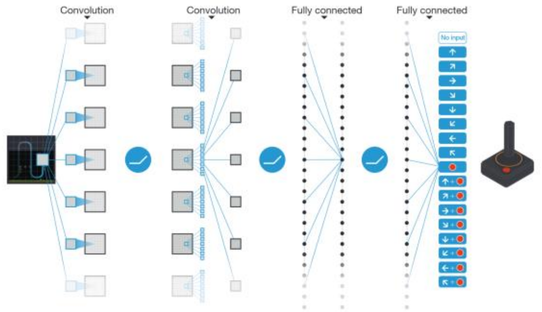
An introduction to Reinforcement Learning and Deep RL

Lucile Sassatelli (UCA, CNRS, I3S and IUF)

LJAD-I3S seminar, October 21st, 2020

General introduction

- Recent advances in Deep RL have demonstrated (super) human-level performance in complex, high-dimensional spaces:
  - DQN for Atari 2600 (2015), AlphaGo (2016), AlphaGo Zero (2017)
  - AlphaStar (2019), Hide and Seek (2019)



UNIVERSITÉ CÔTE D'AZUR

Outline

1. Main concepts and notations
   - Reward, objective, value function, etc.

2. Tabular methods
   - K-armed bandits, Monte Carlo, Time-Difference learning, On- and Off-policy

3. Function approximation
   - Deadly triad, policy-gradient methods

4. Perspectives on DRL

   Recent advances, limits, beyond RL

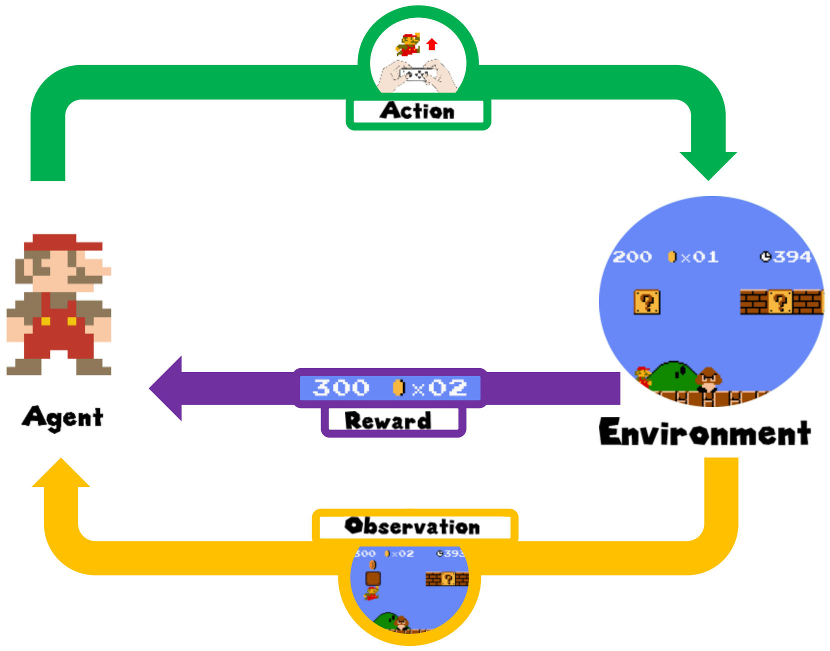
References

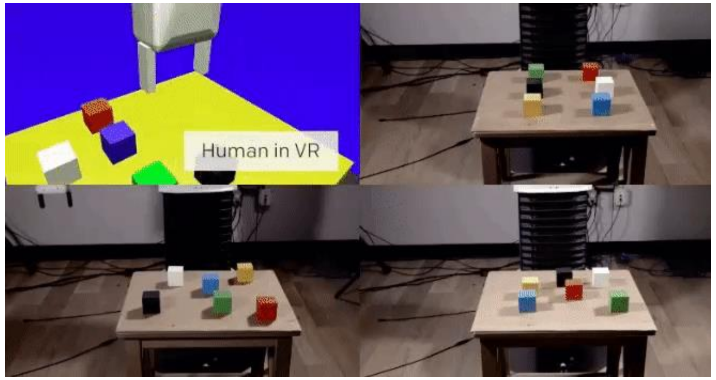
- Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An introduction. MIT Press, 2018.
  - Freely available at: http://www.incompleteideas.net/book/the-book-2nd.html
- M. Romero, F. Precioso, L. Sassatelli. Lab session on Deep Reinforcement Learning. UCA Deep Learning School 2019.
  - Freely available at: https://colab.research.google.com/drive/13Q3nTOJY9vYbhg1C0aXXIVM-IQaVVSaU#scrollTo=EhiX9eQ7DvCe
- Alex Irpan. Deep RL does not work yet. Blog: https://www.alexirpan.com/2018/02/14/rl-hard.html
- Henderson et al.. Deep Reinforcement Learning that Matters. AAAI 2018.
- Hafner et al.. Learning Latent Dynamics for Planning from Pixels. ICML 2019
- Ha and Schmidhuber. World Models. Arxiv 2018.
- Yan et al.. Learning in situ: a randomized experiment in video streaming. NSDI 2020.

UNIVERSITÉ CÔTE D'AZUR

Definition of Reinforcement Learning (RL)

- RL is a branch of machine learning aimed at teaching an agent (or several) to react to a dynamic environment to maximize some return.



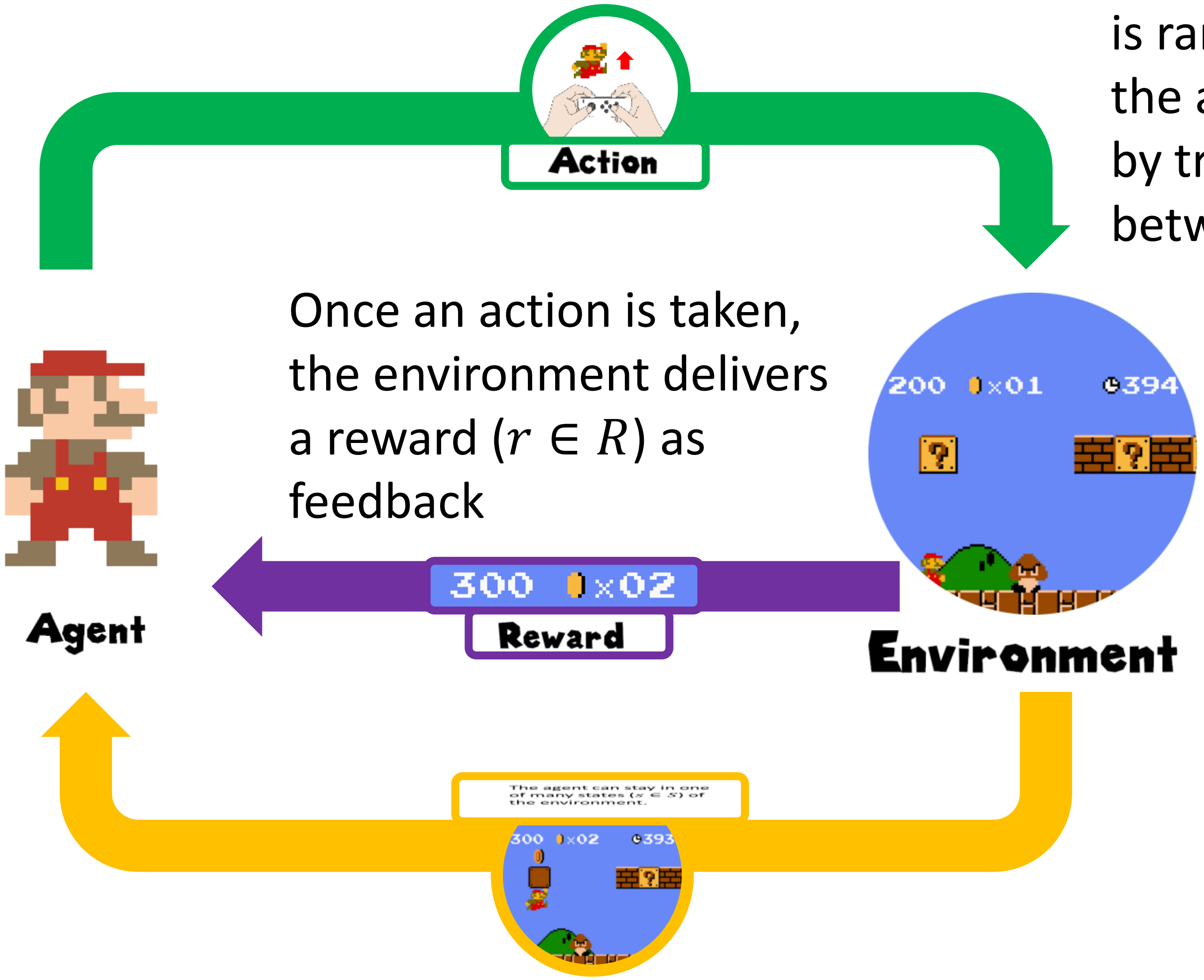


UNIVERSITÉ CÔTE D'AZUR

RL formal description: main elements

Which state the env will evolve to is random and partly depends on the action of the agent: modeled by transition probabilities between states (P)

The agent can choose to take one of many actions ($a \in A$) when the env in is state $s \in S$.



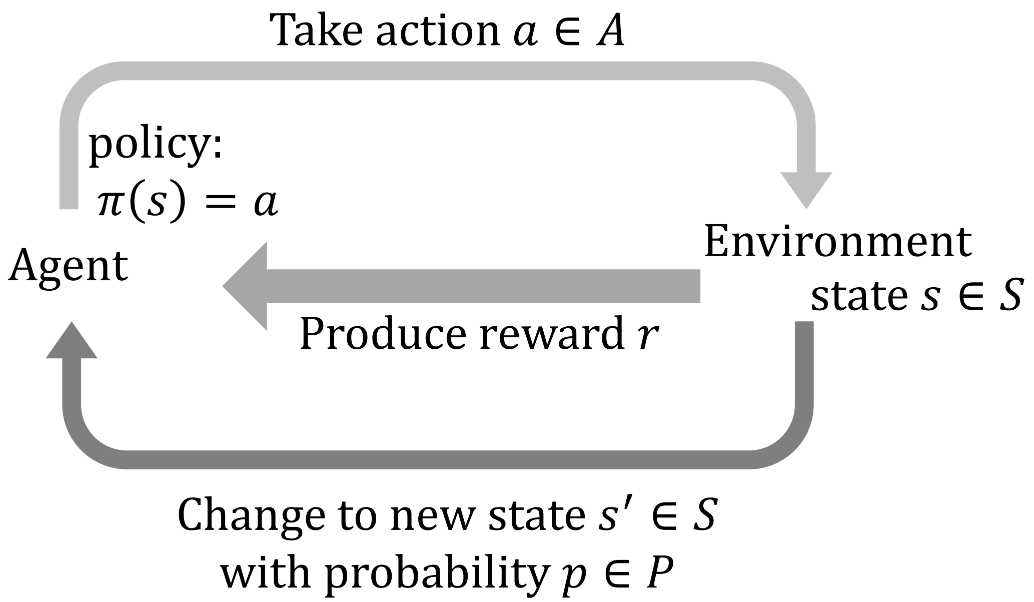
Once an action is taken, the environment delivers a reward ($r \in R$) as feedback

The env can be in one of many states ($s \in S$).

UNIVERSITÉ CÔTE D'AZUR

RL formal description: the policy function π(.)

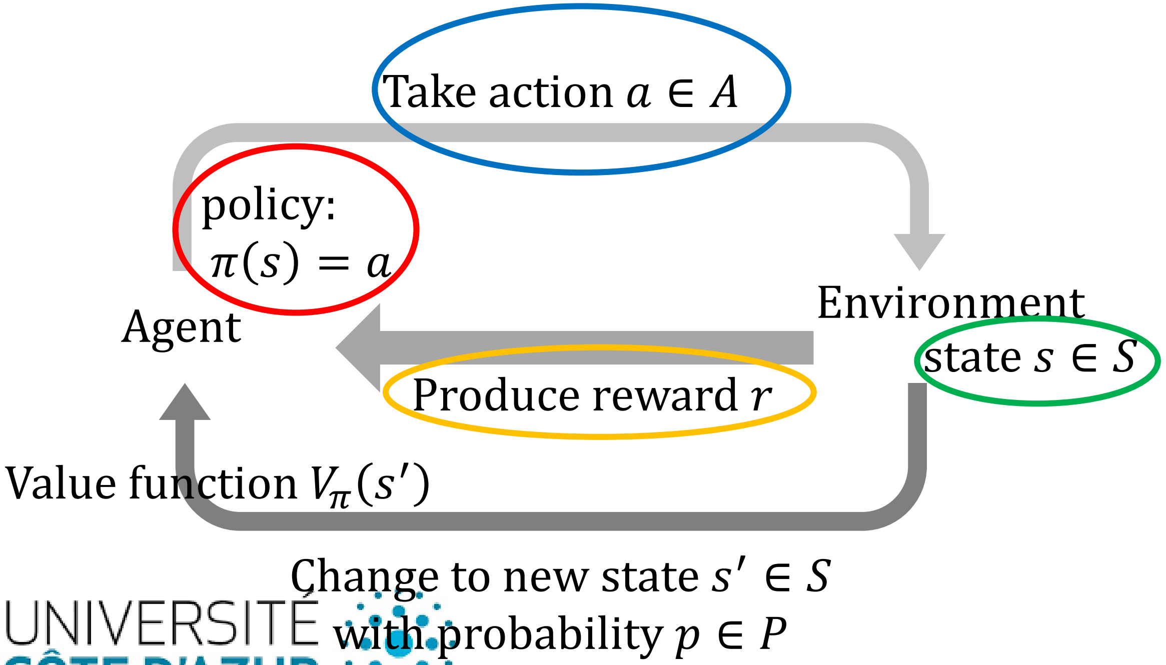
- The agent's behavior is described by policy function $\pi(s)$, indicating which action to take in state s:
  - Deterministic: $\pi(s)=a$
  - Stochastic: $\pi(a|s)=P_{\pi}[A=a|S=s]$



The objective of RL

- The formal objective of RL is finding π(.) such that:

$$\pi(\cdot) = \arg\max_{\pi} \mathbb{E}_{s \in S, a \sim \pi(\cdot)} [\textstyle\sum_{t=0}^{\infty} \gamma^t r_t]$$



The discounting factor $\gamma \in [0,1]$ penalizes the rewards in the future.

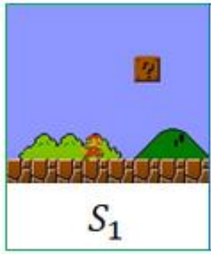
Cumulative discounted reward from time t is defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

UNIVERSITÉ CÔTE D'AZUR

Transitions and reward

- π(.) does not only impact the rewards, but also the succession of states.



- An **episode** describes the evolution of the variables of interest:

$$(S_0, A_0, R_0), \ldots (S_t, A_t, R_t), \ldots, (S_T, A_T, R_T)$$

UNIVERSITÉ CÔTE D'AZUR

Model of transitions

- The model is a descriptor of the environment. If the environment has the Markov property:

$$P[S_{t+1} | S_1,\ldots,S_t] = P[S_{t+1} | S_t]$$

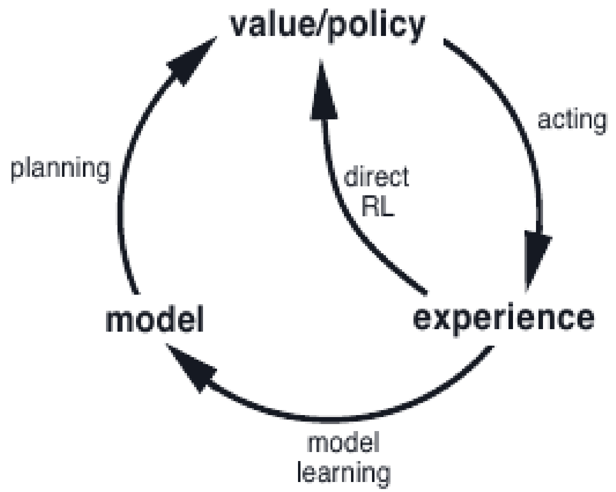
- then the environment can be entirely described by the **transition probability**:

$$P(s',r | s,a) = P[S_{t+1}=s',R_{t+1}=r | S_t=s,A_t=a]$$

UNIVERSITÉ CÔTE D'AZUR

Do you know the model of the environment?

- YES, we know the model:
  - planning with perfect information → find the optimal solution with Dynamic Programming.
- NO, we do not:
  - Learn to act with incomplete information
    → model-free RL: do not explicitly learn the environment model, focus on reward
    → model-based RL: learn the model explicitly as part of the algorithm



Quality function $Q_\pi(s,a)$ and Value function $V_\pi(s)$

- To choose the action, the **action value function** (also known as “Q-value” or "Q-function", Q standing for “Quality”) can be considered:

$$\boxed{Q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]}$$

- **Q_π(s,a)** can be handily expressed from $V_\pi(s')$, which corresponds to Bellman equations:

$$Q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_\pi(s')$$

where

$$\boxed{V_\pi(s') = \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s']}$$

- **V_π(s’)** is the **state value function**: it predicts the cumulated discounted sum of future reward, starting from state s’ and given we follow policy π(.).

UNIVERSITÉ CÔTE D’AZUR

Outline

1. Main concepts and notations
   - Reward, objective, value function, etc.

2. Tabular methods
   - K-armed bandits, Monte Carlo, Time-Difference learning, On- and Off-policy

3. Function approximation
   - Deadly triad, policy-gradient methods

4. Perspectives on DRL

   Recent advances, limits, beyond RL

Multi-armed bandits

- Expected reward for action q:

$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a]$$

- The Exploitation vs. Exploration dilemma
  - ε-greedy policy
  - Other ways

A simple bandit algorithm

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

Loop forever:

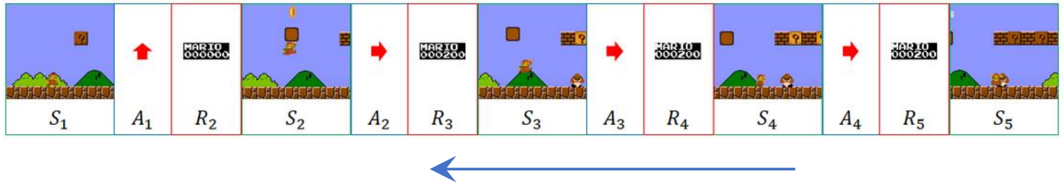
$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

© Sutton and Barto

UNIVERSITÉ CÔTE D'AZUR

Monte Carlo methods



- Generate entire trajectory then update $Q_\pi(s,a)$
- Prediction:

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

      Append G to $Returns(S_t)$

      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

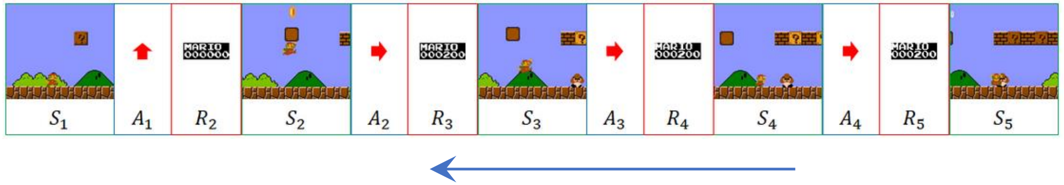
Monte Carlo methods

2 possible ways for control:

- On-policies: evaluate and improve the policy that is used to generate the data
  - simpler
- Off-policies: evaluate and improve a policy (target) different from that used to generate the data (behavior)
  - Greater variance and slower convergence
  - More powerful and general

UNIVERSITÉ CÔTE D'AZUR

Monte Carlo methods



- On-policy control:

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

  $\pi \leftarrow$ an arbitrary ε-soft policy

  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

      Append G to $Returns(S_t, A_t)$

      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

      For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

MC methods

- Off-policy
  - Relative probability of the trajectory under the target and behavior policies (the importance-sampling ratio) is $\prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

MC methods

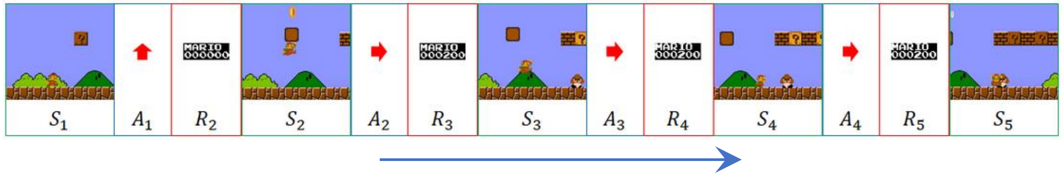
- Generalized Policy Iteration (GPI): interacting process of policy evaluation and policy improvement
- MC methods may be less harmed by violations of the Markov property
- Issue of maintaining sufficient exploration
- Off-policy methods subject of many researches

UNIVERSITÉ CÔTE D'AZUR

Temporal-Difference learning

- TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap).
- At time t+1 they immediately update with observed R_{t+1} and the estimate $V(S_t+1)$:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$



Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

UNIVERSITÉ CÔTE D'AZUR

Time-Difference learning

- Guessing from a guess: is it sound?
  → Yes, convergence proof for the <u>estimation</u> (not on policy optimality)

- Convergence proofs apply to the tabular case: much harder with function approximation

UNIVERSITÉ CÔTE D'AZUR

Sarsa: On-policy TD Control

- On-policy: continually estimate Q_π and change π greedily wrt Q_π

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop 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

- Convergence of Sarsa:
  - Depends on the policy's dependence on Q
  - Converges with proba 1 to optimal policy if <u>all S-A pairs visited</u> an infinite number of times, and the policy converges to the greedy policy

UNIVERSITÉ CÔTE D'AZUR

Q-learning: Off-policy TD Control

- Early breakthrough in RL (1989)

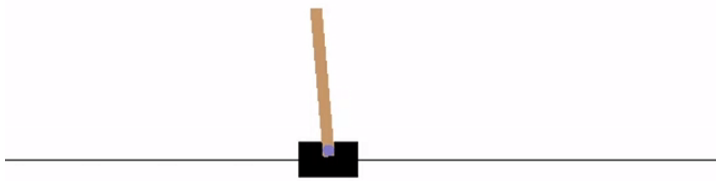
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

- Q has been shown to converge with probability 1 to Q*:
  - <u>All (S,A) pairs</u> must continue to be visited and updated
  - Usual conditions on step-size

Tabular Q-learning: implementation



- Q-learning: all (S,A) pairs can be simply tracked in a dictionary (tabular approach)
  - the state and action spaces need to be discretized

  → The dimension is always a limiting factor

	a_1	a_2	a_3	a_4	a_5	…
s_1	$Q(s_1, a_1)$	$Q(s_1, a_2)$	$Q(s_1, a_3)$	$Q(s_1, a_4)$	$Q(s_1, a_5)$	…
s_2	$Q(s_2, a_1)$	$Q(s_2, a_2)$	$Q(s_2, a_3)$	$Q(s_2, a_4)$	$Q(s_2, a_5)$	…
s_3	$Q(s_3, a_1)$	$Q(s_3, a_2)$	$Q(s_3, a_3)$	$Q(s_3, a_4)$	$Q(s_3, a_5)$	…
s_4	$Q(s_4, a_1)$	$Q(s_4, a_2)$	$Q(s_4, a_3)$	$Q(s_4, a_4)$	$Q(s_4, a_5)$	…
s_5	$Q(s_5, a_1)$	$Q(s_5, a_2)$	$Q(s_5, a_3)$	$Q(s_5, a_4)$	$Q(s_5, a_5)$	…
s_6	$Q(s_6, a_1)$	$Q(s_6, a_2)$	$Q(s_6, a_3)$	$Q(s_6, a_4)$	$Q(s_6, a_5)$	…
…	…	…	…	…	…	…

UNIVERSITÉ CÔTE D'AZUR

Outline

1. Main concepts and notations
   - Reward, objective, value function, etc.

2. Tabular methods
   - K-armed bandits, Monte Carlo, Time-Difference learning, On- and Off-policy

3. Function approximation
   - Deadly triad, policy-gradient methods

4. Perspectives on DRL

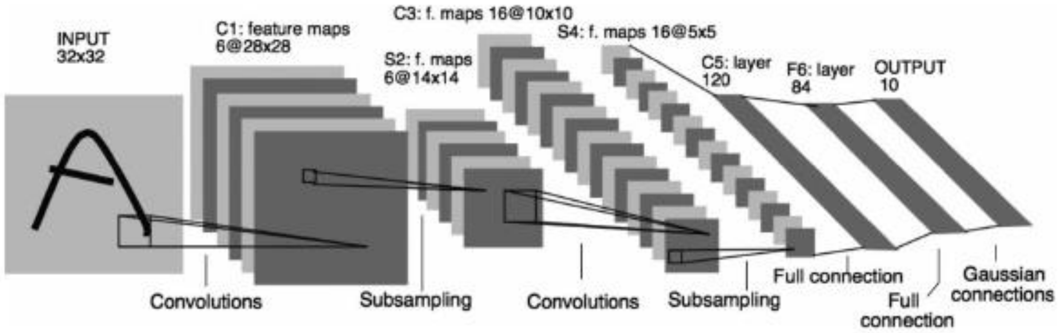
   Recent advances, limits, beyond RL

Function approximation

- The approximate value function is represented not as a table but as a functional parameterized with weight vector **w**:

$$\hat{v}(s,\mathbf{w}) \approx v_\pi(s)$$

- Also suited to partially observable problems
- Key-challenge: feature representation of space
  - With linear models: Fourier, RBF, etc.
  - With non-linear models: ANN learn feature representation appropriate to a certain problem



Function approximation

- Semi-gradient method: bootstrapping with only part of the gradient
  - do not converge as robustly as gradient methods

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose $A \sim \pi(\cdot|S)$
        Take action A, observe R, S'
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
        $S \leftarrow S'$
    until S is terminal

On-policy Control with Approximation

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

UNIVERSITÉ CÔTE D'AZUR

Off-policies Control with Approximation

- Deadly triad: training could be unstable if the updates are not done according to the on-policy
  - Off-policy training
  - Bootstrapping
  - Function approximation

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \rho_t \delta_t \nabla \hat{v}(S_t, \mathbf{w}_t)$$

$$\delta_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)$$

UNIVERSITÉ CÔTE D'AZUR

Off-policy Control with Approximation

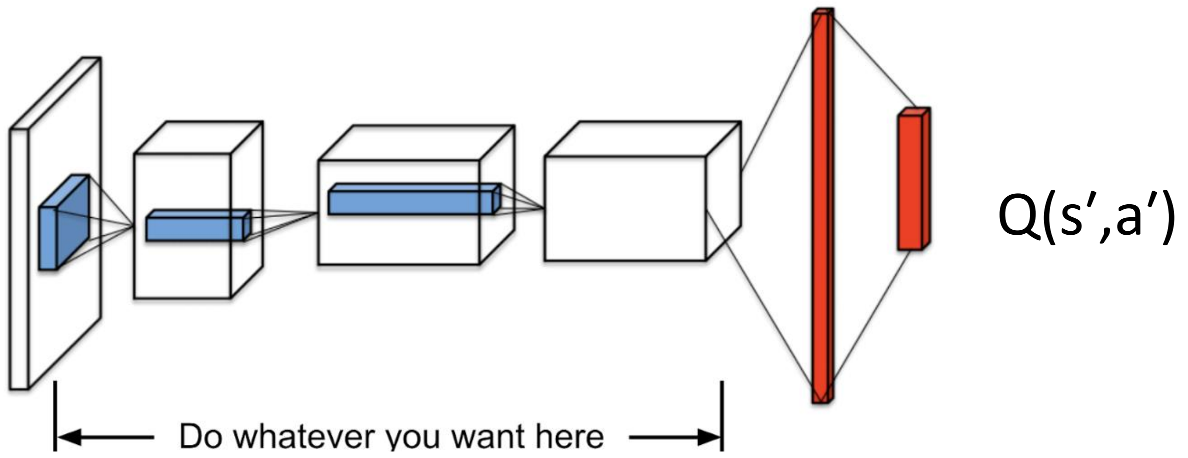
- Alternatives to semi-gradient methods are costly

- If any two elements of the deadly triad are present, but not all three, then instability can be avoided

→ Which one to give up?

Deep Q-Network (DQN, 2015)

- DQN aims at greatly improving and stabilizing the training with:
  - Experience replay: All the episode steps (S_t, A_t, R_t, S_{t+1}) are stored in one replay memory: samples drawn at random and used multiple times.
  - Partly frozen target network: estimate of Q(s',a') obtained from $Q(s',a';w^-)$, where w^- is the weights parameters updated less frequently that w



© L. Weng

Policy gradient methods

- Estimate $\pi(a|s, \boldsymbol{\theta})$
  - Can be simpler for discrete and continuous action spaces
  - Can approach deterministic and stochastic policies
  - Embed exploration
  - Inject prior knowledge on π
  - Action probabilities can change smoothly compared with ε-greedy (-> stronger convergence guarantee)
- REINFORCE update (based on the policy gradient theorem):

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$$

UNIVERSITÉ CÔTE D'AZUR

Policy gradient methods

- Reduce variance by adding a baseline: $\hat{v}(S_t, \mathbf{w})$

→ actor-critic methods

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (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 $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

UNIVERSITÉ CÔTE D'AZUR

Outline

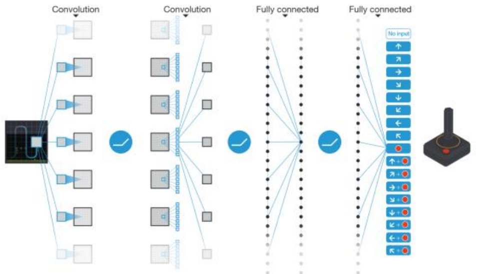
1. Main concepts and notations
   - Reward, objective, value function, etc.
2. Tabular methods
   - K-armed bandits, Monte Carlo, Time-Difference learning, On- and Off-policy
3. Function approximation
   - Deadly triad, policy-gradient methods
4. Perspectives on DRL

   Recent advances, limits, beyond RL

UNIVERSITÉ CÔTE D'AZUR

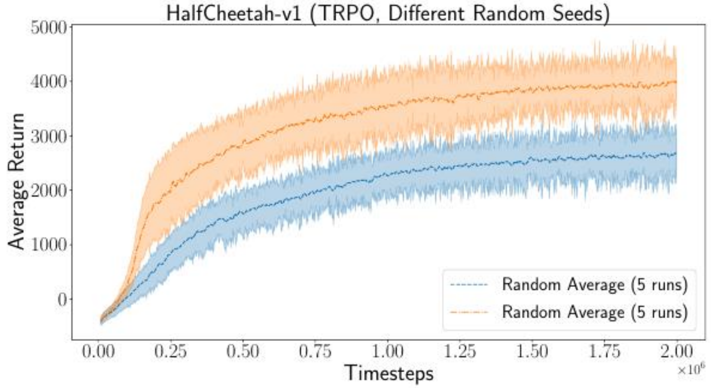
Recent progresses in Deep RL

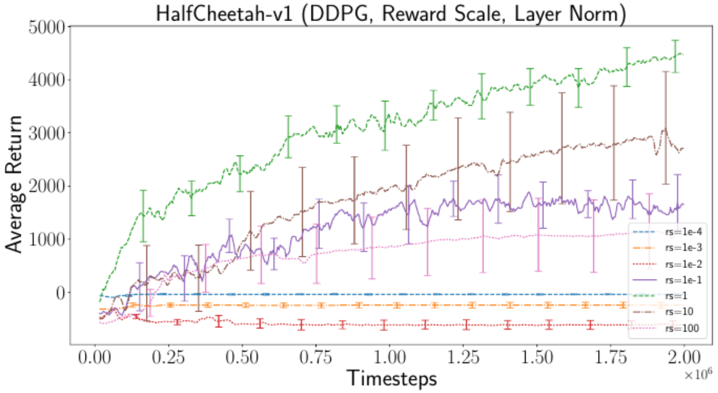
- Atari (Mnih et al., 2015):
  - DQN on 49 Atari 2600 video games
- Alpha Go (Silver et al., 2016):
  - RL for policy and value function to drive MCTS in self-play
  - Human supervision for initializing weights before RL (3 weeks)
- Alpha Go Zero (Silver et al., 2017):
  - No human data
  - Action policy-driven MCTS for self-play RL
- StarCraftII (Vinyals et al. 2019):
  - Change view, discover new strategies with new assets
  - Human data supervision, manipulation of reward, selected multi-agent RL, attention
- Hide-and-Seek (Baker, 2019)
  - Self-play, PPO, freeze adversaries, attention



Limits and challenges

- Deep RL is not plug and play [1]:
  - DRL can be much sample inefficient
  - Fair competitors can be hard to find
  - RL requires a reward function to design
  - Local optima can be hard to escape
  - May easily overfit
  - Results unstable and hard to reproduce
- Random seeds, re-scaling the reward [2]





UNIVERSITÉ CÔTE D'AZUR

[1] Alex Irpan. Deep RL does not work yet. Blog: https://www.alexirpan.com/2018/02/14/rl-hard.html
[2] Henderson et al.. Deep Reinforcement Learning that Matters. AAAI 2018.

Beyond RL

- Imitation learning
- Inverse Reinforcement Learning
- Planning from learned deep models of the environment [1,2,3]

UNIVERSITÉ CÔTE D'AZUR

[1] Hafner et al.. Learning Latent Dynamics for Planning from Pixels. ICML 2019
[2] Ha and Schmidhuber. World Models. Arxiv 2018.
[3] Yan et al.. Learning in situ: a randomized experiment in video streaming. NSDI 2020.

Thank you!