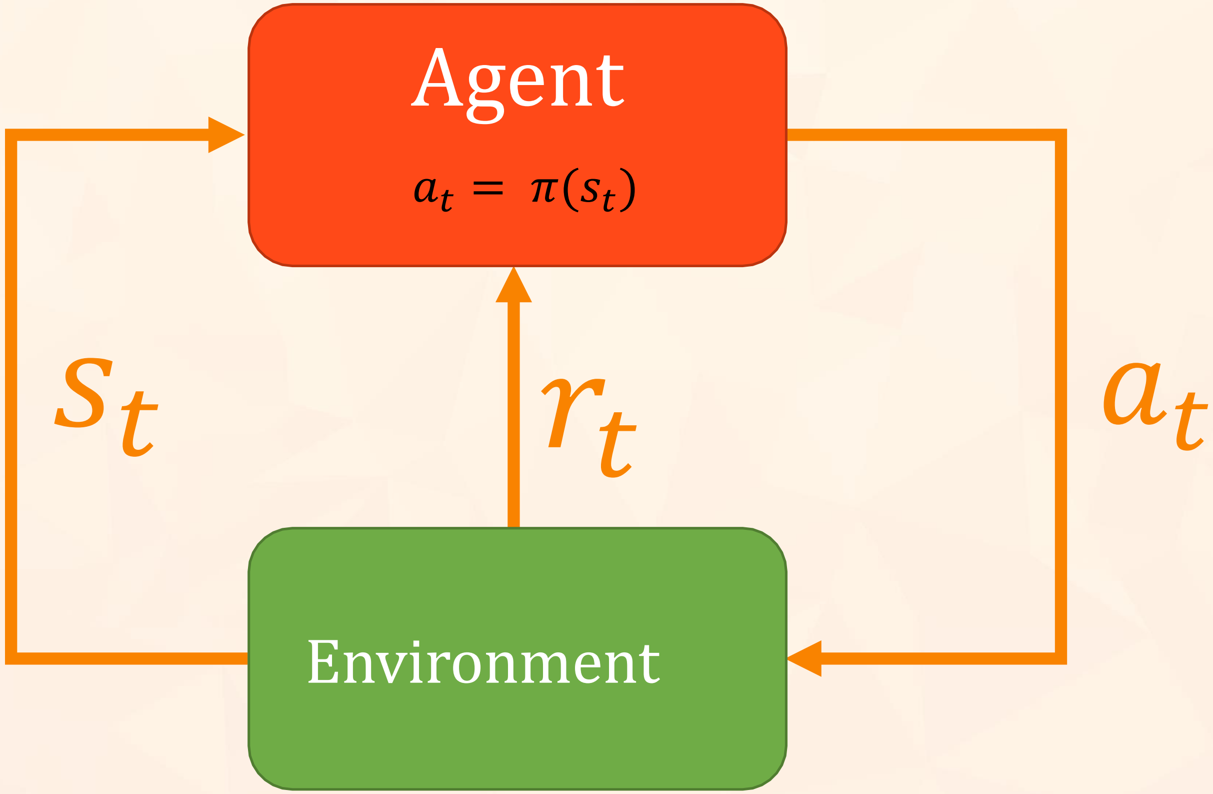
$$\tau = (s_0, a_0, r_0, \underbrace{s_1, a_1, r_1}_{\text{a transition}}, \dots, s_T, a_T, r_T)$$

For Mario, an episode = a run on a level

The env. start in state $s_0 \in S_{initiaux}$
It ends at $t = T$ when $s_t \in S_{finaux}$

Comment : there may be non terminal environments !

RL Framework : cumulative future reward G_t

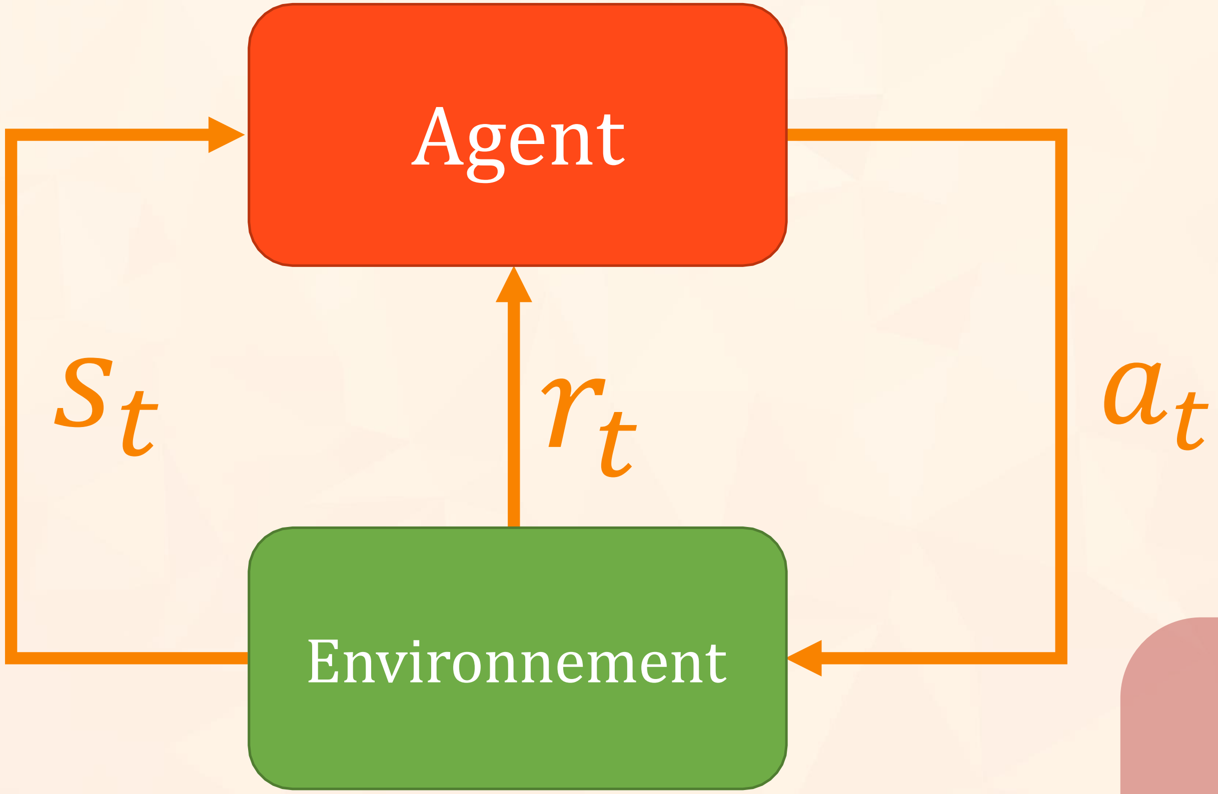


<u>Goal</u> : maximise the return G_t :

$$G_t = \sum_{t' \geq t}^{T} r_{t'}$$

<u>Objective</u> : Find $\pi^* = \underset{\pi}{\operatorname{argmax}} \, E[G_t | \pi]$

RL Framework : the environment



The environment is determined by probability distributions we call the <u>model</u> :

$$P^{a}_{s \to s'} = P(S_{t+1} = s' | \, S_t = s, A_t = a)$$

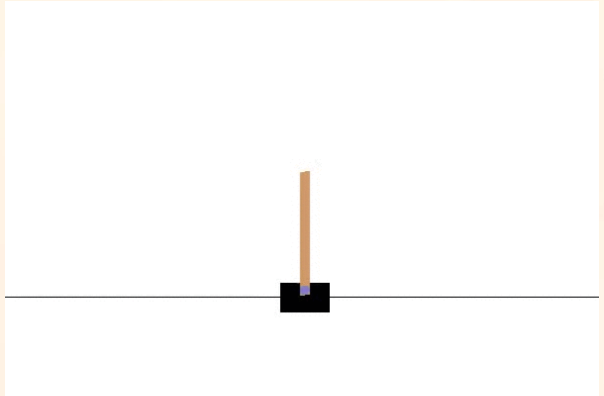
$$R^{a}_{s} = E[R_t | S_t = s, A_t = a]$$

Reinforcement Learning

Model-based	Model-free
We know the model and we will exploit it directly	We need to interact with the environment

Some examples of environments

Example 1 : CartPole



State : s = (position and speed in x and θ)

Action : a ∈ {←, →}

Reward : r = +1

Example 2 : Video game such as Mario

State : s = 

Action : a ∈ { jump, ←, → }

Reward : r = $\frac{dx}{dt}$

Some examples of environments

Example 3 : Chess (against a given opponent)

State : s =  ou (1.e4e5 2.Nc3Nf6 3.f4d5)

Action : a = next move

Reward : r = +1 when victory, -1 when defeat, 0 else

Example 4 : Robots

State : s = pressure/position sensors

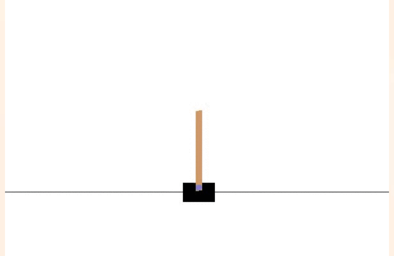
Action : a = orders for each robotic muscle

Reward : r = reward for standing up straight, moving an object

Different kinds of environments

The env. can be :

- deterministic or stochastic (= include randomness)
  The goal is to find π that maximizes $E[G_t \mid \pi]$



- Terminal or not: T can be $+\infty$
  So that $G_t = \sum_{t' \geq t}^{+\infty} r_t$ doesn't diverges, we introduce the Discount Factor $\gamma \in [0,1[$. $\gamma = 0{,}99$ for example.

$$G_t = \sum_{t' \geq t}^{T} \gamma^{t'-t} r_t \quad = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

- Markovian or not :
  The Markov Property : the current state contains all the information of the previous states

Different kinds of environments

The environment can be:

- parfaitement observable ou non :
  On parlera alors d'observations plutôt que d'état : $o_t = x(s_t)$

- Model-based ou Model-free:
  Model-based = accès au modèle $P^{a}_{s \to s'}$, et $\mathrm{R}^{\mathrm{a}}_{\mathrm{s}}$ (cas des échecs et autres jeux adversariaux)
  Model-free = modèle inaccessible/trop complexe, nécessité d'interagir avec l'environnement (Mario, CartPole, simulateurs physiques)

Reinforcement Learning

Model-based	Model-free
On connait le modèle de l'env. et on va l'exploiter directement	On a besoin d'interagir avec l'env.

Different kinds of environments

Environnement	Deterministic Or Stochastic	Terminal ?	Observability ?	Markovian ?	Model-based or Model-free
Mario	Deterministic	Yes	Partial	No	Model-free
Chess (given $\pi_{opponent}$)	Stochastic if $\pi_{opponent}$ is	No	Total	Yes	Model-based
CartPole	Deterministic	No (but we stop it at $h = 500$ steps)	Total	Yes	Model-free
Realistic robotic environment	Stochastic	No	Partial	Depends on the quality of the sensors	Model-free

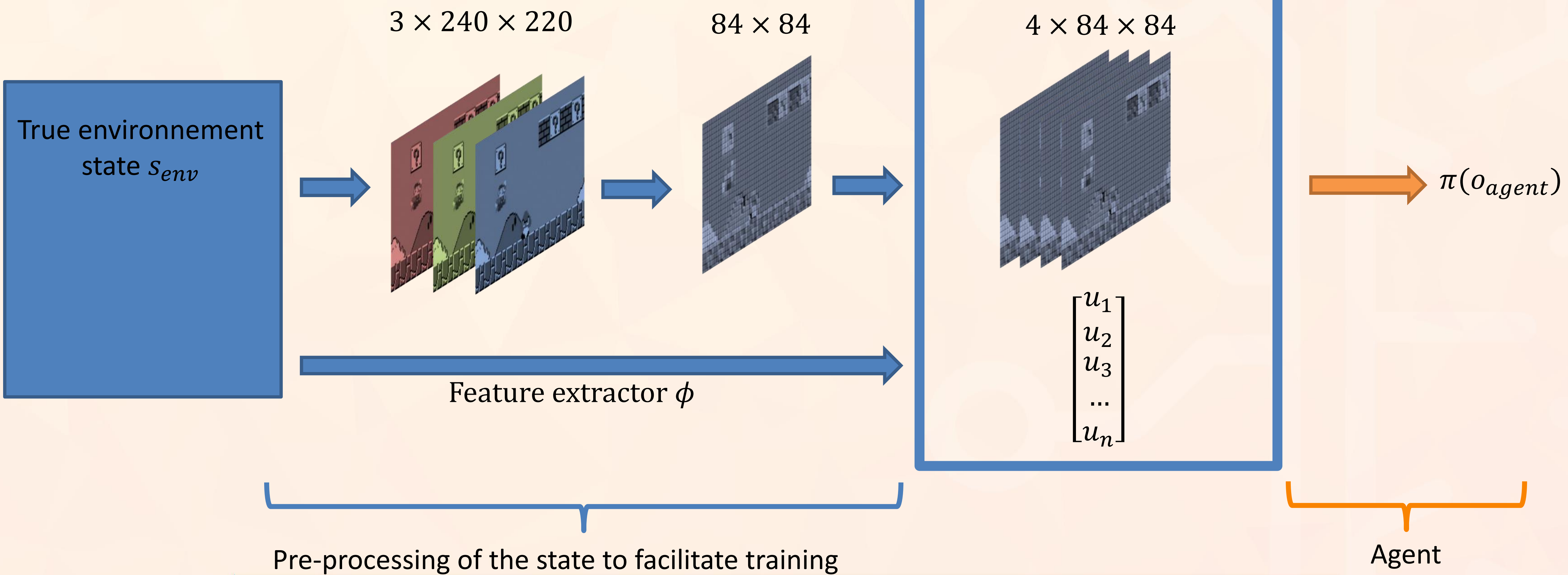
Some env. are also non-stationary (the model changes over time) and require the application of algorithms that adapt continuously.

Environnement Shaping

Pre-treatment of the environment: states

Observation space: must be condensed enough

- Rich enough to contain all important information
- Small enough to allow for quick learning



Pre-treatment of the environment: reward

Reward as a goal: The agent maximizes the reward, so design it to match your goal
Reward as a signal: The relative values of the reward help the agent learn what to do

Reward sparse: reward rarely non-zero, difficult for the agent to learn
Dense reward: non-uniform reward, helps the agent to accomplish sub-goals

Environnement	Reward sparse	Reward dense
Mario	+1 on successful level	$\frac{dx}{dt}$
Échecs	+1/-1 at the end of the game	n for each piece taken, $+/-100$ at end of game
Labyrinthe	+1 on exit	$\frac{d}{dt}(proximity\ to\ goal)$

Reward shaping

Poorly defined reward can lead to unexpected behavior

Pre-treatment of the environment: action

Reduce the action space as much as possible :

Actions :
a ∈ { jump, ←, →, jump + →, jump + ←, fireball, fireball + →, … }

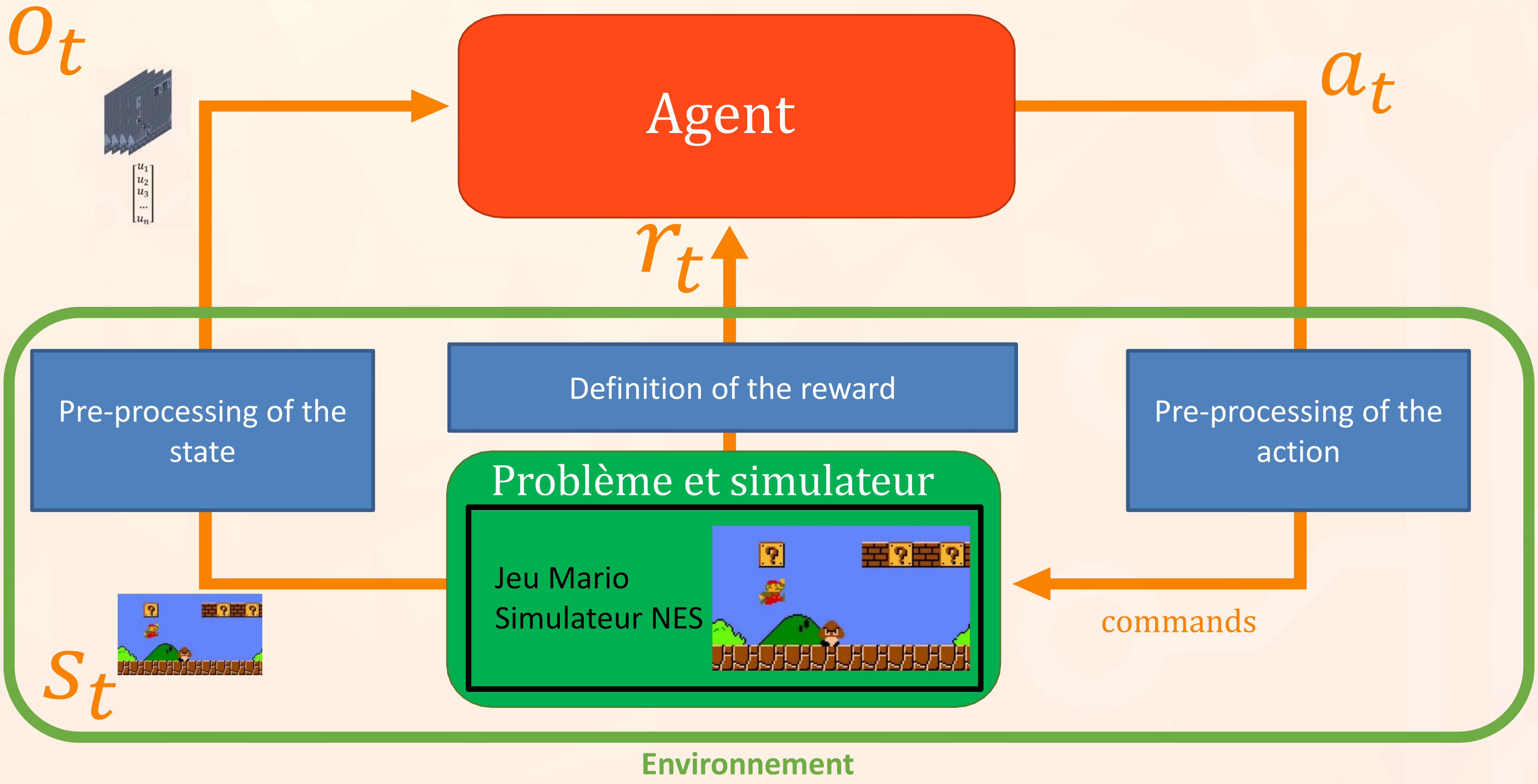
Actions that are « sufficient » :
a ∈ {→, jump + →}

Create automated actions

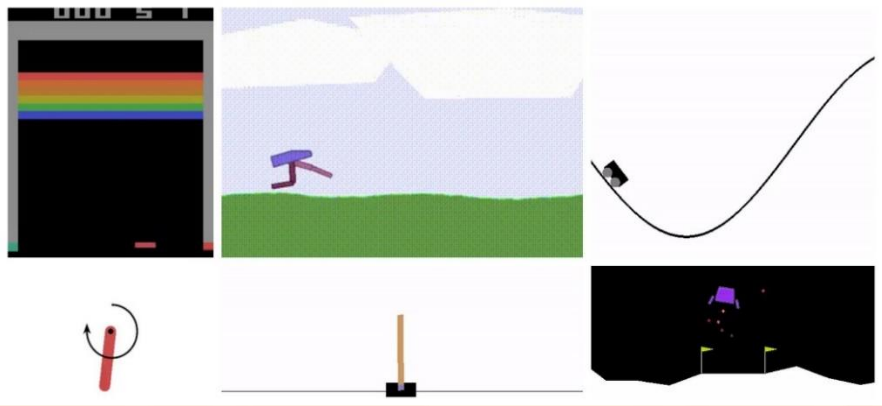
Example: a = "kill enemy“
Taking the action a has the effect of generating a sequence of commands that are supposed to kill the next enemy in the game.

Note: one can even have a "sub-agent" learn to perform macro-action a well, and in this case it is called hierarchical reinforcement learning.

Pre-treatment of the environment



Define your own environments



Gym : Library to define your environments or use already implemented ones.

OceanEnv : example of a custom environment with Gym :

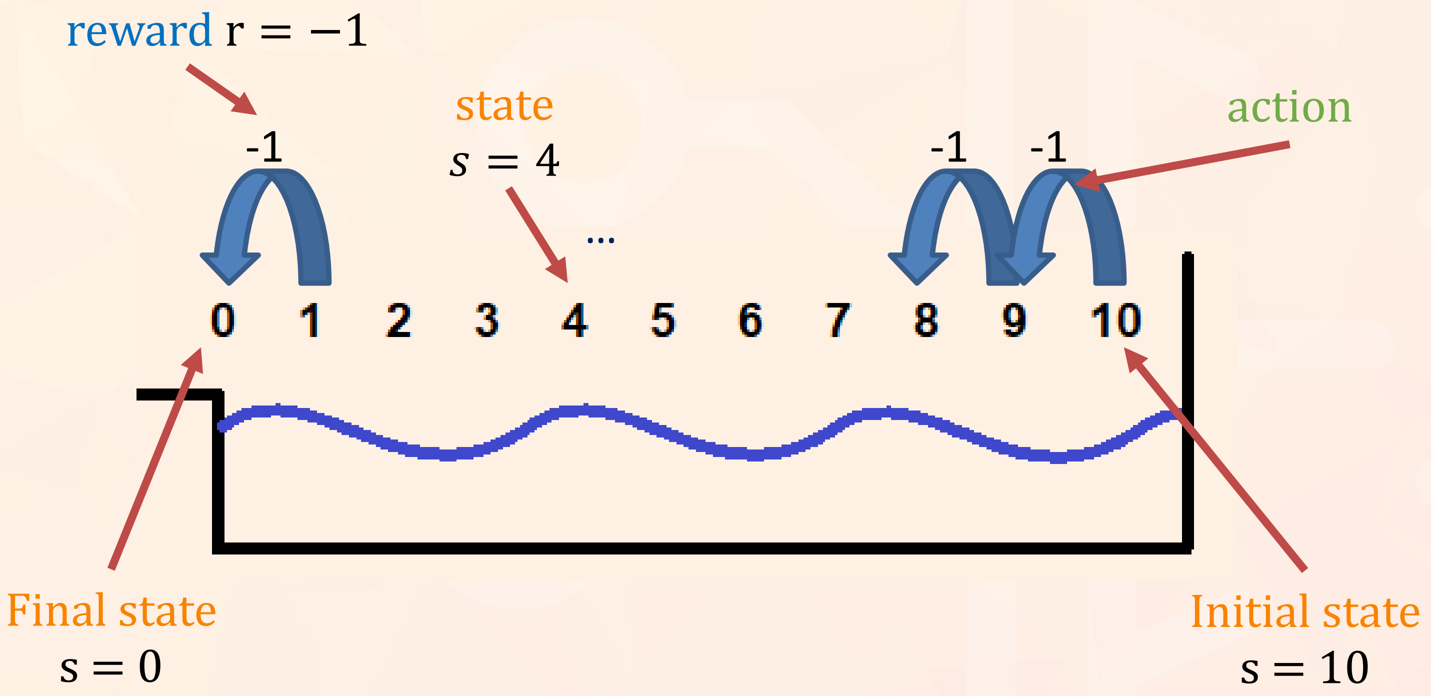
État : $s \in \{0, 1, \ \dots, \ 10\}$, is the distance to the shore

Action : $a \in \{$"move away, move towards"$\}$

Reward : $r = -1$ at each t(punishment as long as the shore is not reached)

Initial state : $s_0 = 10$

Final state: $s_T = 0$



How to learn ?

State value and action value

State value:

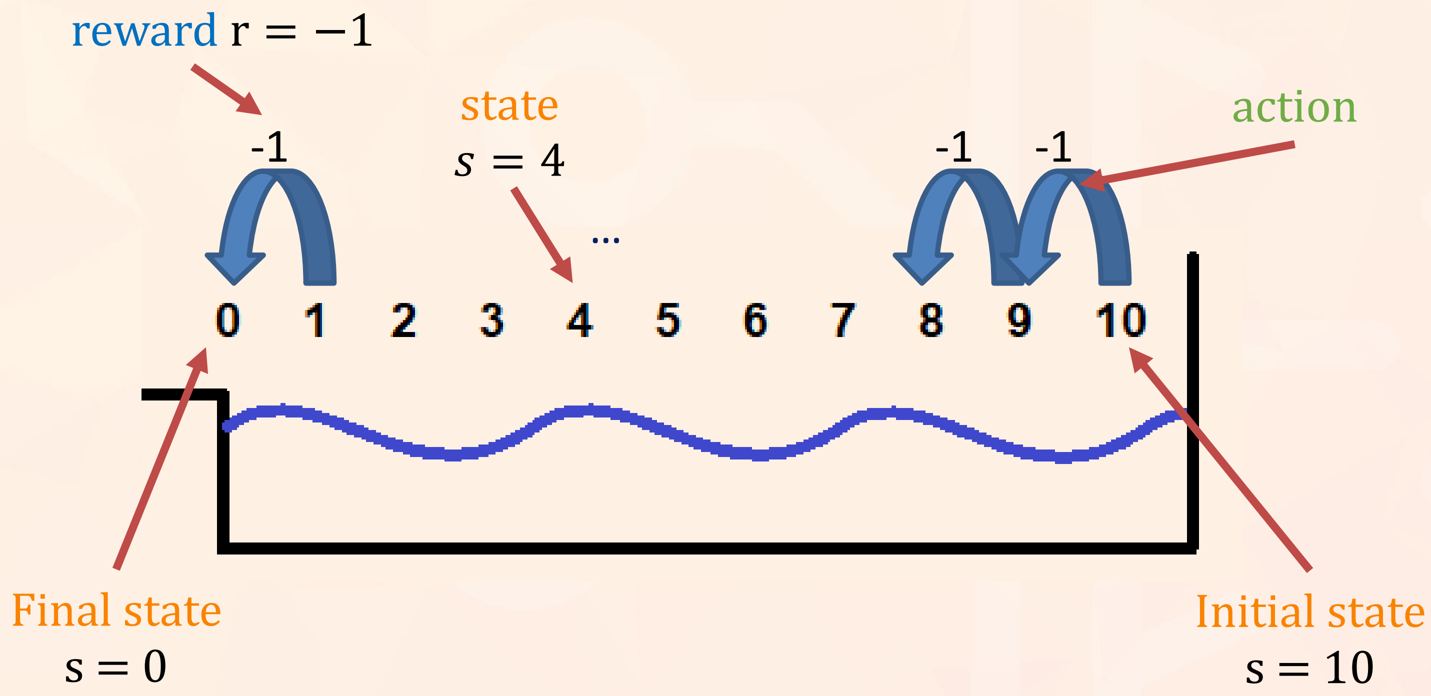
$v_\pi(s) = E[G_t|\ S_t = s, \pi\ ]$ how good is my state s with the policy π

Action value :

$q_\pi(s,a) = E[G_t|\ S_t = s, A_t = a, \pi\ ]$ how good my action is in the state s and with the policy π

$$v_{\pi_{get\ closer}}(s) = ?$$
$$v_{\pi_{move\ away}}(s) = ?$$

$$q_{\pi_{get\ closer}}(s, get\ closer) = ?$$
$$q_{\pi_{get\ closer}}(s, move\ away) = ?$$



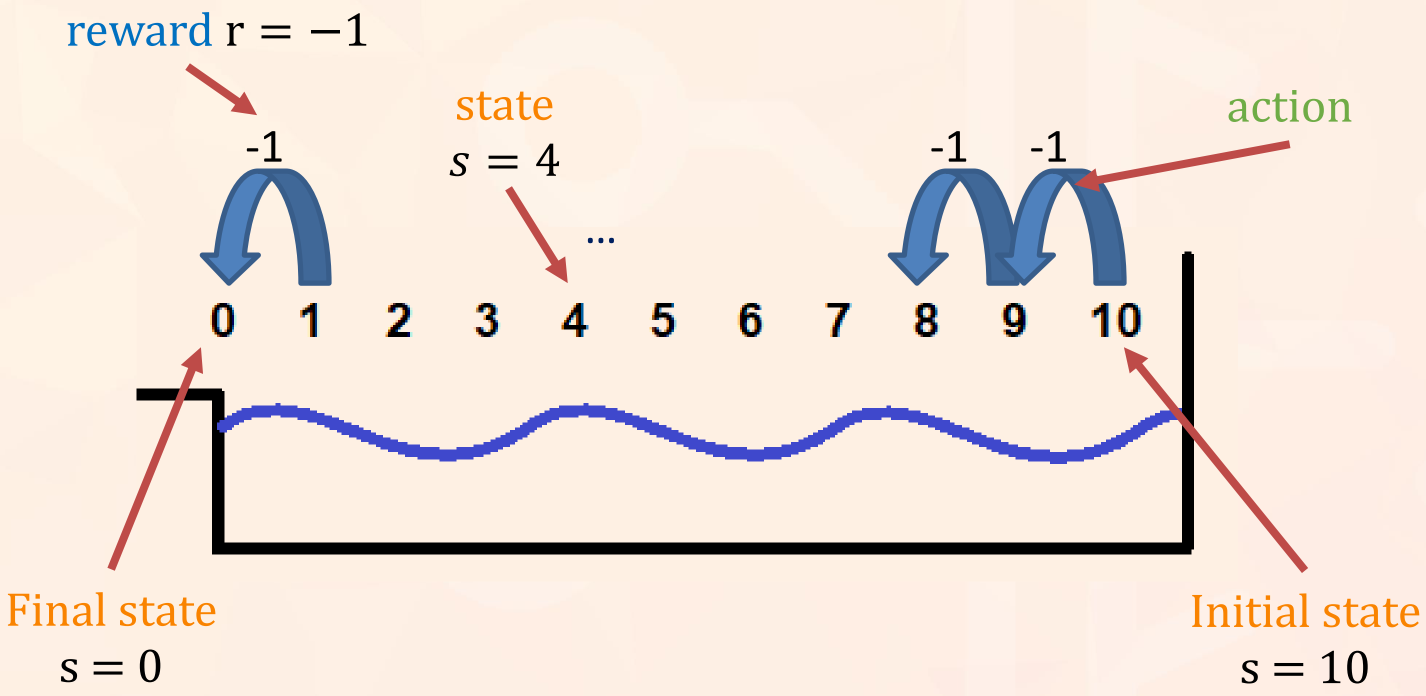
State value and action value

State value:
$v_\pi(s) = E[G_t |\, S_t = s, \pi\,]$ how good is my state s with the policy π

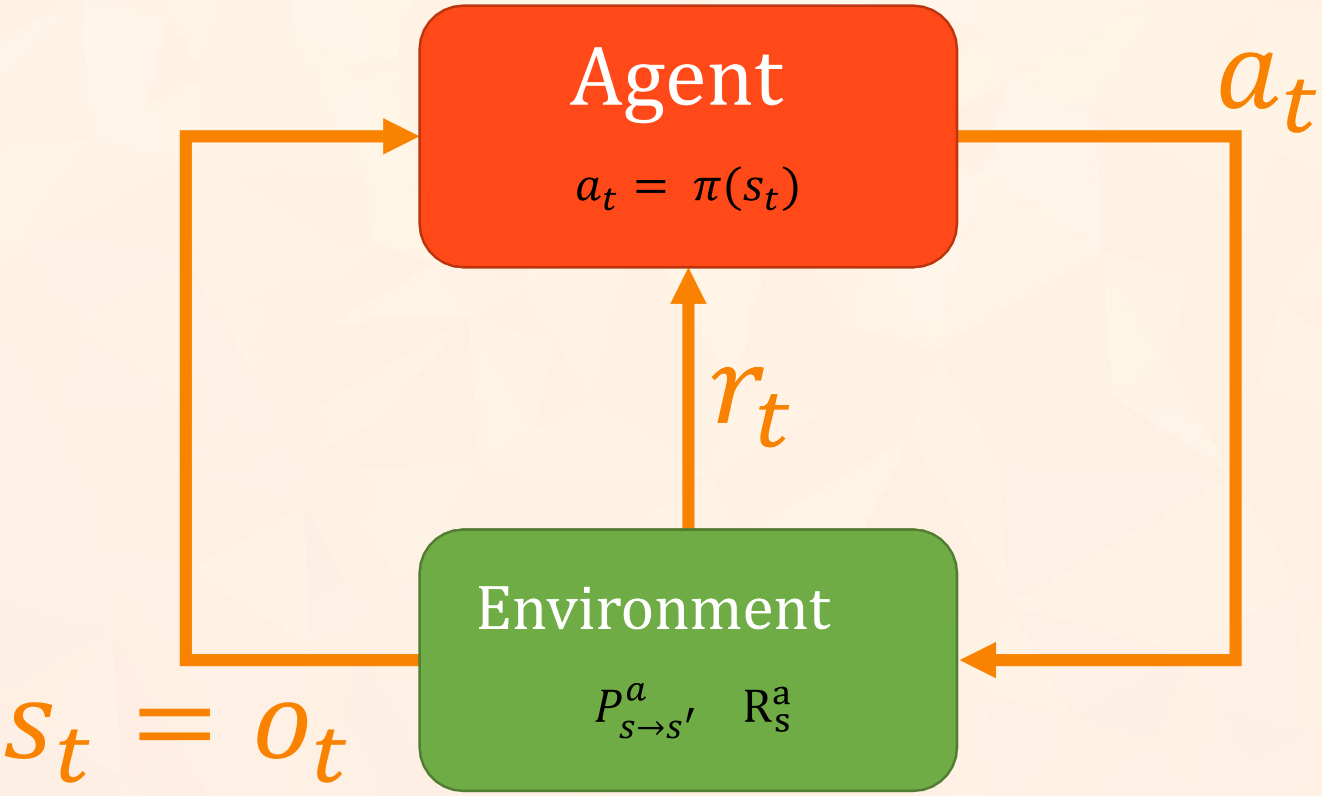
Action value :
$q_\pi(s, a) = E[G_t |\, S_t = s, A_t = a, \pi\,]$ how good my action is in the state s and with the policy π

$$v_{\pi_{get\ closer}}(s) = -s$$
$$v_{\pi_{move\ away}}(s) = -\infty$$

$$q_{\pi_{get\ closer}}(s, get\ closer) = -s$$
$$q_{\pi_{get\ closer}}(s, move\ away) = -s - 2$$



Markovian Decision Process



MDP:

- Environment is fully Markovian
- Environment is fully Observable

$MDP = (S, A, P^a_{s\to s'}, R^a_s, P_{s_0})$

Policy :

$$\pi(a|s) = P(A_t = a|S_t = s)$$

State value:

$$v_\pi(s) = E[G_t|\ S_t = s, \pi\ ]$$

Action value:

$$q_\pi(s, a) = E[G_t|\ S_t = s, A_t = a, \pi\ ]$$

Goal : Find π maximizing G_t :

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t}\, r_{t'}$$

Prediction Problem and Control Problem

Prediction Problem :

Compute values v_π and q_π

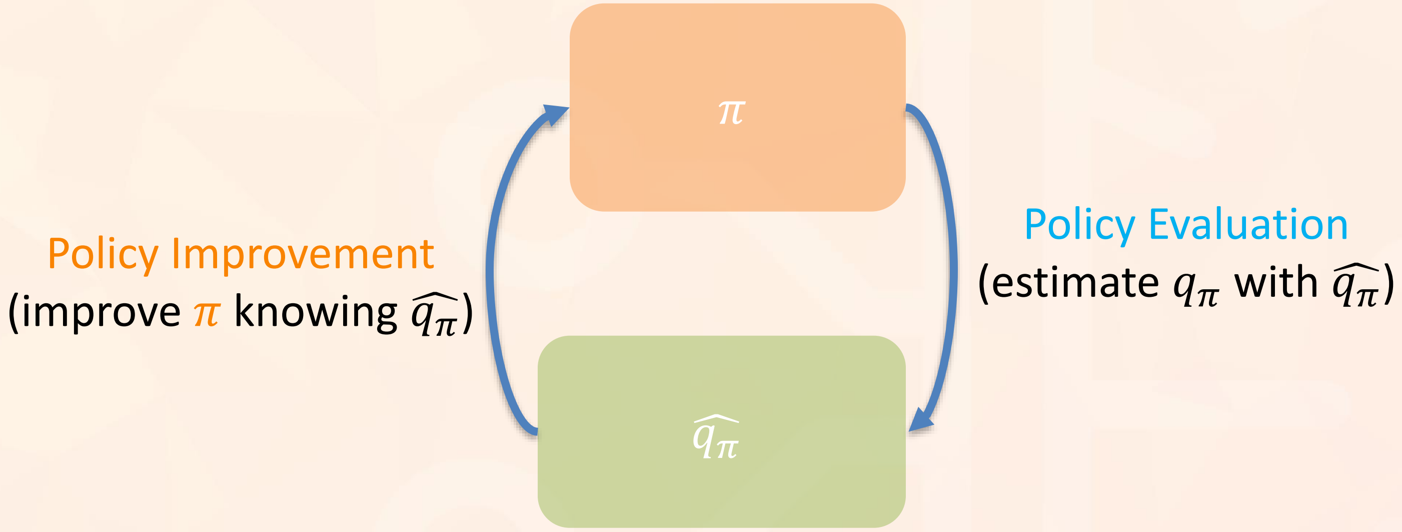
Estimators : $\hat{v}_\pi$ et $\hat{q}_\pi$

Control Problem

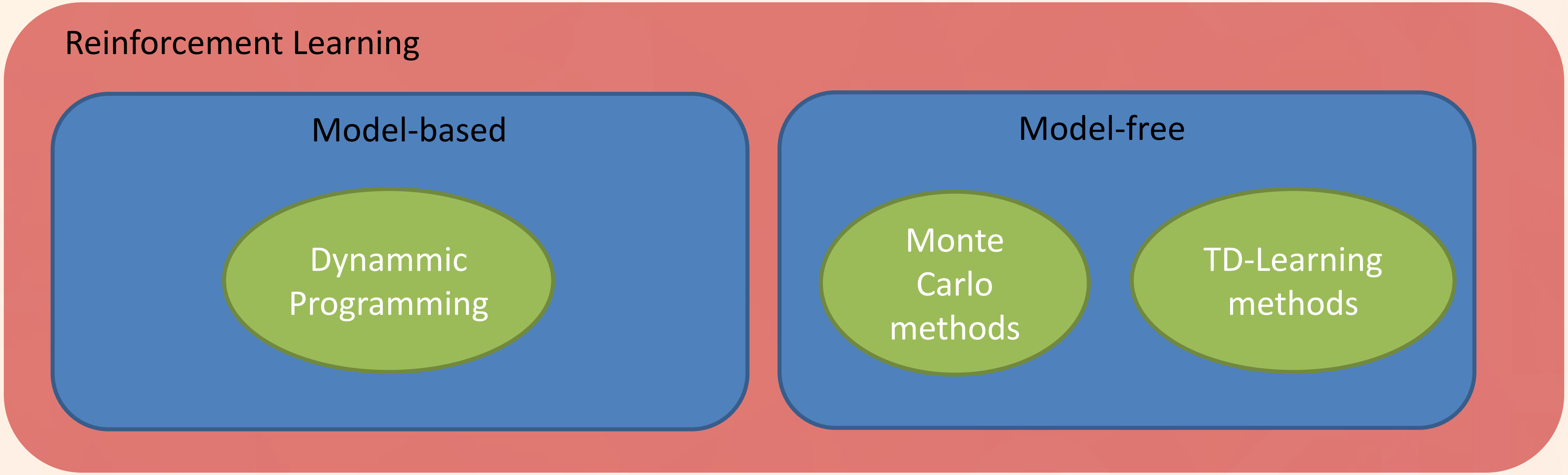
Improve π

Knowing $\hat{q}_\pi(s,a)$ allows you to choose the best actions

Generalized Policy Iteration :



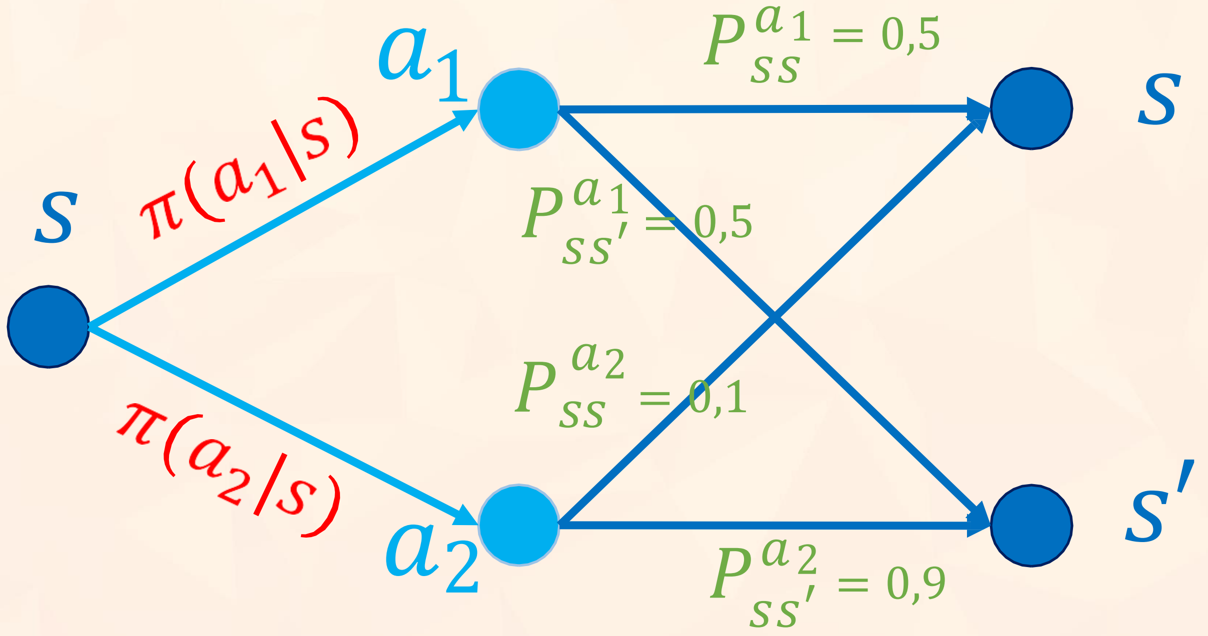
Model based vs. Model free RL



Model-Based Reinforcement Learning

Dynamic Programming : Prediction Problem

On know the model ($P^{a}_{s\rightarrow s'}$, R^{a}_{s}) and policy π. First, we seek to estimate $v_\pi(s)$ and $q_\pi(s,a)$ (Prediction Problem).



$$v_\pi(s) = \sum_{a\in A} \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = R^{a}_{s} + \gamma \sum_{s'\in S} P^{a}_{ss'} v_\pi(s')$$

Bootstrapping: use state/action values to calculate other state/action values

Note: here the actions and states are discrete and few.

Dynamic Ford-Bellman equations

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')\right)$$

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \left(\sum_{a' \in A} \pi(a'|s') q_\pi(s',a') \right)$$

By noting V the vector of state values $V = \left(v_\pi(s)\right)_{s \in S}$ one obtains an equation of the form $V = f(V)$.
Iterative convergence method :

$$\begin{cases} V_0 \text{ arbitrary} \\ V_{k+1} = f(V_k) \end{cases}$$

Iterative Policy Evaluation

Algorithm : Iterative Policy Evaluation
Algorithm used in Dynamic Programming for the Prediction Problem

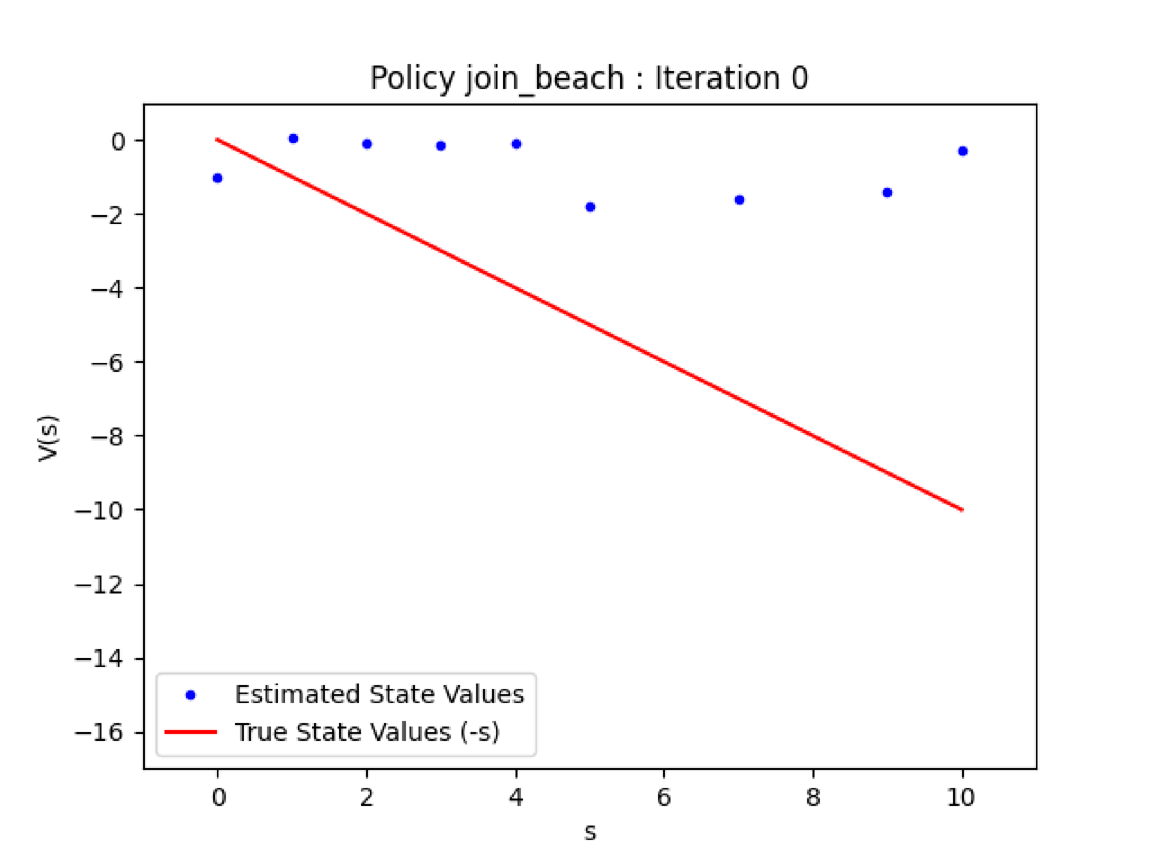
Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)\left(R_s^a + \gamma \sum_{a \in A} P_{ss'}^a v_\pi(s)\right)$$

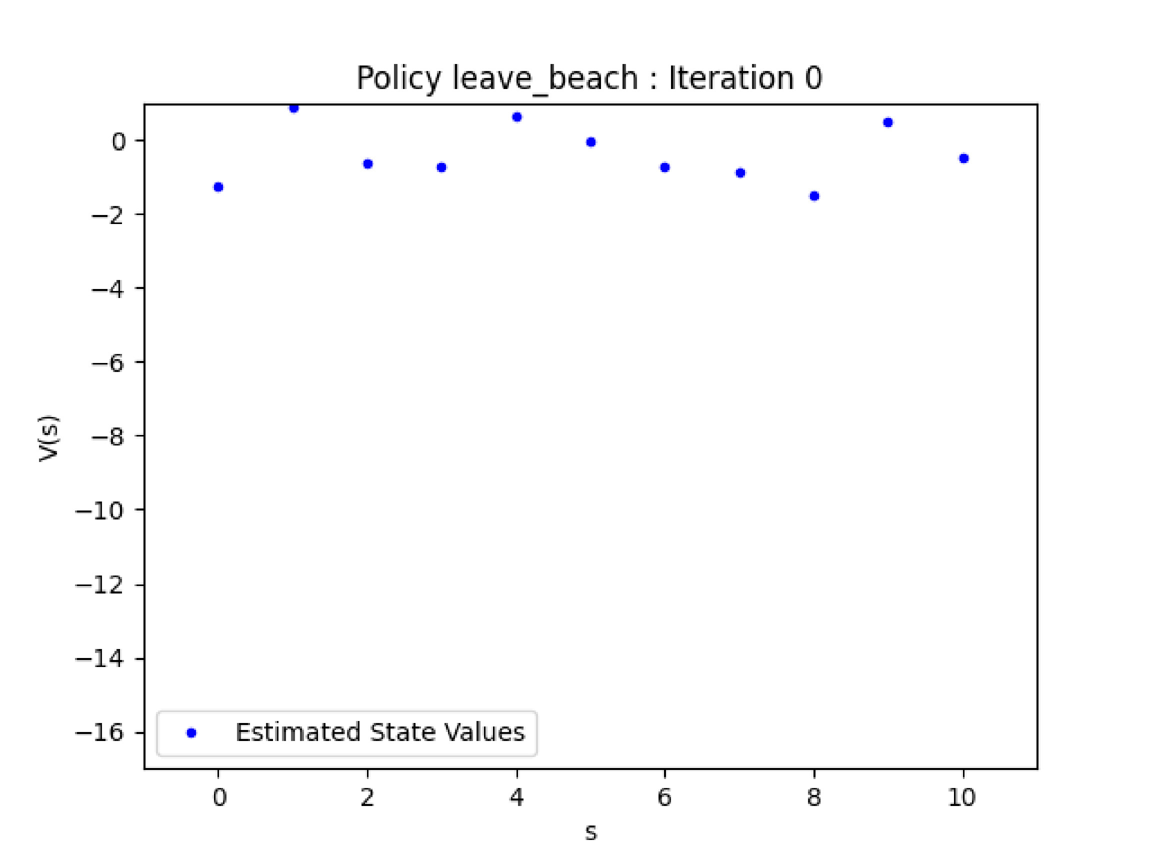
We get $\hat{v}_\pi(s) \approx v_\pi(s)$.

Iterative Policy Evaluation : Results

π = get closer
$(\gamma = 0{,}98)$



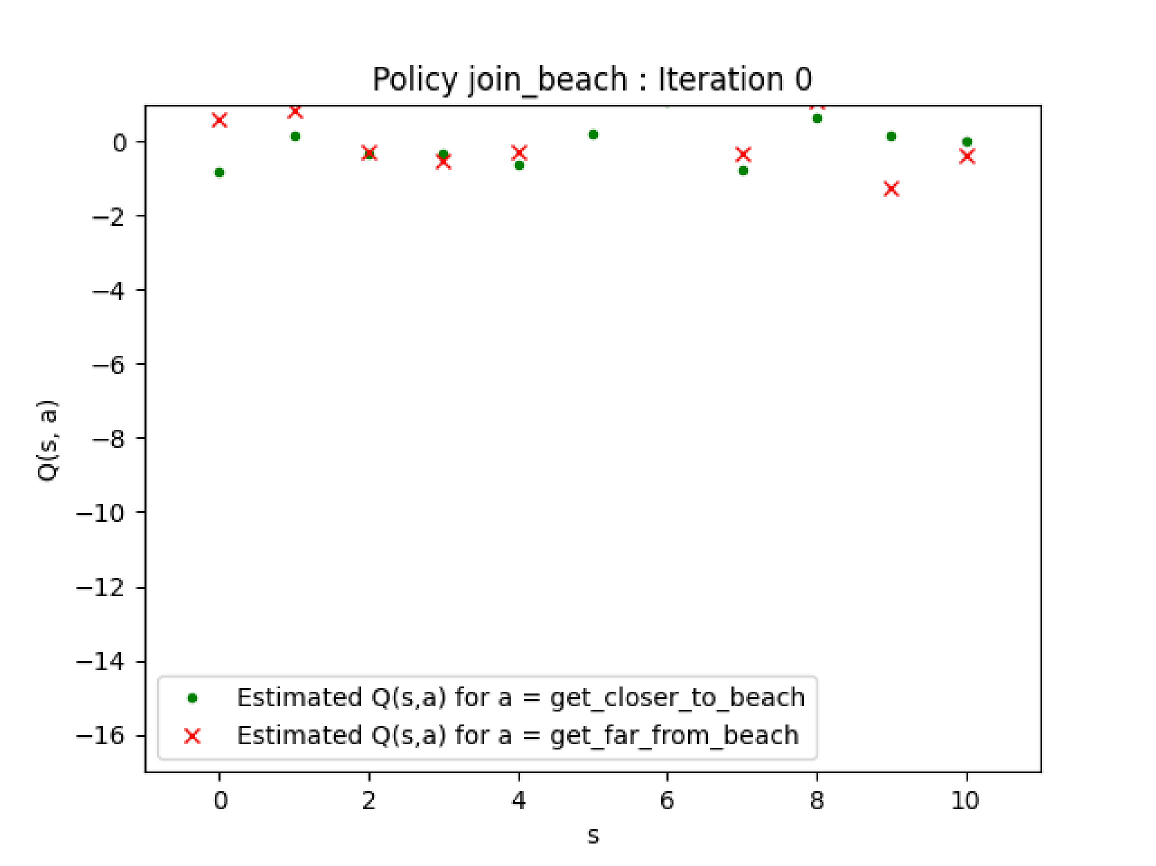
π = move away
$(\gamma = 0{,}8)$



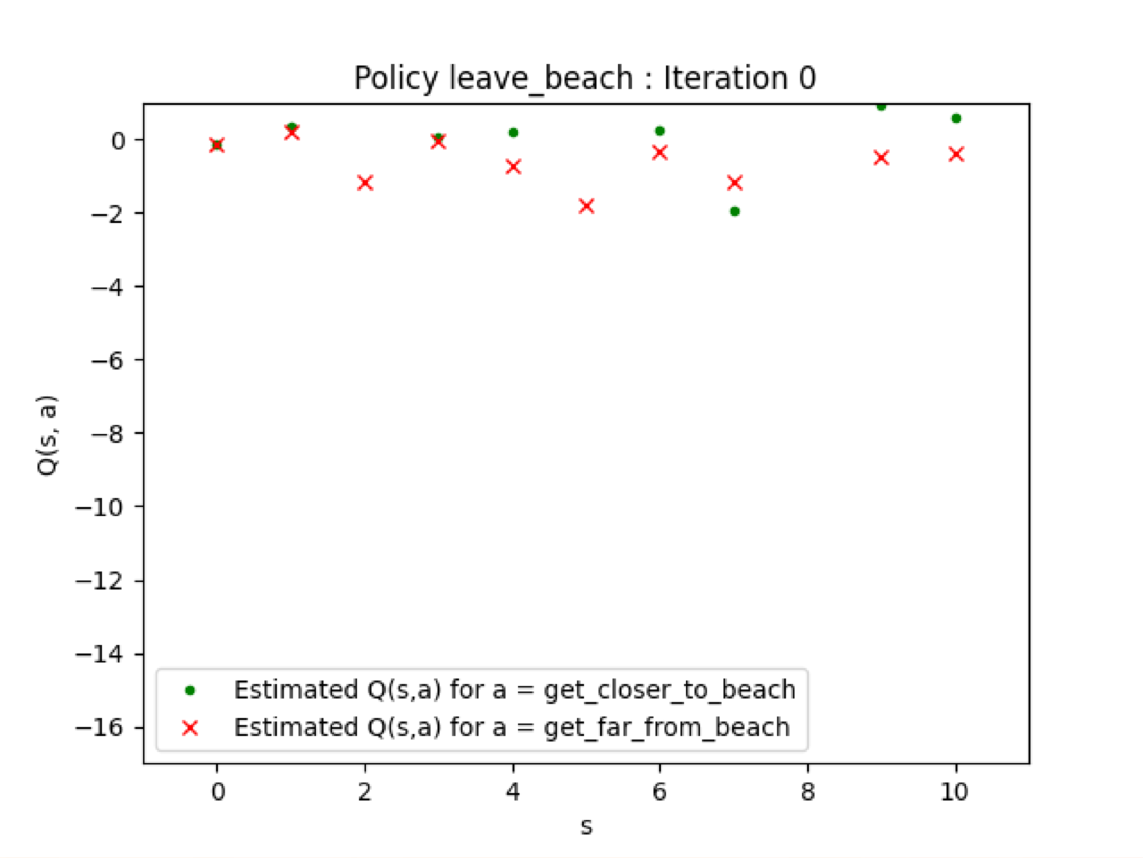
The order in which we go through the states is important

Iterative Policy Evaluation : Results

π = get closer
$(\gamma = 0{,}98)$



π = move away
$(\gamma = 0{,}8)$



The order in which we go through the states is important

From evaluating to Control Problem

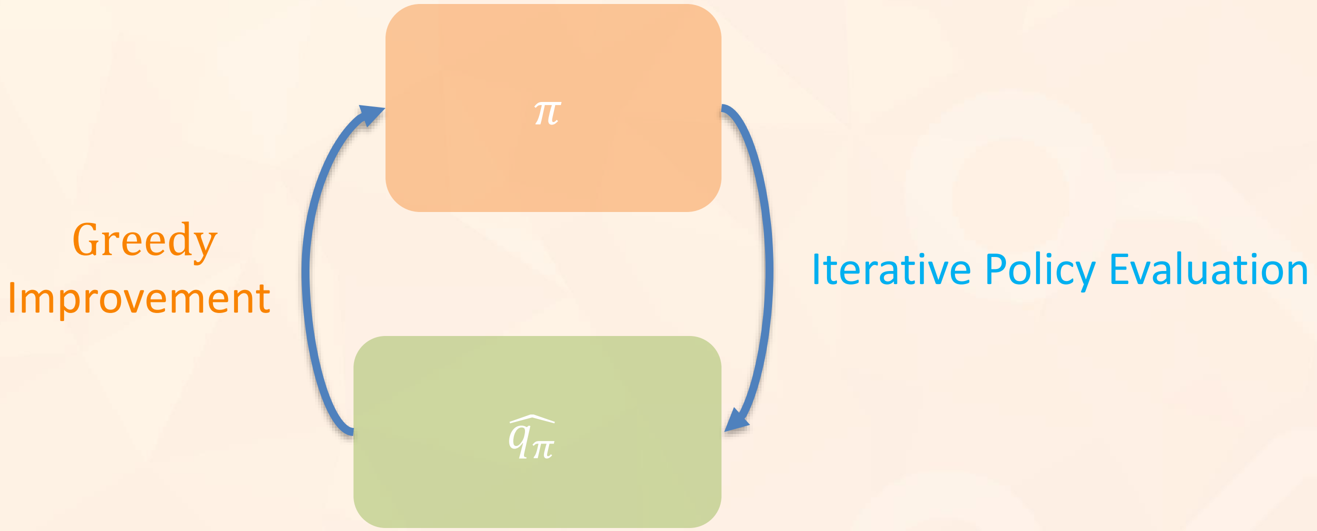
Policy improvement :

$$\pi(s) \coloneqq \underset{a}{\operatorname{argmax}}\, \hat{q}_\pi(s,a) \approx \underset{a}{\operatorname{argmax}}\, q_\pi(s,a)$$

$$\pi \xrightarrow[\text{Iterative Policy Evaluation}]{} \widehat{q_\pi} \xrightarrow[\text{Greedy Improvement}]{} \pi'$$

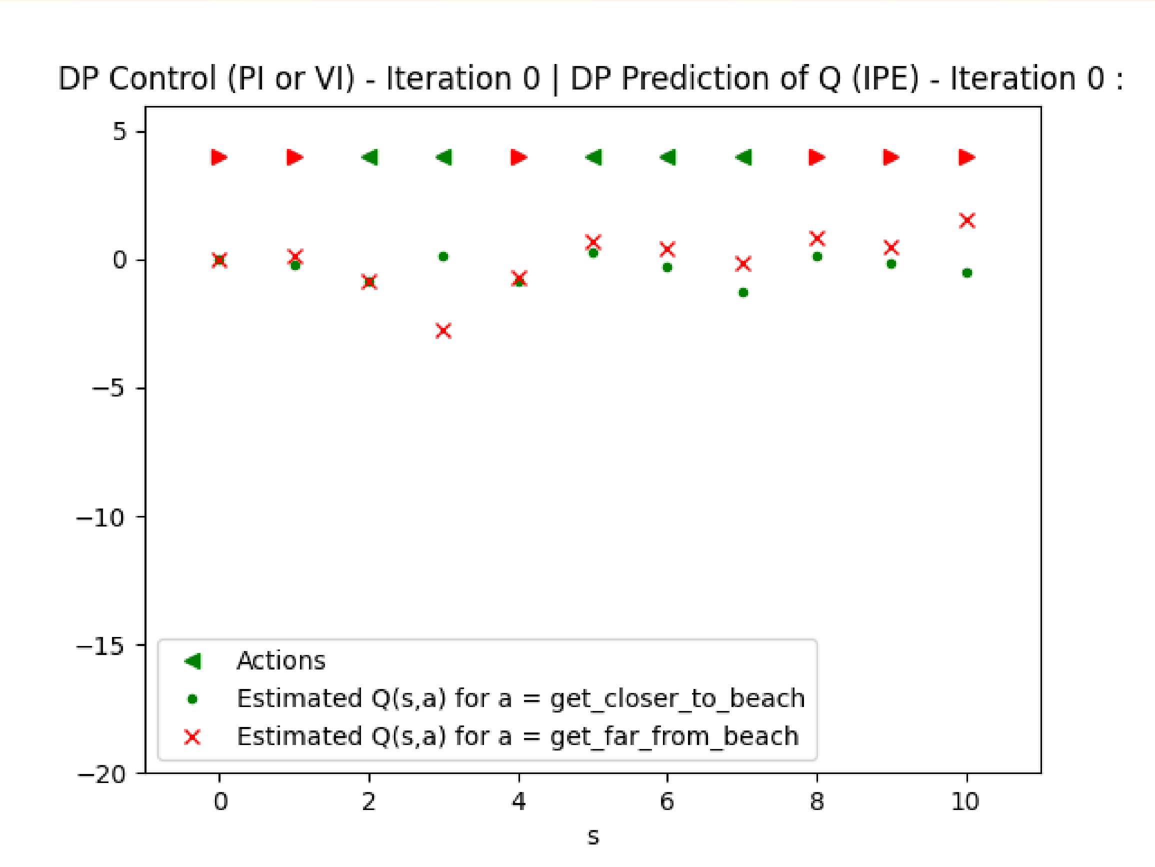
Policy Iteration

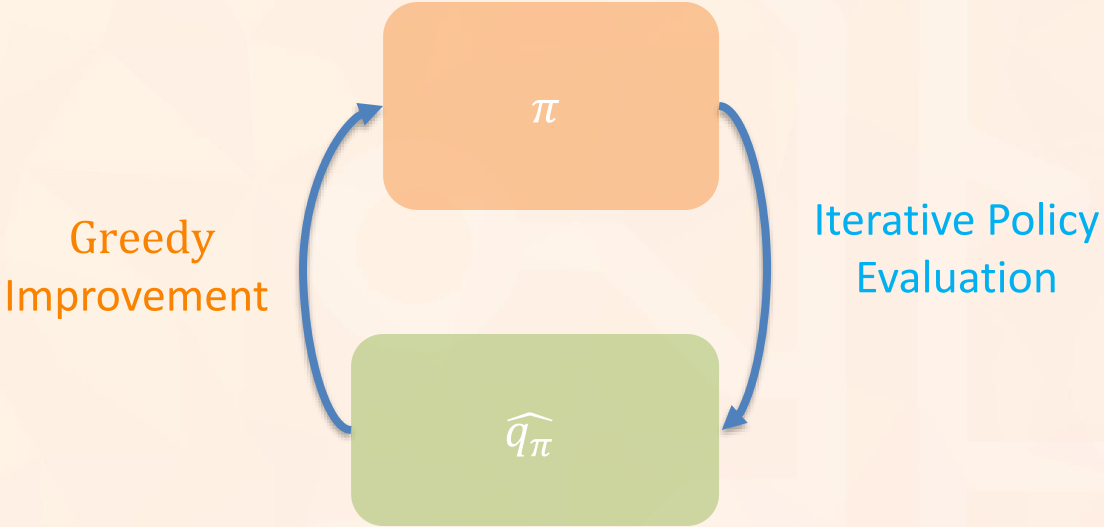
Algorithm : Policy Iteration
Algorithm used in Dynamic Programming for the Control Problem



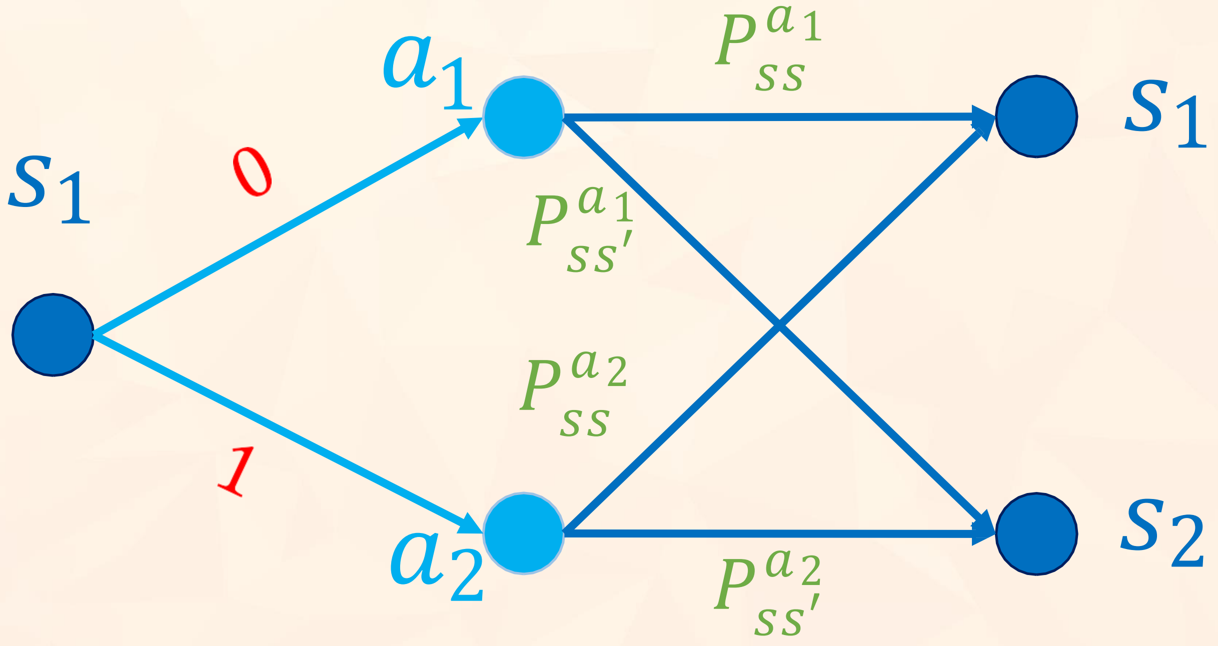
Policy Iteration : Results

Policy Iteration ($n_{\text{iter}} = 5$ evaluation iteration)





Équations de Ford-Bellman optimales



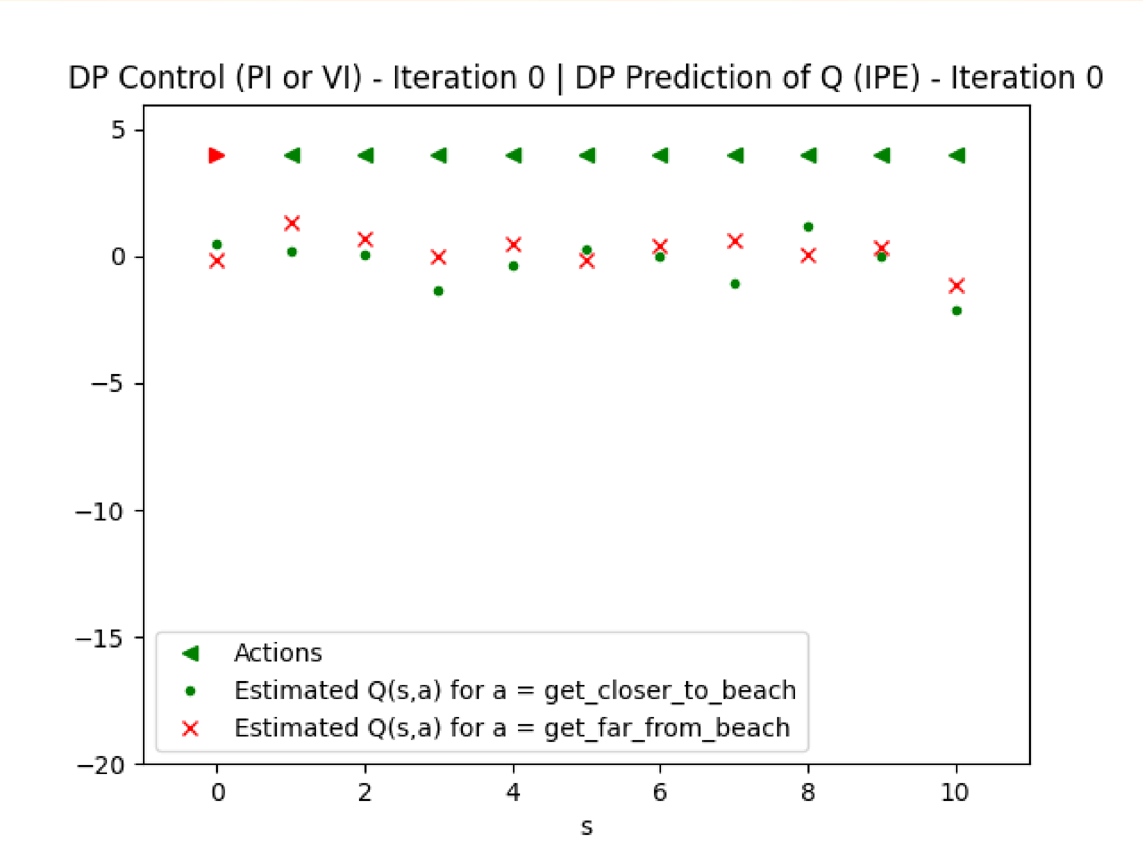
We evaluate not just any π but directly π^* the optimal policy.

$$v_\pi(s) = \max_{a \in A}\left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')\right)$$

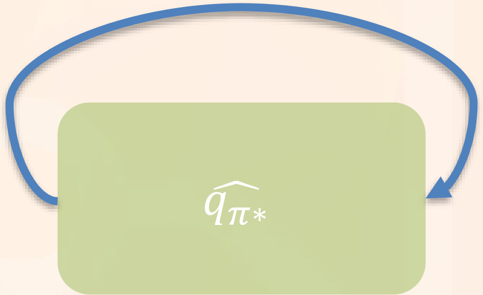
$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a' \in A}(q_\pi(s',a'))$$

Value Iteration

Algorithm: Value Iteration
Algorithm used in Dynamic Programming to solve the Control Problem.



Evaluation of π^* the optimal policy

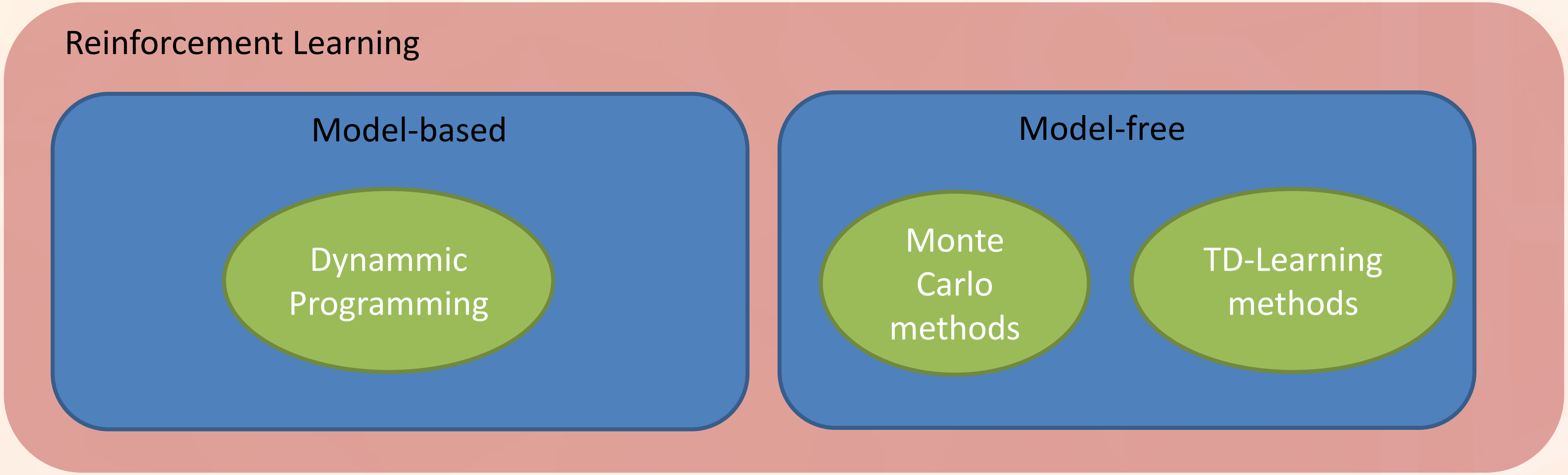


$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a' \in A}(q_\pi(s',a'))$$

Dynamic Programming : Conclusion

Avantages :
- Converge rapidement vers la solution optimale
- Fortes fondations mathématiques

Inconvénients :
- Adaptés à des petits espaces d'observations/actions discrets (finis) et non continus
- Model-Based : nécessite d'avoir accès au modèle



Monte Carlo Methods

MonteCarlo : Learn by interacting

Goal: Learn $\widehat{v_\pi}(s)$ from the observed G_t.

MonteCarlo (for 1 episode) :
- We play an episode τ where we observe S_τ states.
- $\forall\, s_t \in S_\tau, \widehat{v_\pi}(s_t) \leftarrow G_t$

MonteCarlo (for N episodes) :
- We play N episodes where we observe states S
- $\forall\, s \in S, \widehat{v_\pi}(s) \leftarrow \mathrm{mean}(\{G_t | S_t = s\})$

MonteCarlo for q (for N episodes):
- We play N episodes where we observe state-action couples (s, a)
- $\forall\, (s, a), \widehat{q_\pi}(s, a) \leftarrow \mathrm{mean}(\{G_t | S_t = s, A_t = a\})$

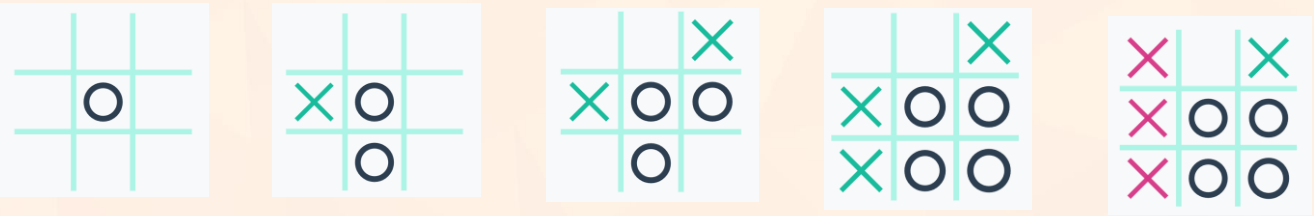
N = tradeoff time/variance

Avantages : intuitive, mathematically true : $\underbrace{E[G_t | S_t = s]}_{v_{MC}} = v_\pi(s)$, the Monte Carlo estimator v_{MC} is said to be non-biased
Inconvénients : terminal, high variance

MonteCarlo : Learn by interacting

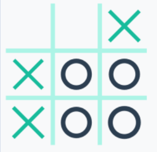
Environment: tic-tac-toe against a randomly playing opponent
Reward r = ±1 when winning/losing, 0 otherwise

End
$r = -1$



End
$r = +1$

Learning is terminal-required: you have to wait until the end of an episode to update v

s	$\widehat{v_\pi}(s)$
	$\frac{-1+1}{2}$
	-1
	$+1$
	$\frac{-1+1}{2}$
	$+1$

Remark : cumulative vs moving average

Cumulative average

$$\hat{X}_N = \frac{x_1 + x_2 + \cdots + x_N}{N}$$

Incremental formula:

$$\hat{X}_{N+1} = \frac{N}{N+1}\hat{X}_N + \frac{1}{N+1}x_{N+1}$$

- $\hat{X}_N$ tends to $E[X]$
- Suitable for stationary env. and policies
- All x_i weigh the same

Example for Monte Carlo :

At the end of each episode, for every s seen in t_s:

$$\widehat{v_\pi}(s) = \frac{N(s)}{N(s)+1}\widehat{v_\pi}(s) + \frac{1}{N(s)+1}G_{t_s}$$

Moving average

Incremental formula:

$$\hat{X}_{N+1} = (1-\alpha)\hat{X}_N + \alpha x_{N+1}$$

On notera : $\hat{X} \leftarrow x_i$

- $\hat{X}$ get closer to $E[X]$ permanently
- Suitable for non-stationary env. and policies
- Recent x_i weigh more

Example for Monte Carlo :

At the end of each episode, for every s seen in t_s:

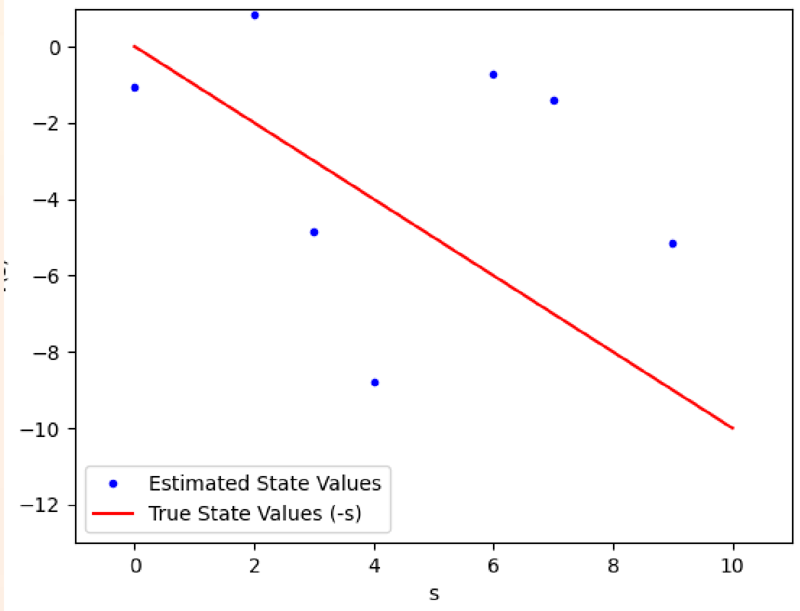
$$\widehat{v_\pi}(s) = 0{,}99\ \widehat{v_\pi}(s) + 0{,}01\ G_{t_s}$$

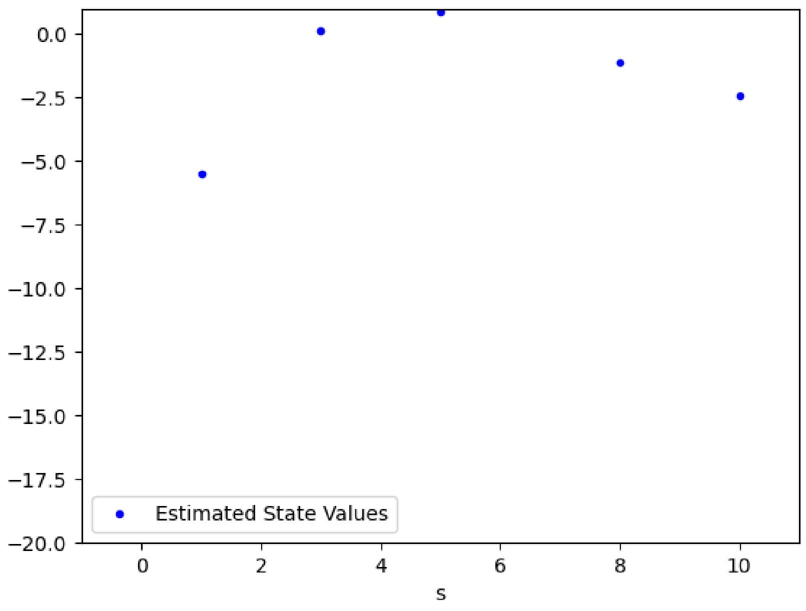
Monte Carlo : Results for $\hat{v}_\pi(s)$

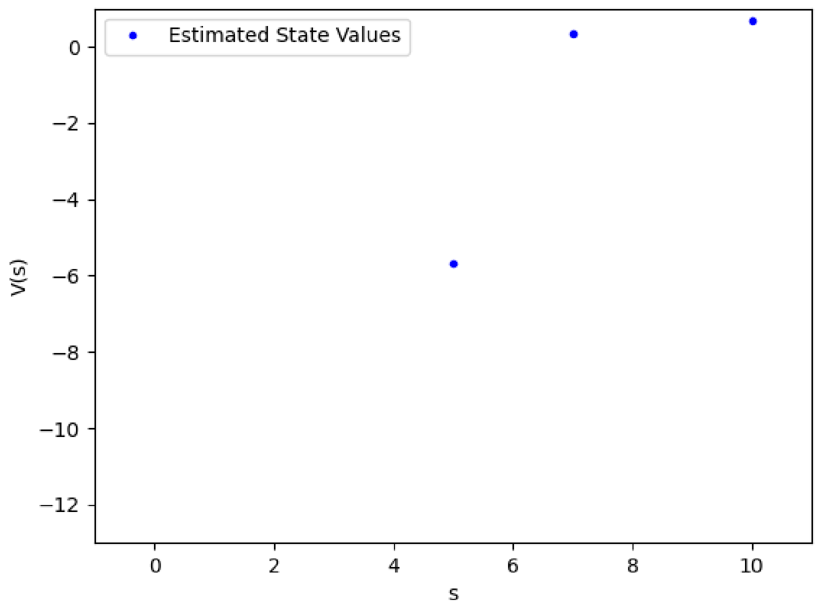
π = get closer

$$\pi = \begin{cases} \text{get closer,} & 80\% \\ \text{move away,} & 20\% \end{cases}$$

π = move away







Implementation notes: the $\widehat{v_\pi}(s)$ are randomly initialized and we use moving average to learn $v_\pi(s)$.

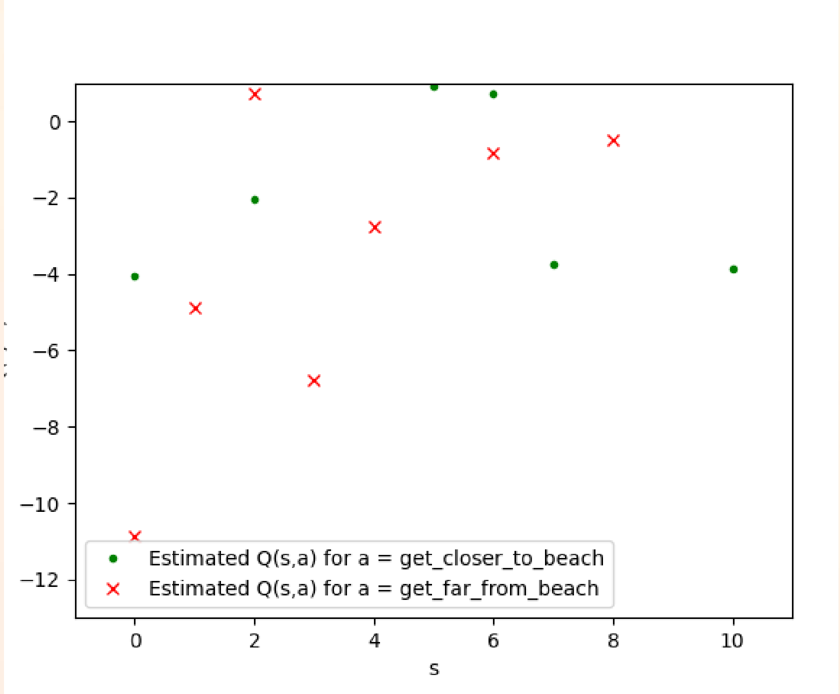
Exploration problem: for $\pi_{move\ away}$, we never see states close to the shore, so we cannot evaluate them

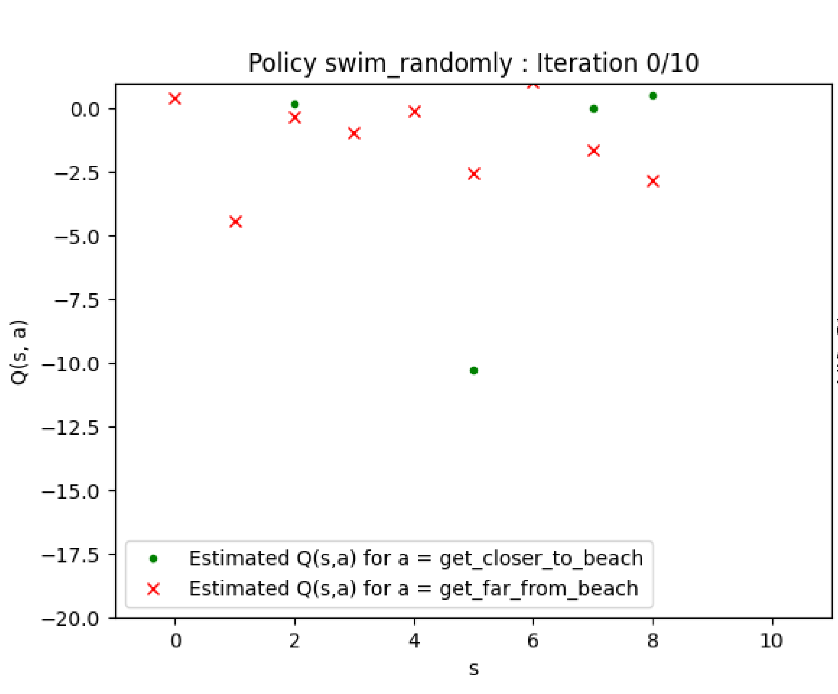
Monte Carlo : Results for $\hat{q}_\pi(s)$

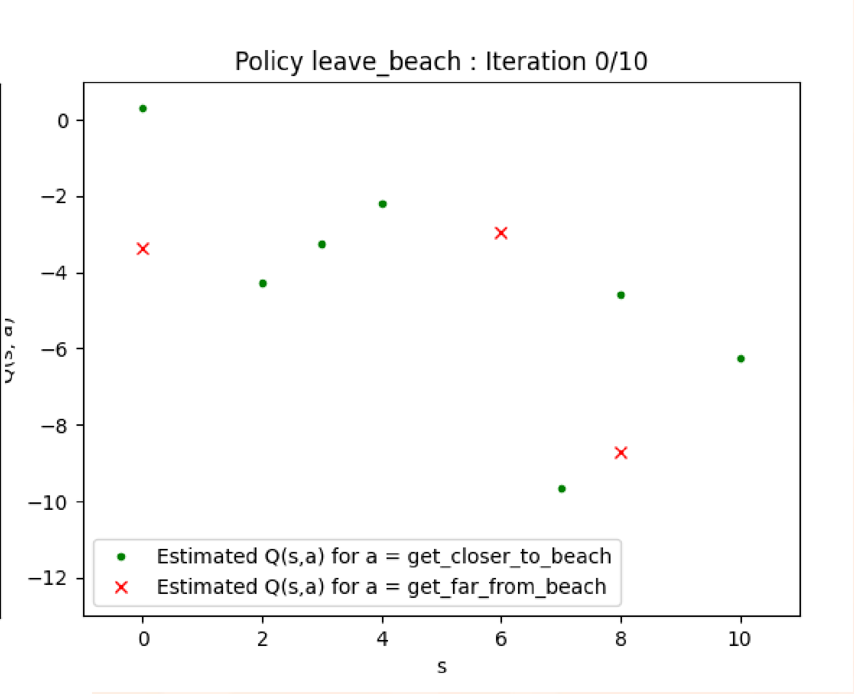
π = get closer

$$\pi = \begin{cases} \text{get closer,} & 80\% \\ \text{move away,} & 20\% \end{cases}$$

π = move away







Exploration problem: actions that are never taken, and states that are never reached while playing π are not evaluated

MonteCarlo does NOT evaluate deterministic policies well

Monte Carlo : Control Problem

Algorithm : Monte Carlo Control

π

Improvement of π

$$\pi(s) \coloneqq \underset{a}{\operatorname{argmax}}\, \hat{q}_\pi(s,a)$$

(greedy policy)

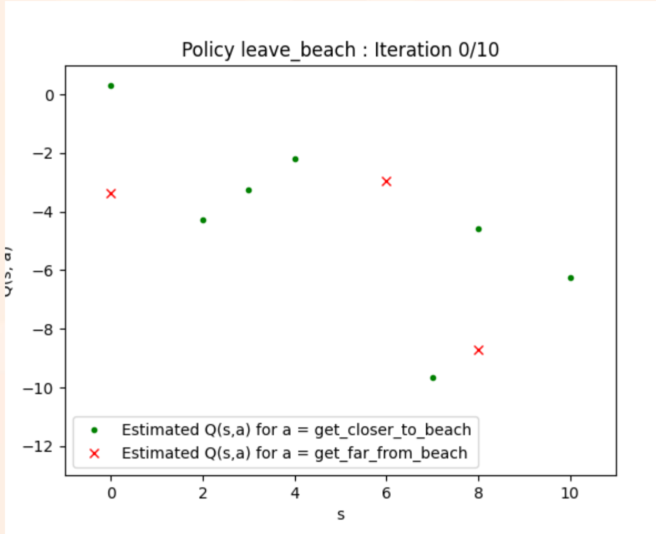
Evaluation of π by Monte Carlo Prediction

$$\widehat{q_\pi}(s,a) \leftarrow G_t | s, a$$

Duration = N episode ? Until convergence ?

$\widehat{q_\pi}$

Problem of the greedy policy: we have seen that Monte Carlo does not evaluate deterministic policies well for unchosen actions, because the algorithm needs experiments where these actions take place.



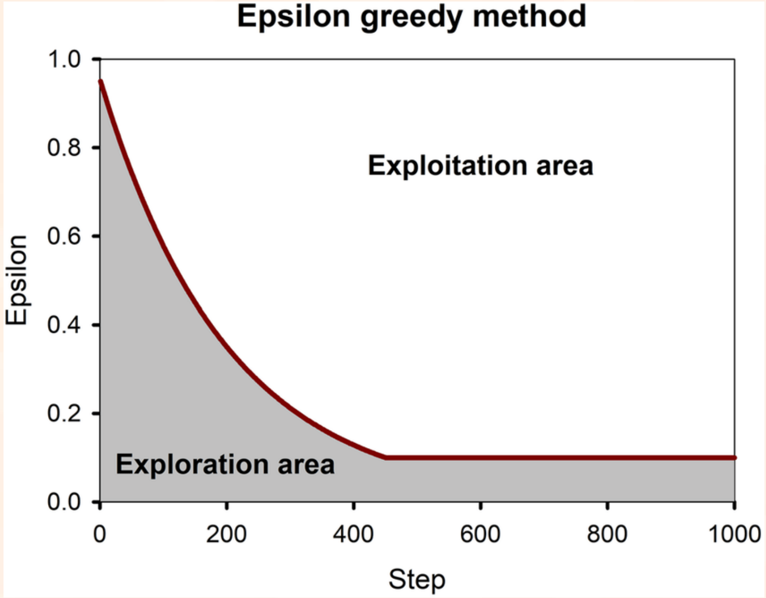
The Exploration vs. Exploitation tradeoff

Exploration problem: some actions are not visited very often and are therefore less well estimated.
Solution: use more exploratory policies

ε-greedy policy : $\pi(s) \coloneqq \begin{cases} \underset{a}{\operatorname{argmax}}\, \hat{q}_\pi(s,a) & \text{with probability } 1-\varepsilon \\ a \sim A & \text{with probability } \varepsilon \end{cases}$

Boltzmann policy : $\pi(a|s) \coloneqq \frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}}$

UCB policy : $\pi(s) \coloneqq \underset{a}{\operatorname{argmax}}(\underbrace{\hat{q}_\pi(s,a)}_{\text{exploitation}} + \underbrace{c\sqrt{\frac{\log(t)}{N(s,a)}}}_{\text{exploration}})$



Note: The exploitation/exploration tradeoff is an essential aspect of RL. It is widely studied in one of the fundamental problems of RL, the N-Bandit Problem.

Monte Carlo : Control Problem

Algorithm : Monte Carlo Control

π

Improvement of π

$\pi \coloneqq \epsilon\,\text{greedy}(\,\hat{q}_\pi)$

Evaluation of π by Monte Carlo Prediction

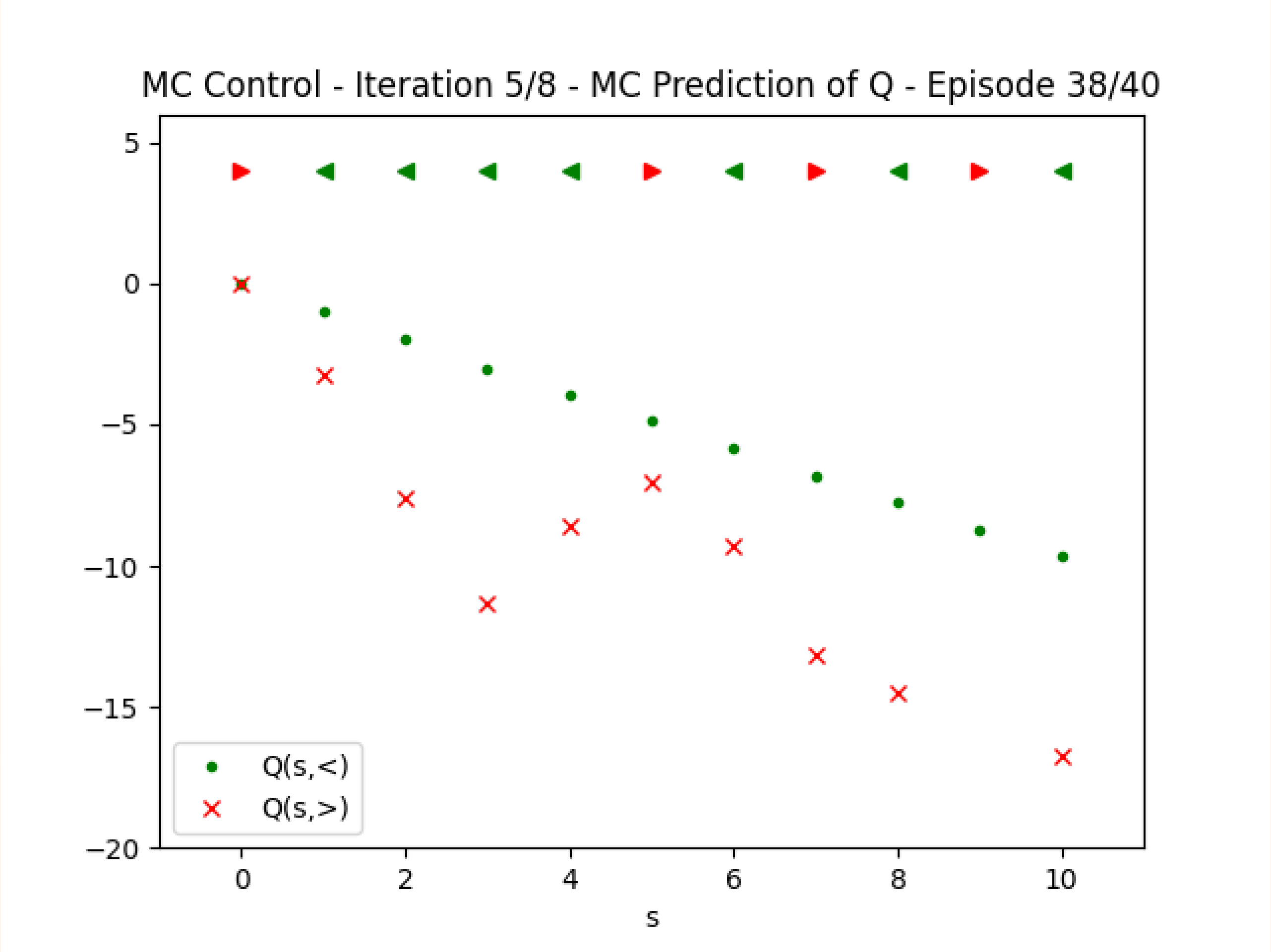
$\widehat{q_\pi}(s,a) \leftarrow G_t | s, a$

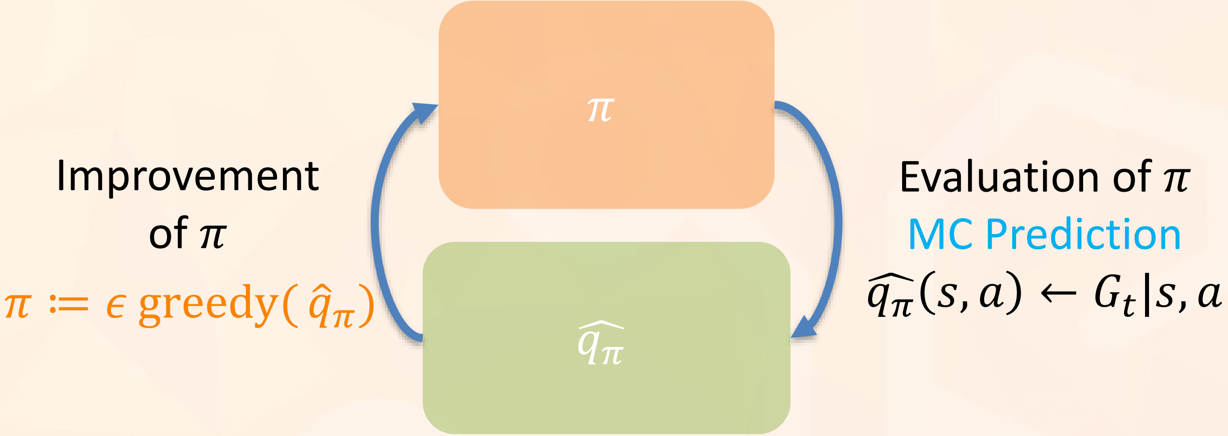
Duration = N episode ?

Until convergence ?

$\widehat{q_\pi}$

Monte Carlo Control : Results





Implementation Notes: The q values are initialized randomly at the beginning, then at each evaluation phase (Prediction) they are initialized like the previous Q values.

We explore with a $\boldsymbol{\epsilon}$ greedy policy with ϵ constant at 0,1.

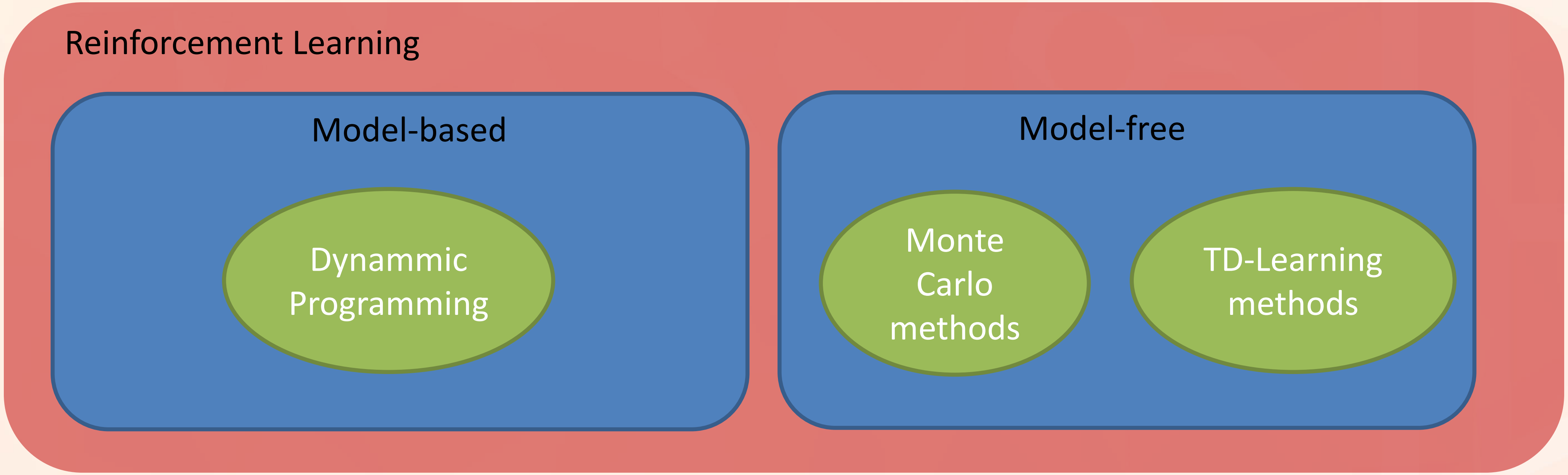
Monte Carlo : Conclusion

Advantages:

- Can learn from real experiences so suitable for real problems

Disadvantages:

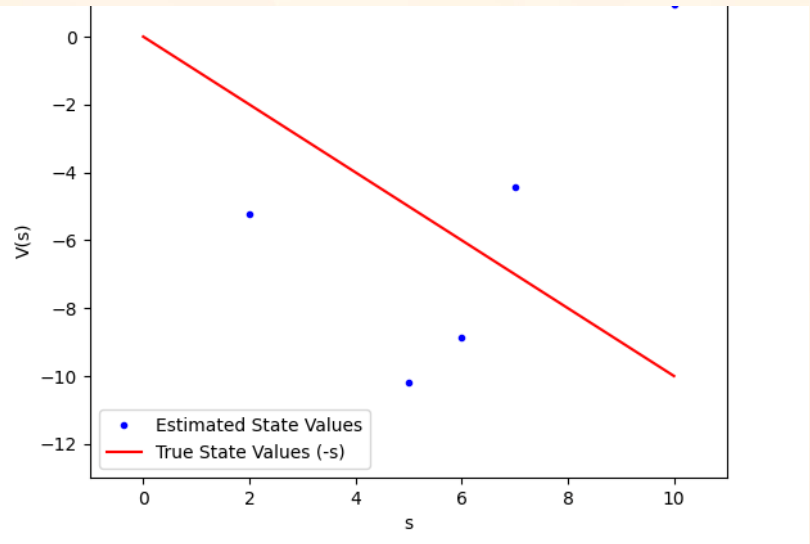
- Terminal required : One has to wait for the end of an episode to estimate the values
- Variance of the estimator high when T becomes large, which makes converging harder



TD Learning methods

To TD Learning : bias-variance tradeoff

Example of algo with bias :



What is the bias of an estimator $\hat{v}$?

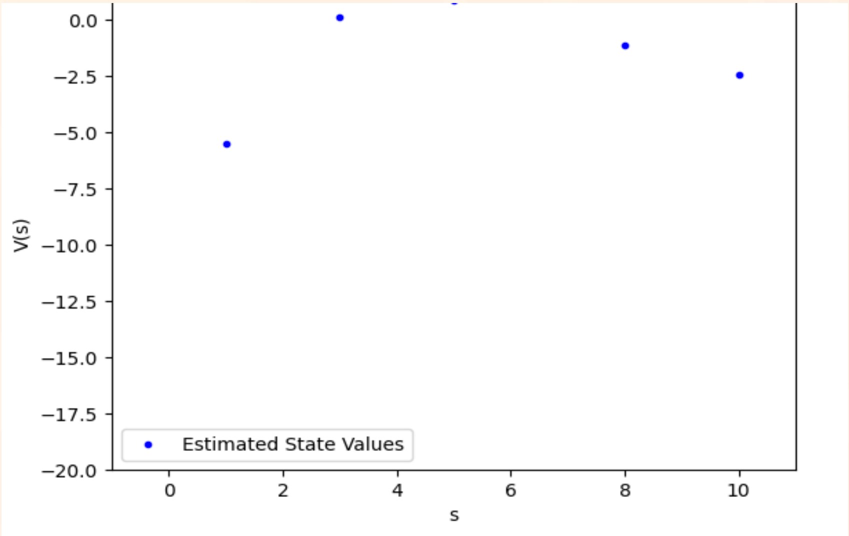
It is the systematic error $|E[\hat{v}] - v_\pi|$.

If the bias is non-zero, we learn towards a bad value.

$$v_{\text{MonteCarlo}} = \quad R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ \ldots + \gamma^{T-t} R_T$$

- Unbiaised: $E\left[v_{\text{MonteCarlo}}\right] = v_\pi(s_t)$
- High variance because s_t and R_T are highly correlated

Example of algo with high variance :



Qu'est ce que la variance ?

This is a typical mistake $(\hat{v} - E[\hat{v}])^2$ obtained for 1 estimate.

If the variance is high, it will take a lot of sampling to get a good estimate.

Sources of variance: transitions/reward/stochastic agent policy

TD Learning

As in Monte Carlo we learn by experience, but here by bootstrapping we do not wait for the end of the episode:

$\widehat{v_\pi}(s_t) \leftarrow r_t + \gamma \widehat{v_\pi}(s_{t+1})$ TD(0)

$\widehat{v_\pi}(s_t) \leftarrow r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ \ldots + \gamma^{T-t} r_T = \ \mathrm{G_t}$ MonteCarlo

$R_t \ + \ \gamma v_\pi(S_{t+1})$ is an unbiased and low variance estimator of $v_\pi(s_t) = E_\pi[G_t | S_t = s_t]$.

$\underbrace{R_t}_{\text{unbiased term, allowing to learn}} \ + \ \underbrace{\gamma \hat{v}_\pi(S_{t+1})}_{\text{biased term, estimates the rewards suite}}$ is an estimator with bias but low variance.

TD Learning : Prediction Problem

Implementation : $\widehat{v_\pi}(s_t) = \widehat{v_\pi}(s_t) + \alpha\underbrace{(r_t + \gamma\widehat{v_\pi}(s_{t+1}) - \widehat{v_\pi}(s))}_{}$

δ_t =Temporal Difference (TD)

with α = 0.01 for example, the Learning Rate

Algorithm : TD(0) (for the Prediction Problem of estimating v)

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        $A \leftarrow$ action given by π for S
        Take action A; observe reward, R, and next state, S'
        $V(S) \leftarrow V(S) + \alpha\left[R + \gamma V(S') - V(S)\right]$
        $S \leftarrow S'$
    until S is terminal

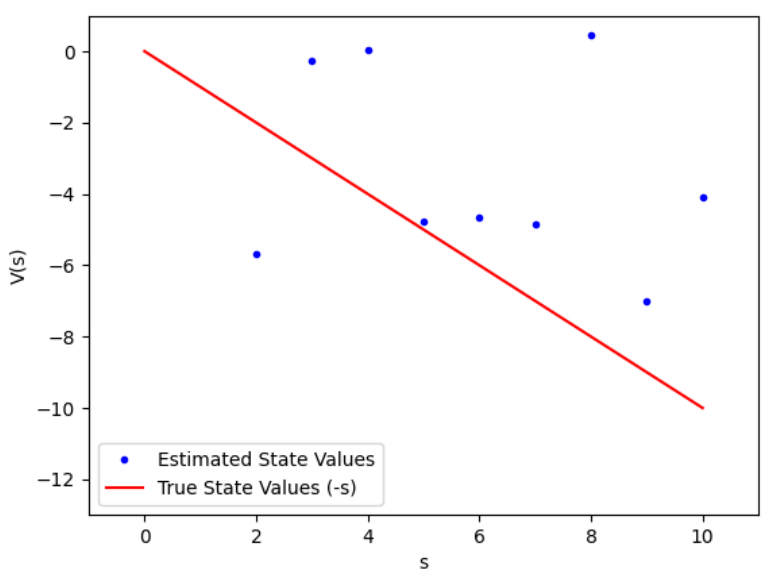
Learning is non necessarily terminal: you can learn while you play!

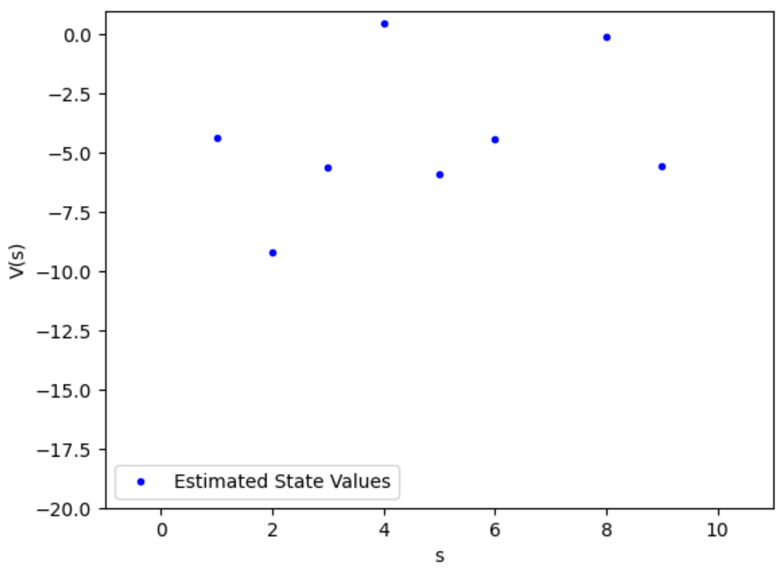
TD(0) : Results for $\hat{v}_\pi(s)$

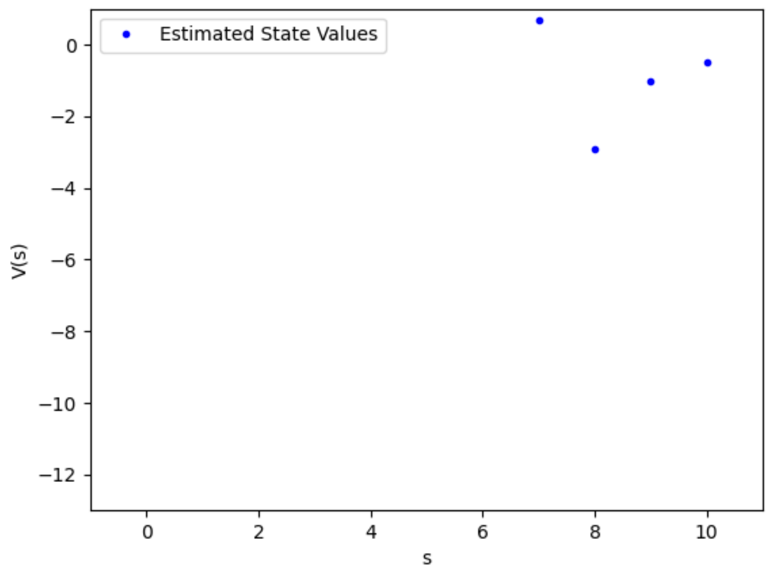
π = get closer

$$\pi = \begin{cases} \text{get closer,} & 80\% \\ \text{move away,} & 20\% \end{cases}$$

π = move away







Exploration problem: for $\pi_{move\ away}$, we never see the states close to the shore, so we can't evaluate them

TD Learning : Control Problem

$\widehat{v_\pi}(s_t) \leftarrow r_t + \gamma \widehat{v_\pi}(s_{t+1})$ TD(0) for estimate $E_\pi[G_t | S_t = s_t]$

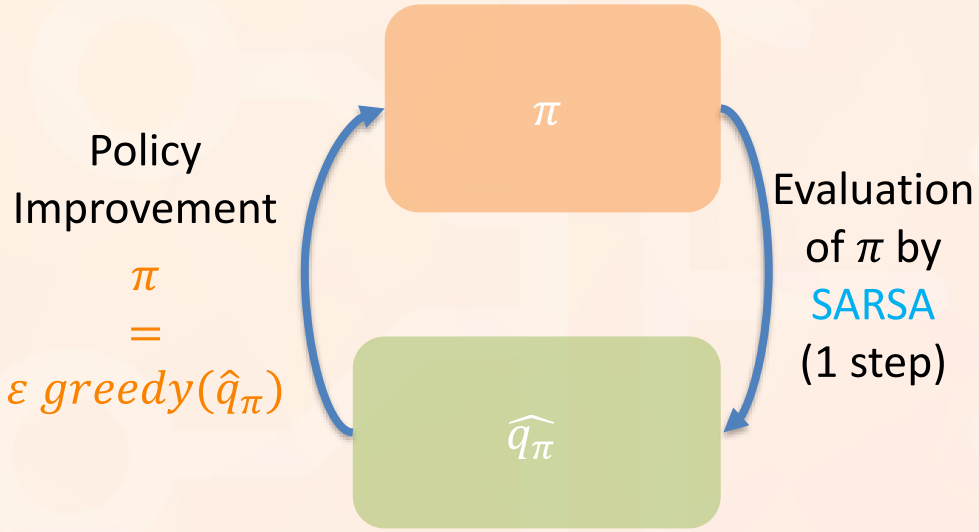
$\widehat{q_\pi}(s_t, a_t) \leftarrow r_t + \gamma \widehat{q_\pi}(s_{t+1}, a_{t+1})$ SARSA $E_\pi[G_t | S_t = s_t, A_t = a_t]$

$\widehat{q_\pi}(s_t, a_t) \leftarrow r_t + \gamma \sum_{a'} \pi(a' | s_{t+1}) \widehat{q_\pi}(s_{t+1}, a')$ SARSA-Expected $E_\pi[G_t | S_t = s_t, A_t = a_t]$

Algorithm : SARSA Control

TD algorithm for the Control Problem

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (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

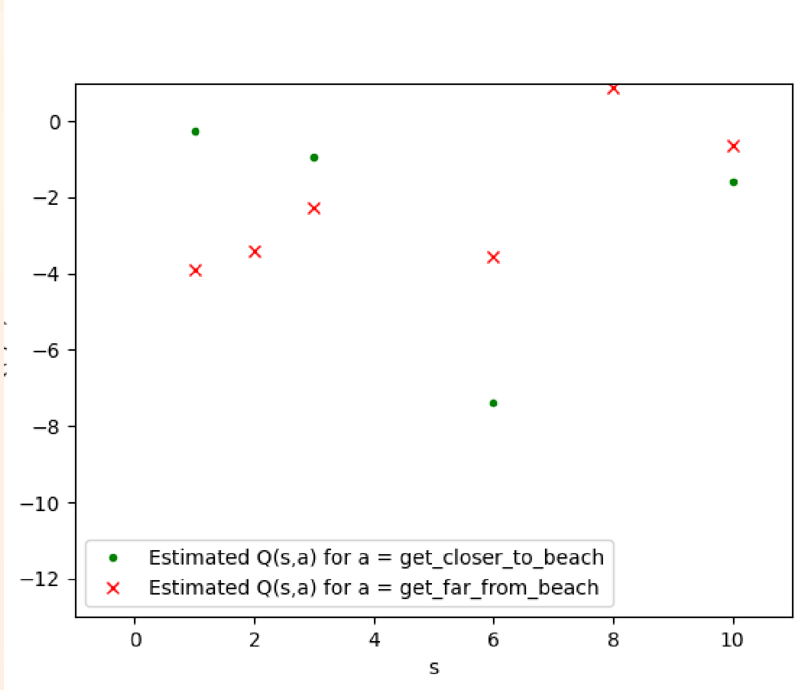


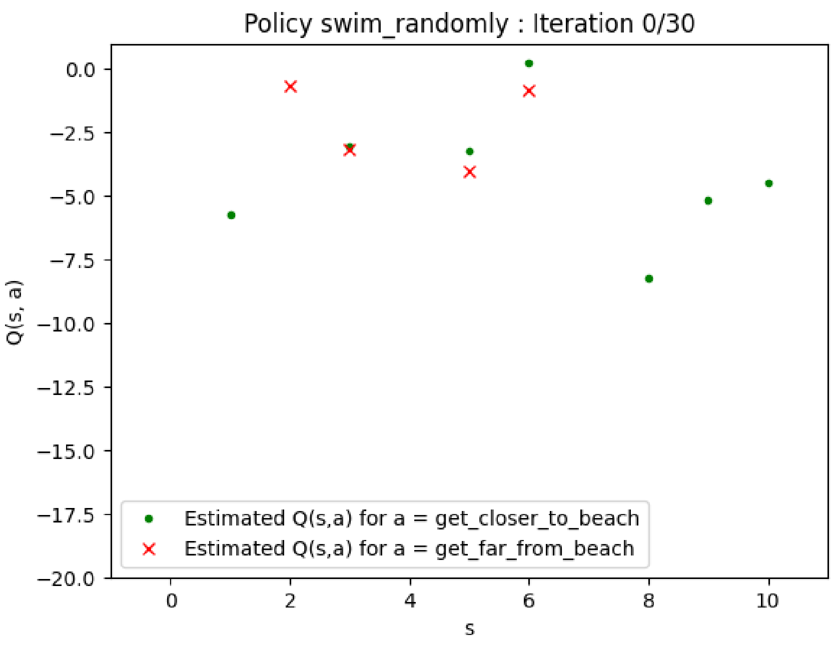
SARSA: Results for $\hat{q}_\pi(s)$

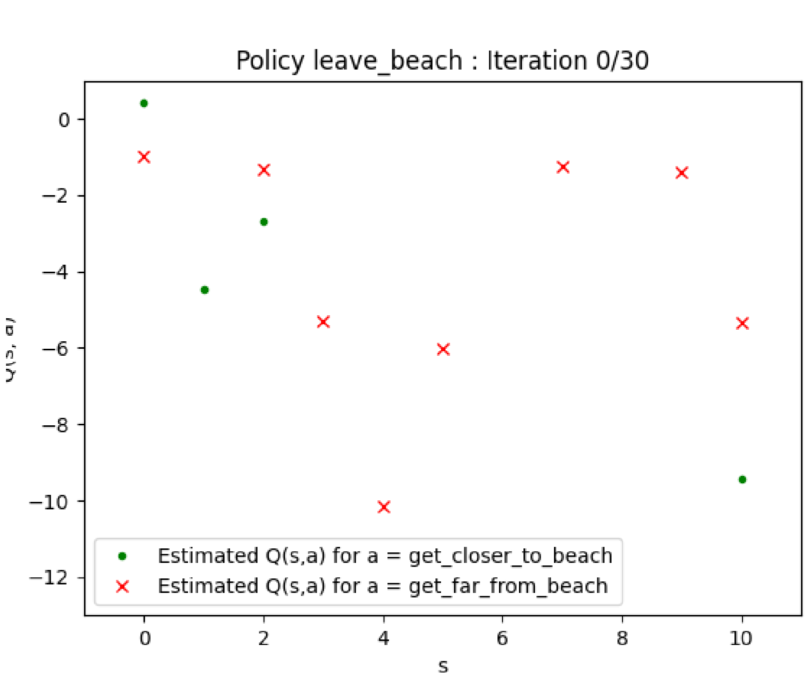
π = get closer

$$\pi = \begin{cases} \text{get closer,} & 80\% \\ \text{move away,} & 20\% \end{cases}$$

π = move away

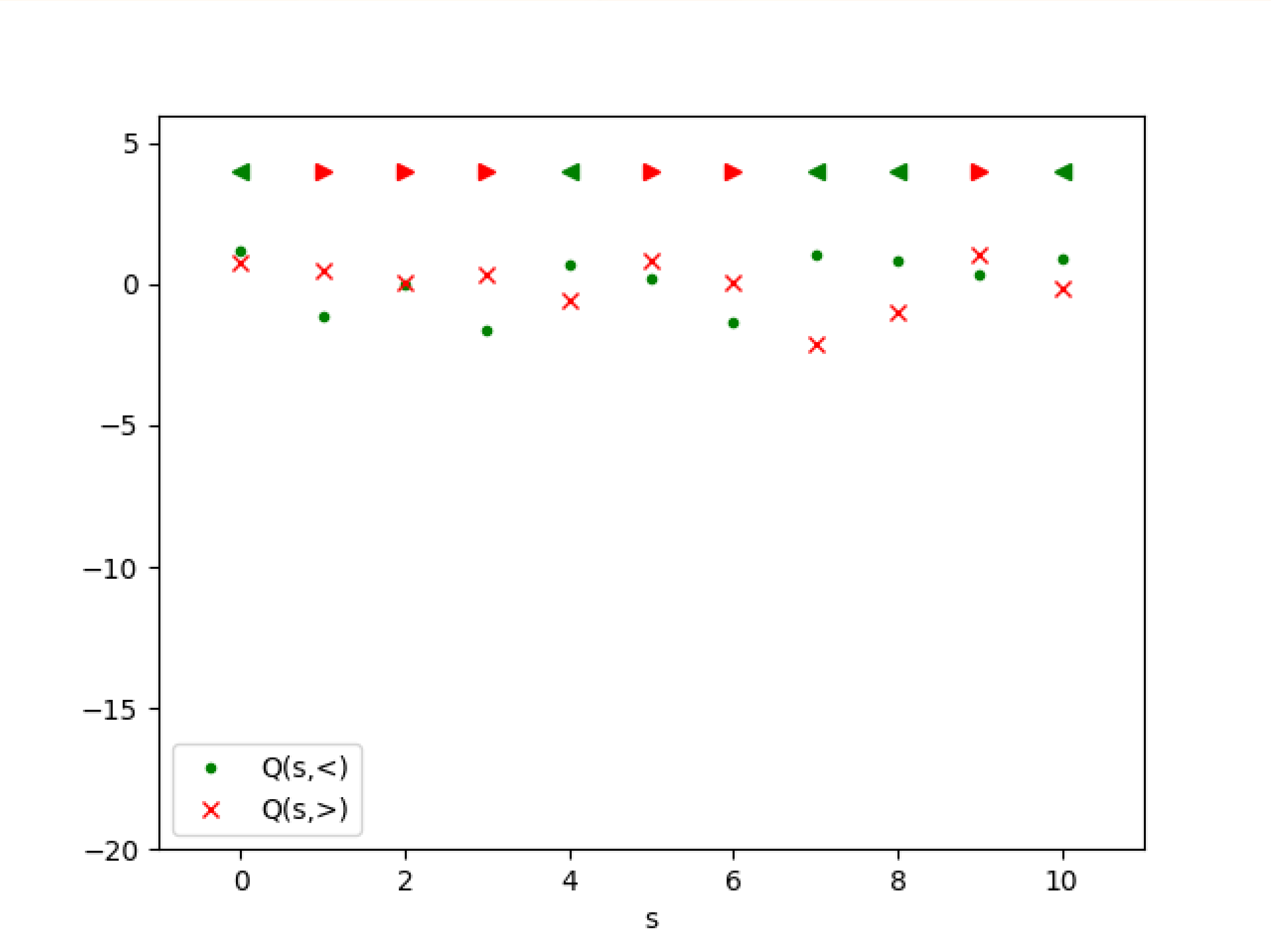






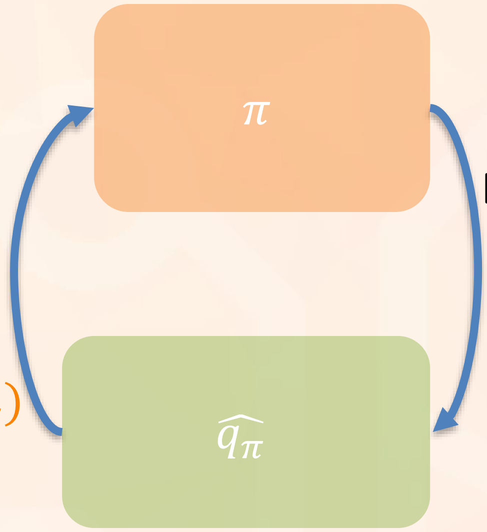
Exploration problem: actions that are never taken, and states that are never reached while playing π are not evaluated

SARSA Control : Results



Policy Improvement

$\pi = \varepsilon\ greedy(\hat{q}_\pi)$



Evaluation of π by SARSA (1 step)

Implementation notes: The q values are randomly initialized at the beginning.We explore with a $\boldsymbol{\epsilon}$ greedy policy with ϵ constant at 0,1.

n-step TD Learning

Rather than bootstrapping after one step, we will bootstrap after n steps

$\widehat{v_\pi}(s_t) \leftarrow r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n \widehat{v_\pi}(s_{t+n})$ n-step TD

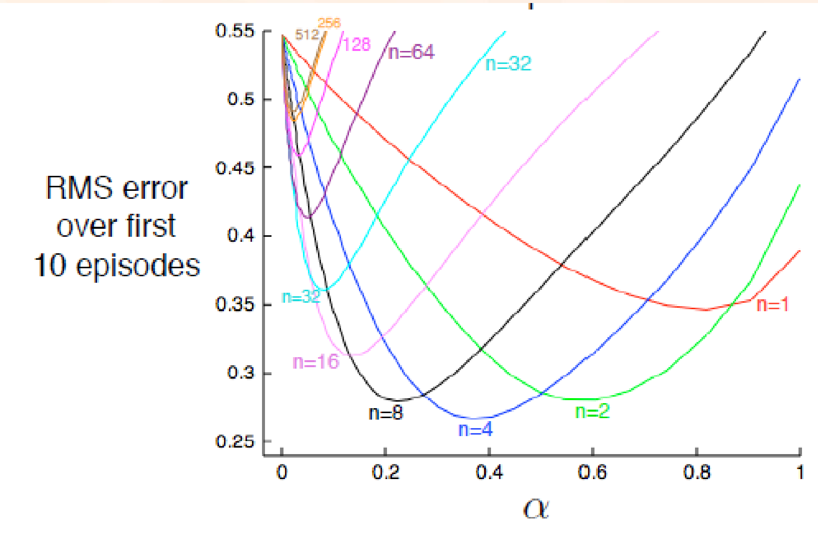
$\widehat{q_\pi}(s_t, a_t) \leftarrow r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n \widehat{q_\pi}(s_{t+n}, a_{t+n})$ n-step SARSA

$\widehat{q_\pi}(s_t, a_t) \leftarrow r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t} r_T$ MonteCarlo $(n \to +\infty)$

Increasing n has the effect of:

- Increase the variance
- Impose a longer wait (n steps) before learning
- Decrease the bias

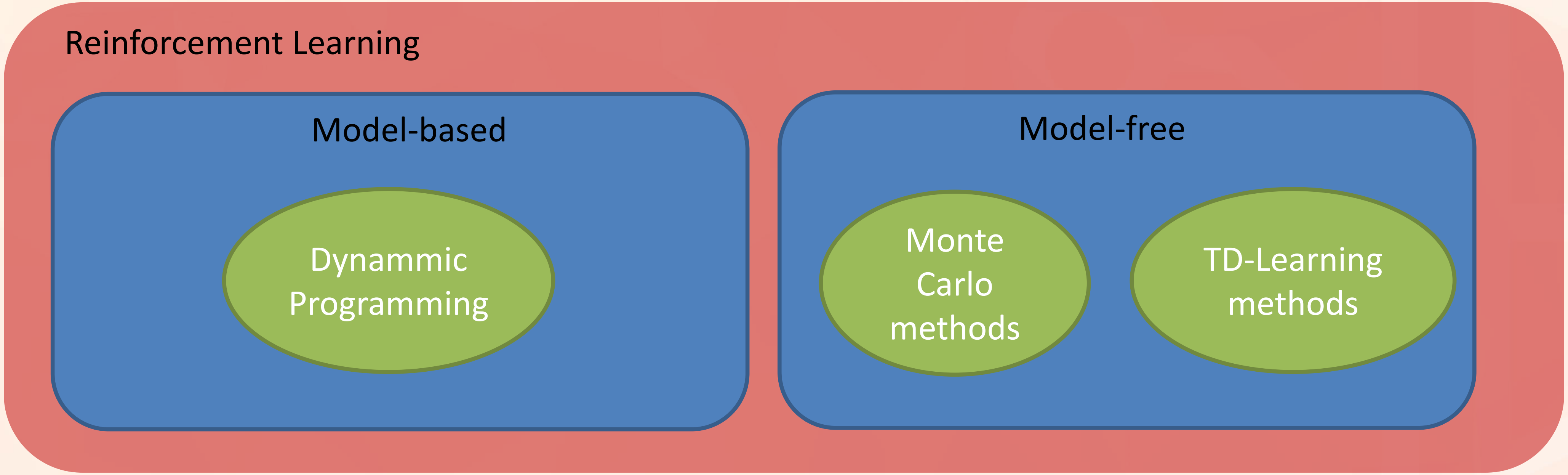
The optimal n hyperparameter depends on env. and other hyperparameters.



TD Learning: Conclusion

Advantages:
- No need to wait until the end of the episode to learn
- Low variance

Disadvantages:
- Biased because $\widehat{v_\pi}(s_t) \neq v_\pi(s_t)$



Off Policy and Q-Learning

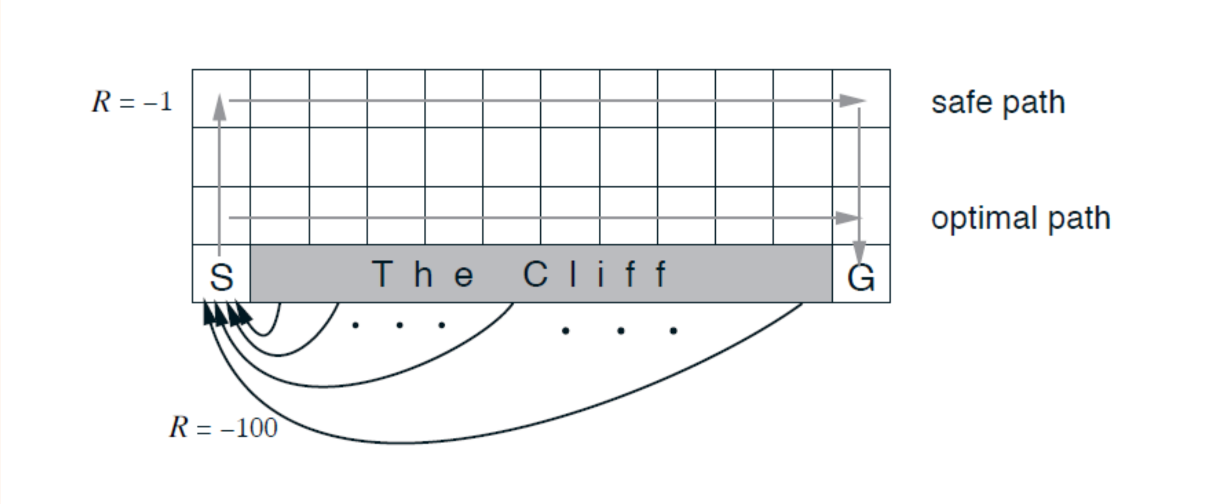
The limits of exploratory policies

Need to explore : $\pi = \varepsilon\ greedy$

Implemented in this way, Monte Carlo Control and SARSA Control then result in $\pi_{target} = \varepsilon\ greedy$ trained to the best explorative policy

However, in some environments, the best exploratory policy is too conservative and is far inferior to the best policy:

Environment : The Cliff



Solution: dissociate target policy π (to be evaluated and improved) from behavioral policy μ (who to play the episodes with):

$$\begin{cases} \pi = greedy \\ \mu = \text{exploration } (\varepsilon\ greedy, UCB, \dots) \end{cases}$$

Using a behavioral policy μ different from the policy one trains π constitutes Off Policy learning.

Off Policy

Def: Off Policy = use a different μ behavior policy than your policy to evaluate and optimize π.

Exemples :
$$\begin{cases} \pi = greedy(\widehat{q_\pi}),\ \pi_{quelconque} \\ \mu = \underbrace{\epsilon\ greedy(\widehat{q_\pi}),\ random,}_{\text{Allows exploration}}\ \underbrace{\pi_{old},\quad \pi_{other\ agent}}_{\text{Sample efficiency}} \end{cases}$$

The RL Off Policy algorithms, of the form $\widehat{q_\pi}(s,a) \leftarrow X$, i.e. those who verify :

$$E_\mu[X] = q_\pi(s,a)$$

Off Policy ?

$\widehat{v_\pi}(s_t) \leftarrow r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ \ldots + \gamma^{T-t} r_T$	MC	for estimating $E_\pi[G_t \mid S_t = s_t]$
$\widehat{v_\pi}(s_t) \leftarrow r_t + \gamma \widehat{v_\pi}(s_{t+1})$	TD(0)	$E_\pi[G_t \mid S_t = s_t]$
$\widehat{q_\pi}(s_t, a_t) \leftarrow r_t + \gamma \widehat{q_\pi}(s_{t+1}, a_{t+1})$	SARSA	$E_\pi[G_t \mid S_t = s_t, A_t = a_t]$
$\widehat{q_\pi}(s_t, a_t) \leftarrow r_t + \gamma \sum_{a'} \pi(a' \mid s_{t+1}) \widehat{q_\pi}(s_{t+1}, a')$	SARSA-Expected	$E_\pi[G_t \mid S_t = s_t, A_t = a_t]$

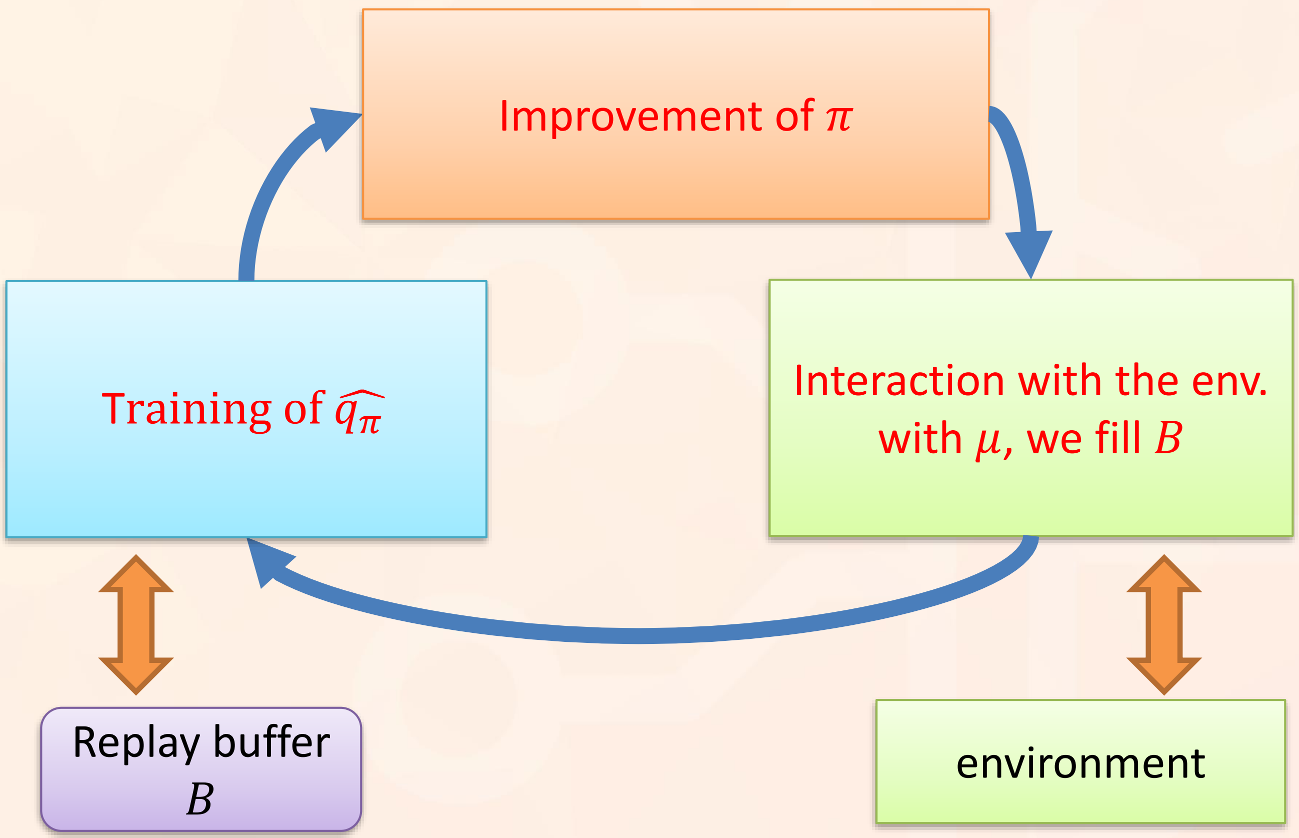
Experience Replay : learn from past experiences

The Off Policy allows you to reuse transitions that were sampled with old policies.

Experience Replay : We are not going to learn only once from each transition (s_t, a_t, r_t, s_{t+1}) : we'll store them in a B memory called *replay buffer.*

Interests :

- Sample efficiency : we use each transition drawn several times
- Decorrelation : Transitions taken from B are decorellated
- Parallelization : we will be able to train at the same time as we play (in a parallel way)
- Avoid catastrophic forgetting (i.e. forgetting information from older transitions)



Q Learning

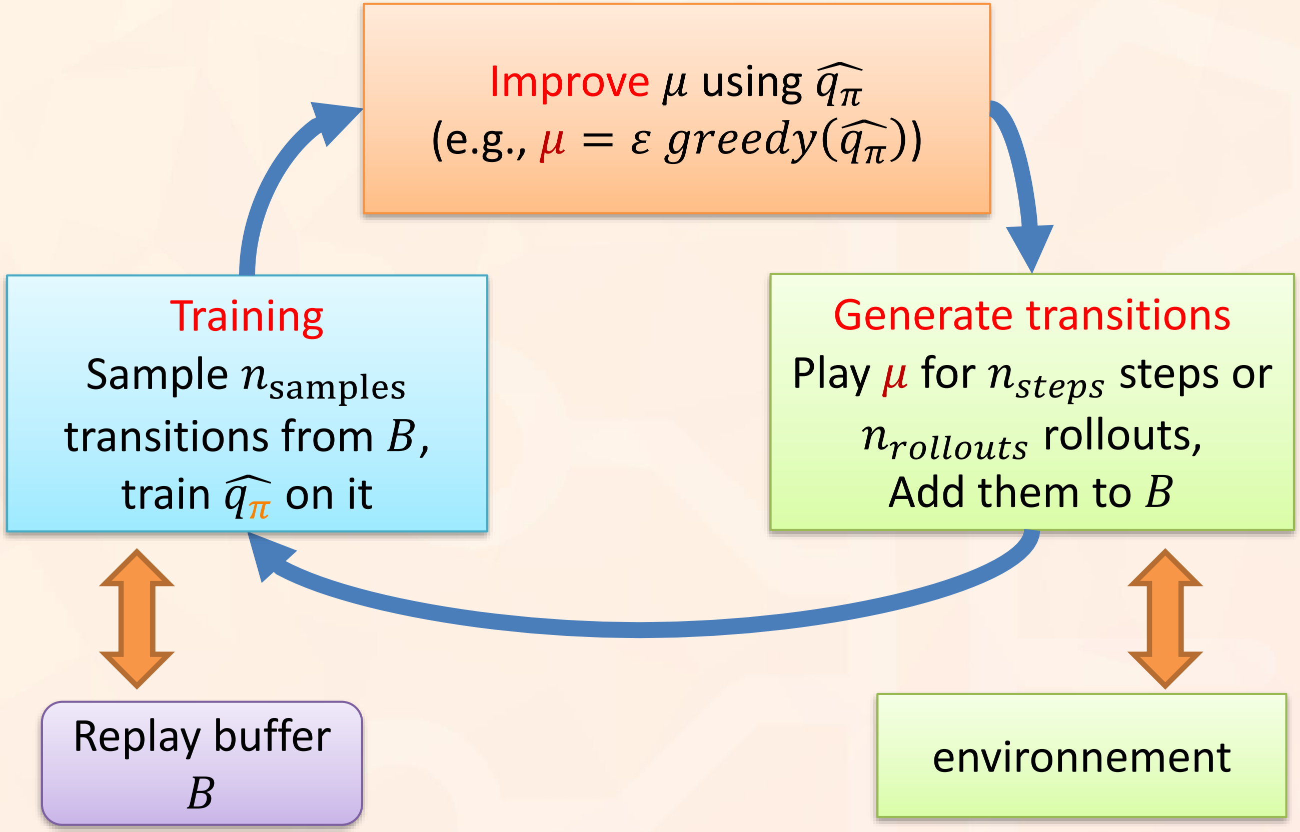
In the case of $\pi = greedy(\widehat{q_\pi})$, SARSA-Expected correspond to :

$$\widehat{q_\pi} \leftarrow r_t + \gamma \max_{a'} \widehat{q_\pi}(s_{t+1}, a') \qquad \text{Q Learning}$$

This algorithm, which has the advantage of being Off Policy and of low variance, is known as Q Learning.

$$\begin{cases} \pi = greedy(\widehat{q_\pi}) \\ \mu = \epsilon\ greedy(\widehat{q_\pi})\ \text{(old)} \end{cases}$$

Q Learning loop :



Note: the Q Learning equation can be seen as an application of the optimal Bellman equation.

Deep Reinforcement Learning

What to do when the spaces for observation (and action) are too large?

Tabular case:

s	0	1	…	$n_{state} - 1$
$\widehat{v_\pi}(s)$	0,00	-0,98	…	-7,96

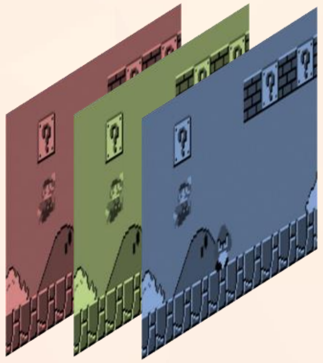
Learning :

$$\widehat{v_\pi}(s) \coloneqq \widehat{v_\pi}(s) + \alpha(X - \widehat{v_\pi}(s))$$

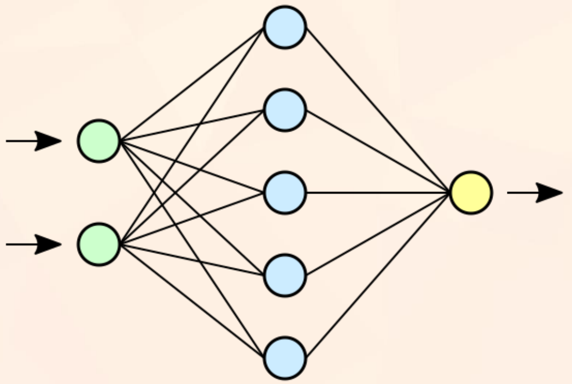
Deep RL :

Learning :

State s



Neural Network
Weights : φ



Estimated
State Value

$\widehat{v_\pi^\varphi}(s)$

$$\varphi \coloneqq \varphi - \alpha \nabla_\varphi \left(\mathrm{Loss}\left(\widehat{v_\pi^\varphi}(s), X\right)\right)$$

With φ parameters of $\widehat{v_\pi^\varphi}(s)$

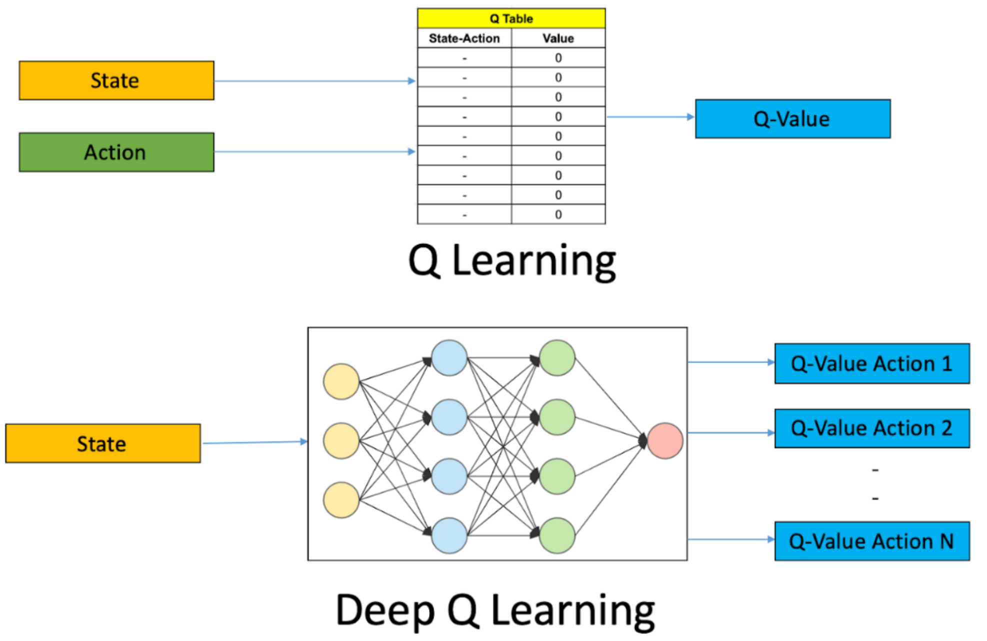
Deep Reinforcment Learning is powerful but poorly understood

Rather than seeing an image as a state among 256^{3HW} we see it as a vector in $[0; 255]^{3HW}$

Combining all this : Deep Q Network (DQN)

Algorithm 1 Deep Q-learning with Experience Replay

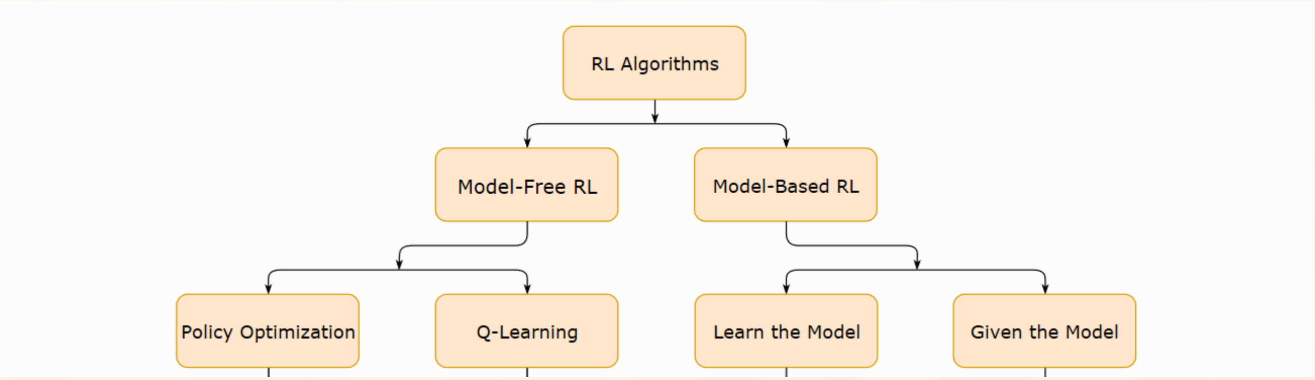
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for



[Playing Atari with Deep Reinforcement Learning]

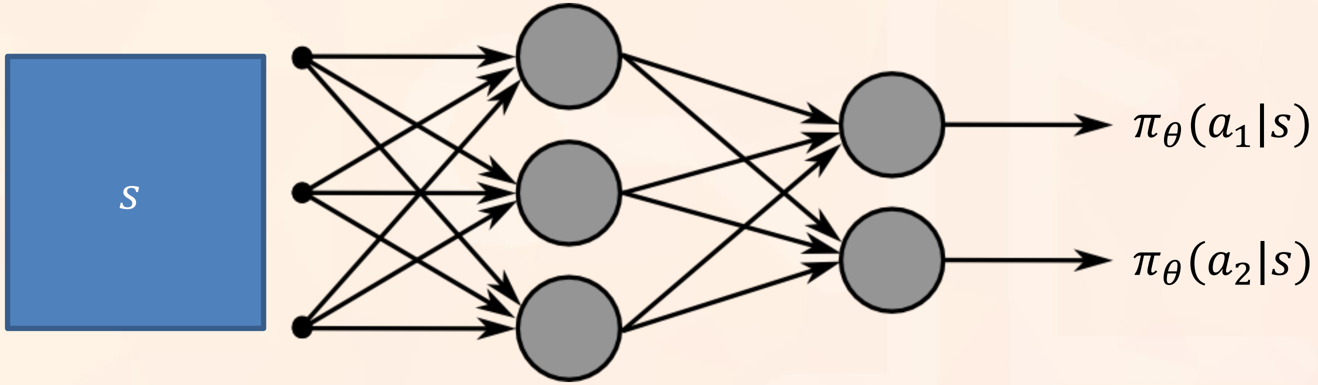
Policy-based RL

Policy Gradients



Parameterized policy: π_θ:

$$\pi_\theta : S \to [0,1]^{n_{actions}}$$
$$s \to (\pi_\theta(a_k|s))_{1 \le i \le n}$$



Goal: Define an objective function $J(\theta)$ differentiable with respect to θ to make a gradient climb :

$$\theta \coloneqq \theta + \alpha \nabla_\theta J(\theta)$$

Remark : rather than a discrete distribution of action in output we can have a continuous distribution $\pi_\theta(s) = (m, \sigma)$
A deterministic policy can also be used $\pi_\theta(s) = a$

Policy Gradients : theory

Objective function:

$$J(\theta) = E_{\pi_\theta}[G_0] = \int_\tau G_0(\tau)\rho_{\pi_\theta}(\tau)d\tau$$

$$E[X] = \int_\omega X(\omega)\rho\,(\omega)d\omega \quad X \to (\Omega, A, \rho)$$

Avec $\rho_{\pi_\theta}(\tau) = C_\tau \prod_{t=0}^{T} \pi_\theta(a_t\,|s_t\,)$

Gradient calculation:

$$\nabla \ln(u) = \frac{\nabla u}{u}$$

$$\nabla_\theta J\,(\theta) \;=\; \nabla_\theta \int_\tau G_0(\tau)\rho_{\pi_\theta}(\tau)d\tau = \int_\tau G_0(\tau)\nabla_\theta\rho_{\pi_\theta}(\tau)d\tau = \int_\tau G_0(\tau)\rho_{\pi_\theta}(\tau)\nabla_\theta \ln(\rho_{\pi_\theta}(\tau))d\tau = E_{\pi_\theta}[G_0\nabla_\theta \ln\left(\rho_{\pi_\theta}\right)]$$

Empirical estimate of $\nabla_\theta J\,(\theta)$:

$$\nabla_\theta J\,(\theta) \approx \frac{1}{N}\sum_i \left[\; G_0^i * \sum_{t\,=0}^{T_i} \nabla_\theta \ln \pi_\theta\left(a_t^i\,\middle|s_t^i\,\right)\;\right]$$

REINFORCE

Causality problem: the first rewards (in G_0^i) have here an influence on the gradients of the last actions
Solution : we pass G_0^i in the sum and remove the rewards before t'.

Policy Gradients : problems

Empirical causal estimate of $\nabla_\theta J_t(\theta)$:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \sum_{t=0}^{T_i} \gamma^t G_t^i \, \nabla_\theta \ln \pi_\theta\left(a_t^i \,|\, s_t^i\right)$$

REINFORCE

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\quad (G_t)$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

Variance problem: gradient policies suffer from large variance problems
Having a non-centric measure of how good the policy is ($G_i^{t'}$) makes learning unstable.

Solution: add a baseline to center this measure:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \sum_{t=0}^{T_i} \gamma^t \left(G_t^i - b(s_t)\right) \nabla_\theta \ln \pi_\theta\left(a_t^i \,|\, s_t^i\right)$$

adding of a baseline

On-policy : these algorithms improve π from itself, and are therefore necessarily On-policy.

Policy Gradients : adding a baseline

Adding a baseline:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \sum_{t=0}^{T_i} \gamma^t (G_t^i - b(s_t))\, \nabla_\theta \ln \pi_\theta(a_t^i \mid s_t^i)$$

adding a baseline:

Effect: reduction of the variance without increasing the bias :

$$\nabla_\theta \int_\tau b(s_t)\rho_{\pi_\theta}(\tau)d\tau = b(s_t)\nabla_\theta \int_\tau \rho_{\pi_\theta}(\tau)d\tau = b(s_t)\nabla_\theta(1) = 0$$

Choice of $G_t^i - b(s)$:

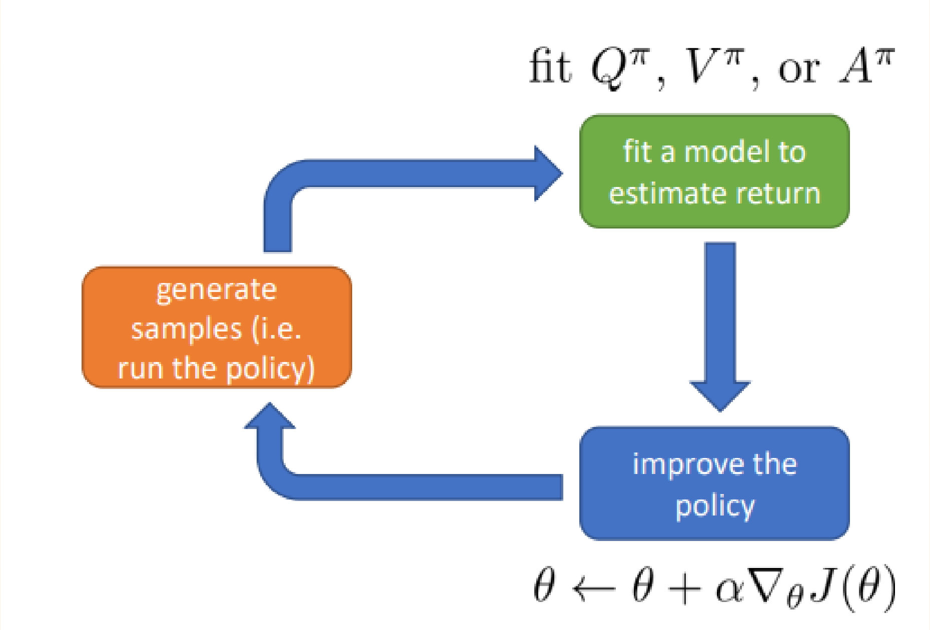
- $G_t - \widehat{v_\pi}(s_t)$
- $R_t + \gamma\widehat{v_\pi}(s_t) - \widehat{v_\pi}(s_{t+1})$
- $\widehat{q_\pi}(s_t, a_t)$
- $\widehat{q_\pi}(s_t, a_t) - \widehat{v_\pi}(s_t) \coloneqq \widehat{A_\pi}(s_t, a_t)$ (advantage function)

The ideal is an unbiased, low variance estimator of A.
The choice is a bias/variance/terminality trade-off

Actor Critic algorithms

Actor Critics are algorithms that train both a policy π (the actor) as well as a "critic" v and/or q that will help train π.

Actor-Critic training loop



Actor π_θ training :

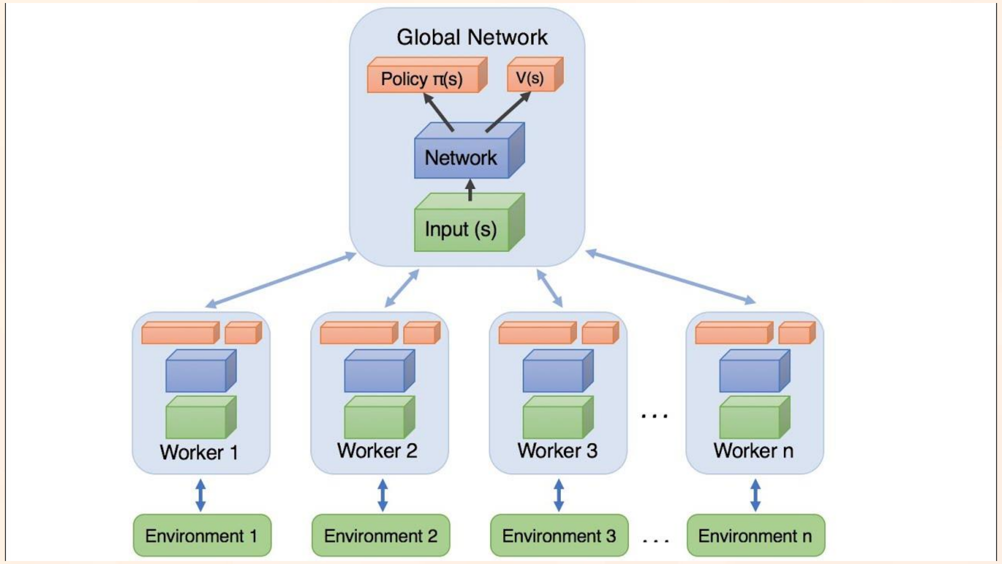
$$\theta \coloneqq \theta + \alpha \sum_{t'=0}^{T} \gamma^{t'} (G_{t'} - \widehat{v_\pi^\varphi}(s)) \ * \nabla_\theta \ln \pi_\theta \left(a_{t'} \middle| s_{t'}\right)$$

Critic $\widehat{v_\pi^\varphi}$ training :

$$\varphi \coloneqq \varphi - \alpha' \, \nabla_\varphi (\mathrm{Loss}\left(\widehat{v_\pi^\varphi}(s), G_t\right))$$

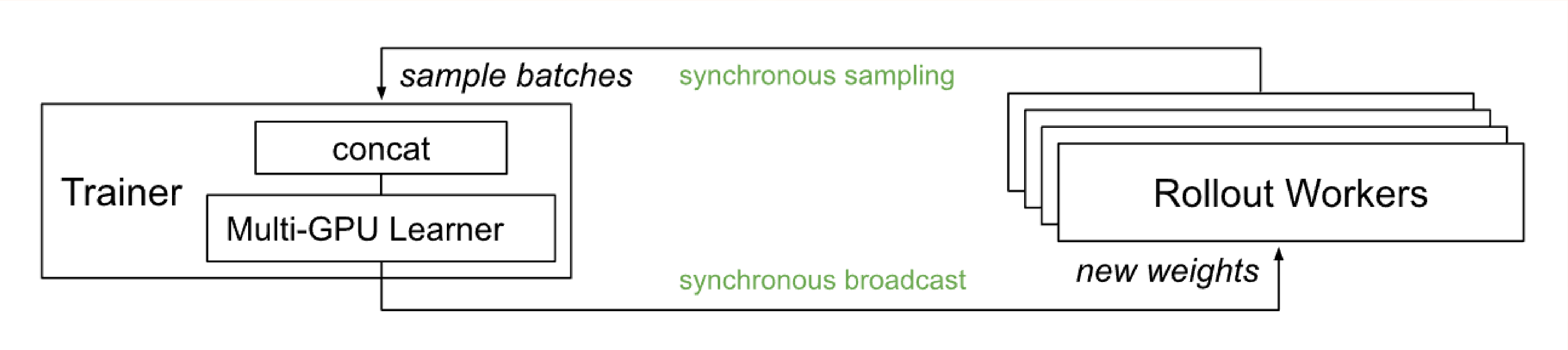
Parallelizing RL

Problem: by working on-policy, we obtain data from the same episodes and therefore correlated, which increases the variance.



Parallelization allows:

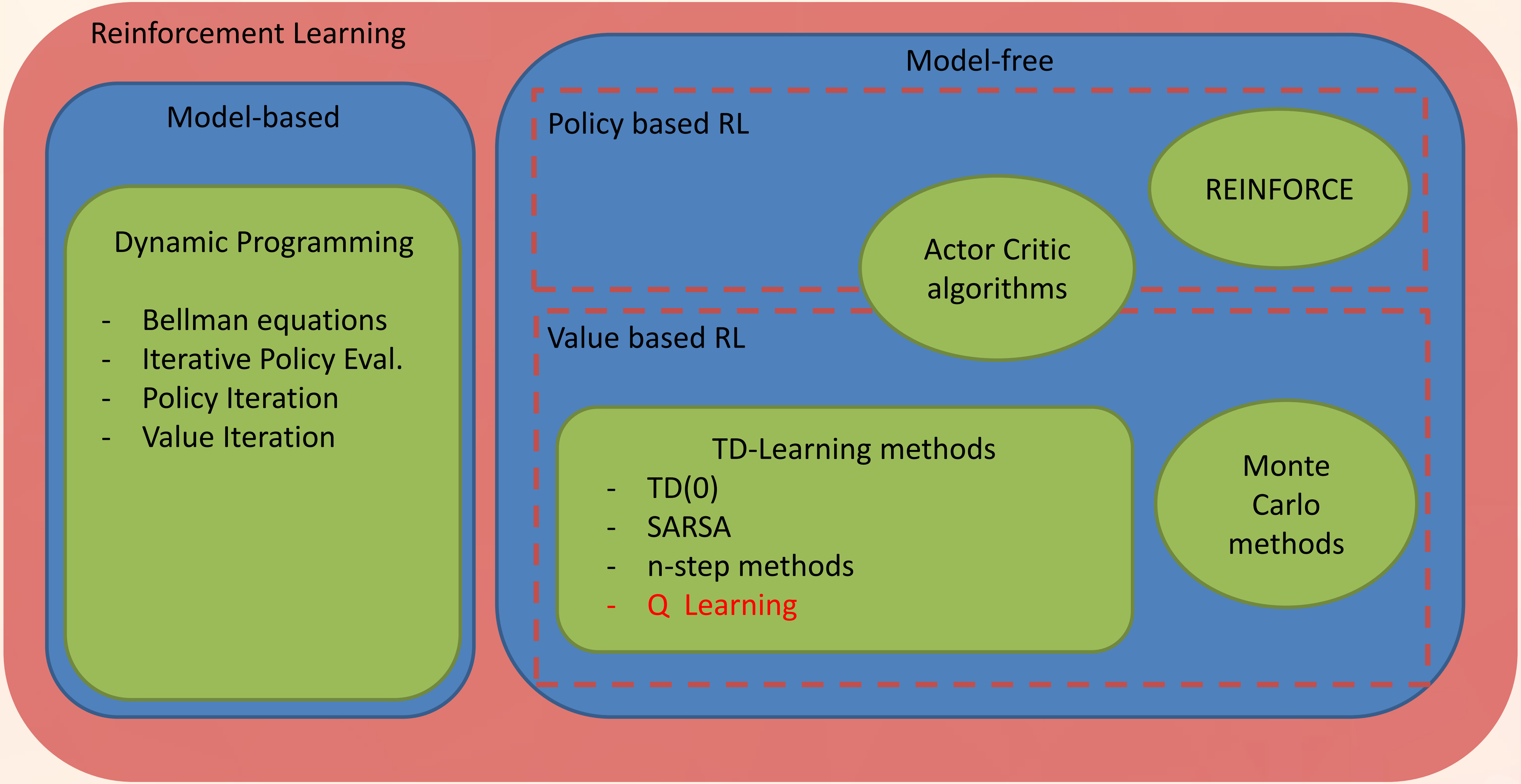
- to decorrelate the data batches, which reduces the variance
- to run several environments at the same time (time saving)
- to vectorize data (time saving through the use of GPUs)



General issues in RL

Instability due to variance in G_t
Instability due to simultaneous training of multiple networks
Little guarantee of convergence in the Deep RL case
High sensitivity to hyperparameters (which are numerous)

RL Taxonomy : a summary of the course



Python librairies for RL

For the environments: Gym

For agents :

<u>StableBaselines3 :</u>
- simple and fast in application
- choice by default

<u>RLlib :</u>
- Dealts with multi-agent and hierarchical RL
- scalable (built on Ray)
- For big projects

Other libraries: CleanRL, ML agent (Unity), OpenAI Baselines, KerasRL ...

Ressources in RL

Reinforcement Learning : an introduction, Sutton & Barto
DeepMind 2021 course on RL: https://dpmd.ai/DeepMindxUCL21
Playing Atari with Deep Reinforcement Learning, DeepMind, 2013
Spinning Up : https://spinningup.openai.com/en/latest/
Lilian Weng's blog on RL: https://lilianweng.github.io
Value-based RL basics : Medium article
Policy-based RL basics : Medium article

The End

Questions ?

APPENDIX

Off Policy using Importance Sampling

It is possible to transform an on-policy algo into an off-policy algorithm if we know π and μ

Importance Sampling: estimate the expectation of a distribution from samples from a different distribution:

$$\mathbb{E}_p[f(\mathbf{x})] = \int p(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} = \int q(\mathbf{x}) \left[\frac{p(\mathbf{x})}{q(\mathbf{x})} f(\mathbf{x}) \right] d\mathbf{x} = \mathbb{E}_q \left[\frac{p(\mathbf{x})}{q(\mathbf{x})} f(\mathbf{x}) \right]$$

Application to RL, example with R_0:

$$\boxed{E_\pi[R_0] = \sum_{a_0} \pi(a_0|s_0) R_{s_0}^{a_0} = \sum_{a_0} \mu(a_0|s_0) \frac{\pi(a_0|s_0)}{\mu(a_0|s_0)} R_{s_0}^{a_0} = E_\mu[\frac{\pi(A_0|S_0)}{\mu(A_0|S_0)} R_0]}$$

High variance

Interpretation : If τ (obtained with μ) is more likely to arrive with μ than with π, it makes sense that it weighs less in the calculation of $\hat{E}_\pi[f(x)]$.

TD(0) Off Policy : $\widehat{v_\pi}(s_t) \leftarrow \left(R_t + \gamma \widehat{v_\pi}(s_{t+1})\right) * \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}$