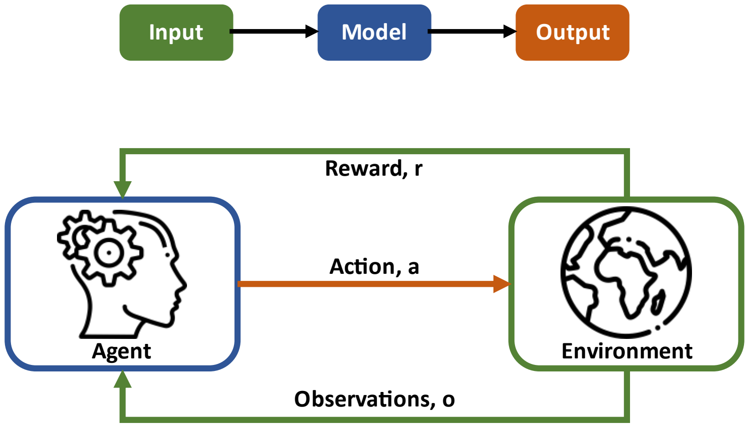
Hands-on Introduction to Deep Learning

Deep Reinforcement Learning

Institut du Développement et des Ressources en Informatique Scientifique



Context

Objective of the section:

- Conceptual discovery of Reinforcement Learning
- Opening on Deep Reinforcement Learning
-

Duration: 45 minutes

Aspects addressed:

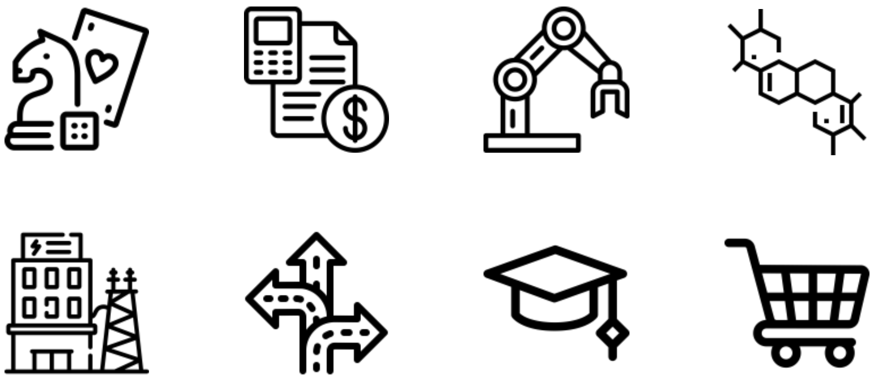
- Reinforcement Learning context uses
- Fields of application
- History or RL
- General concepts
- Limitations of traditional Reinforcement Learning
- Contributions of neural networks to RL
- Tools to train a RL algorithm

Objectives:

- Create an autonomous agent
- Able to make decisions in an environment
- Without a priori knowledge of the solution during training

Reinforcement Learning:

- Agent maximises rewards (indirect supervision)
- Learn from experience (trial and error)



Applications

Various fields of application:

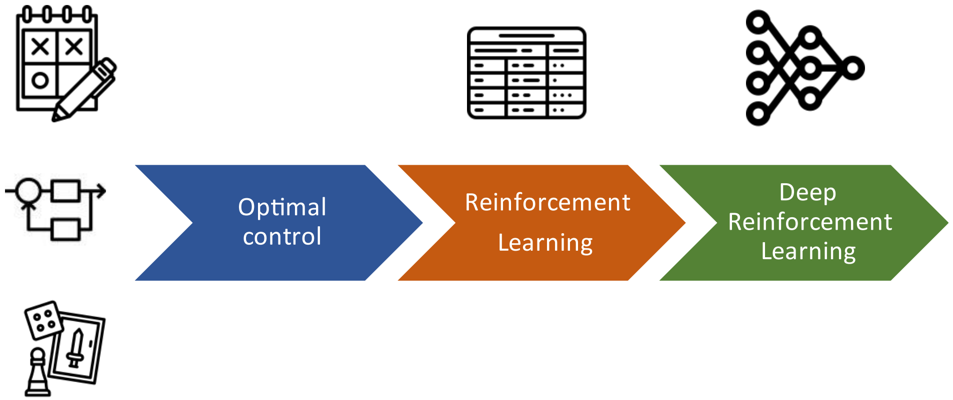
- Games
- Finance
- Robotics
- Health
- Energy Navigation
- Education
- Business

Different environments:

- Real
- Virtual
- Completely known by the agent
- Partially observed by the agent

Various objectives:

- Prediction
- Optimisation
- Decision making
- Recommendation
- Control



History of Reinforcement Learning

3 aspects explored in parallel

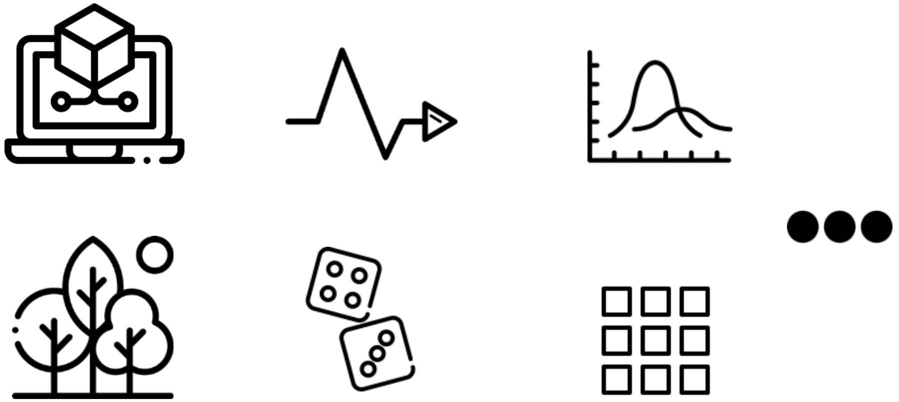
- Try error
- Optimal control
- Game theory

1980' :

- Reinforcement Learning algorithms
- Temporal Difference combined with Optimal Control
- Q Learning

2010' :

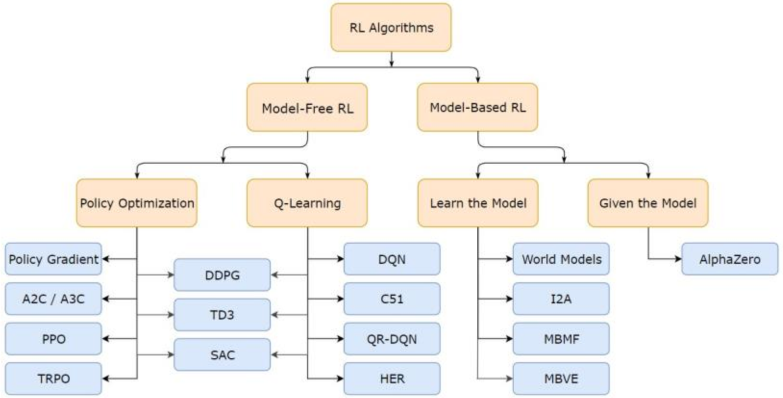
- DRL breakthrough from Deepmind with DQN
- Multiple achievements against best players of various games
- State of the art algorithms in science (Ex : Protein folding with AlphaFold)



Environment

Environment :

- Real or virtual (simulated)
- Static or Dynamic
- Evolves over time (dynamic environment) or only after each action of the agent (example: turn-based games)
- Can be partially or completely observed by the agent
- Rewards the agent according to the state of the environment

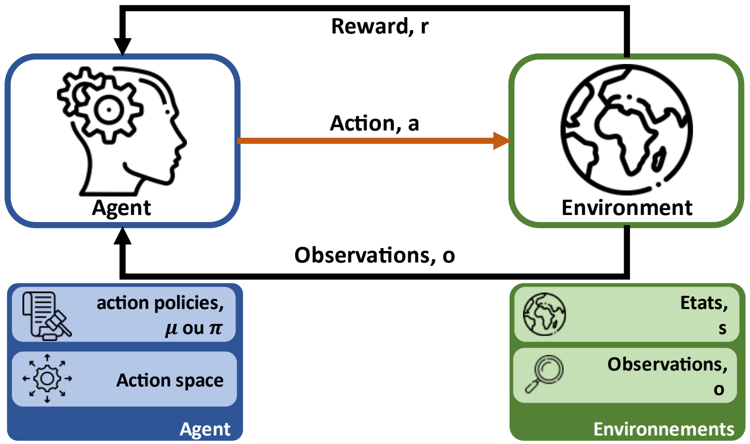


Part 2: Kinds of RL Algorithms — Spinning Up documentation. Spinningup.openai.com. (2022)

Taxonomy

Model-based : Agent has access to a prediction of what is coming next. The prediction can come from a learned model of the environment or simply given to the agent

Model-free : Agent has no access to a prediction of the state transitions and rewards.



Terminology

Agent:

- A predefined set of possible actions
- An action policy
    - Determines which action to choose in response to a state of the environment
    - The action policy used for training may be different from the one that will eventually be used.
    - It can be deterministic or stochastic.

- **Trajectories:**
  $\tau = (s_0, a_0, s_1, a_1, \dots)$
- **Rewards:**
  $r_t = R(s_t, a_t, s_{t+1})$

Finite-horizon undiscounted return

$$R(\tau) = \sum_{t=0}^{T} r_t$$

Infinite-horizon discounted return

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

- **On-policy Value Function:**
  $V^\pi(s) = \underset{\tau\sim\pi}{\mathrm{E}} [R(\tau) \,|\, s_0 = s]$
- **On-policy Action-Value (Q) Function:**
  $Q^\pi(s,a) = \underset{\tau\sim\pi}{\mathrm{E}} [R(\tau) \,|\, s_0 = s, a_0 = a]$
- **Optimal :**
  $\max_\pi$
- **Policies:**
  $a_t = \mu(s_t)$ $\qquad$ $a_t \sim \pi(\cdot|s_t)$

Bellman Equations

$$\boxed{V^\pi(s) = \underset{\substack{a\sim\pi \\ s'\sim P}}{\mathrm{E}} [r(s,a) + \gamma V^\pi(s')]}$$

$$\boxed{Q^\pi(s,a) = \underset{s'\sim P}{\mathrm{E}} \left[r(s,a) + \gamma \underset{a'\sim\pi}{\mathrm{E}} [Q^\pi(s',a')] \right]}$$

Concepts

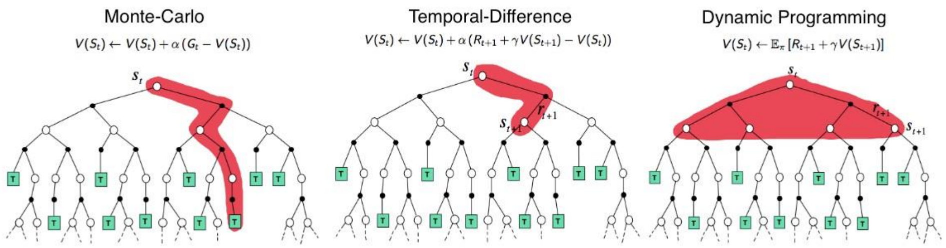
Trajectories: changes in the environment according to the agent's actions

Rewards: Defined by a law taking into account the state generated by the agent's action

Value: Evaluates the value (potential) of a state of the environment according to the expectation of optimal gain from this state

Q function: Evaluates the Quality of a chosen action in a state of the environment

Bellman Equations: refer to a set of equations that decompose the value function into the immediate reward plus the discounted future values.



David Silver's RL course lecture 4: "Model-Free Prediction"

Monte Carlo | Temporal Difference | Dynamic Programming

Dynamic Programming : need to know the environment dynamics

Monte Carlo:

- Need to finish an episode before an update
- High Variance, no bias
- Better for non-Markov

Temporal Dynamics:

- Can learn from incomplete episodes
- Low bias, low variance
- Better exploit of Markov properties

On Policy:

- Same policy used to generate experiences and to improve
- **SARSA** $\boxed{Q(a,s) \leftarrow Q(a,s) + \alpha \cdot (r_s + \gamma \cdot Q(a',s') - Q(a,s))}$

Off Policy:

- One policy (Target policy) to generate samples
- Another different policy optimized during the process
- **Q Learning** $\boxed{Q(a,s) \leftarrow Q(a,s) + \alpha \cdot \left(r_s + \gamma \max_{a'} Q(a',s') - Q(a,s)\right)}$

On | Off Policies

Behaviour Policy : The policy used to determine the actions followed by the agent at a given state.

Target Policy : The policy the agent is learning.

On Policy : Target Policy == Behavior Policy

Off Policy : Target Policy != Behavior Policy

Set values for learning rate α、 discount rate γ、 reward matrix R
Initialize $Q(s,a)$ to zeros
Repeat for each episode,do
    Select state s randomly
    Repeat for each step of episode,do
        Choose a from s using ε-greedy policy or Boltzmann policy
        Take action a obtain reward r from R, and next state s'
        Update $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        Set $s = s'$
        Until s is the terminal state
    End do
End do

Q Table	Actions
Etats	

Q Learning

Example : Tic-Tac-Toe
Liste of states and possible actions at each round
1st round: Nothing on the grid, Actions : 9 actions possible
2^{nd} round: 9 existing states (assuming cross always starts), 8 possible actions
…
For each combination, evaluate its potential by increasing its value if it lead to a better situation or the opposite.

Policy parameters optimization : $\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_{\theta_k})$

Gradient of expected finite-horizon: $\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right]$

Advantage function: $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$

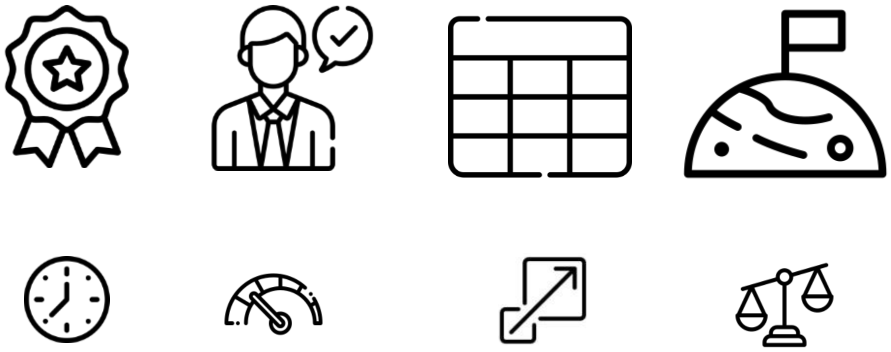
Policy Optimization – Vanilla Policy Gradients

Policy parameters are optimized using gradient ascent.

Gradient can be applied on finite or infinite-horizon expected returns.

In this exemple, the advantage function is used but it could also be the Value or Q functions.

Various algorithms such as Actor-Critic uses value optimization and policy optimization together.



Reinforcement Learning

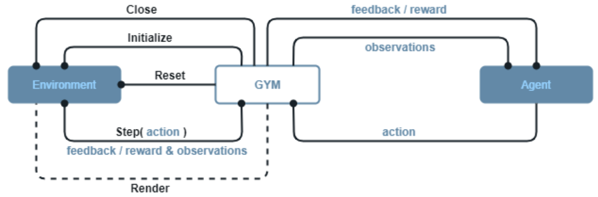
Limits :

- Rewards :
  - Can be difficult to define
  - If rare, experiences do not improve the agent
  - If intermediate rewards are created, they may induce bias and limit the agent's performance
- Exploration-Exploitation trade-off:
  - Explore unknown choices or choices with low short term reward gain to expect high long term gains
  - Choose at each point in time the strategy that has yielded the most rewards so far
- Q table :
  - Combinatorial of action-states too high to be stored and even explored

Solutions:

- Attenuation factor on rewards as a function of time
- epsilon-greedy algorithm
- Deep Reinforcement Learning

Atari
MuJoCo
Toy Text
Classic Control
Box2D
Third Party Environments

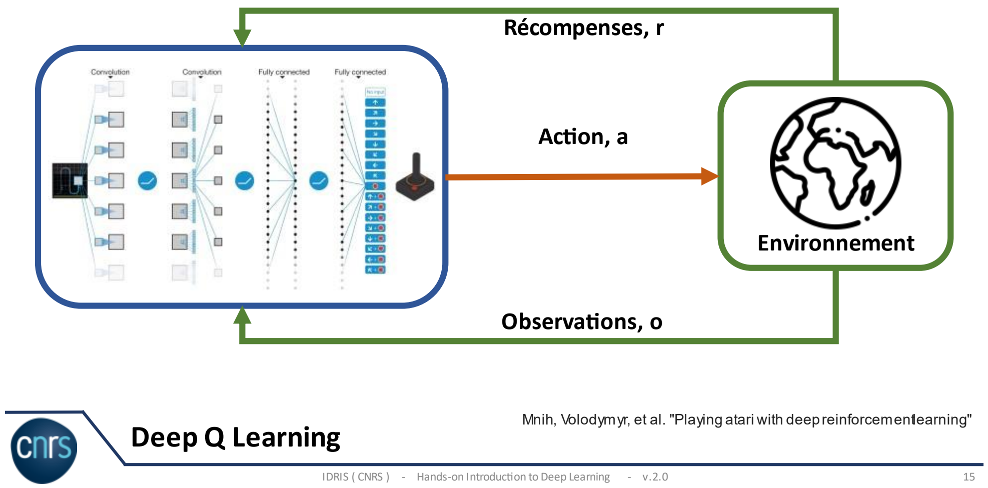


Simulated environments

Gym (OpenAI) :

- an opensource toolkit for developing and comparing reinforcement learning algorithms
- provides a standard API to communicate between algorithms and environments
- a standard set of environments

Useful to create a specific environment for a specialized problematic while having a generic pipeline with standard methods and variables.



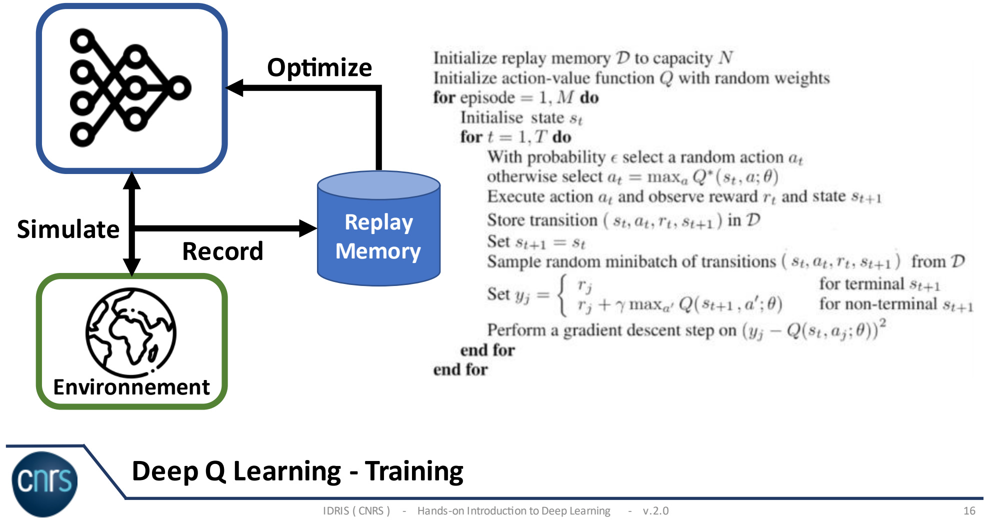
Principle: approximate the Q-table (state-action space) with a neural network

Advantages :

- Reinforcement Learning applicable to complex and real problems
- Use of "raw" observations (example: pixel of a video game)
    - Helps generalization by learning a "representation" of the environment

Limitations:

- Difficult to converge towards a solution
- Slow to train
- Generalization not so obvious



Replication of a supervised learning mechanism

How it works :

1. Environment-Model Interaction
2. Store in memory until a batch is created
3. Update the model with the batch of experiments

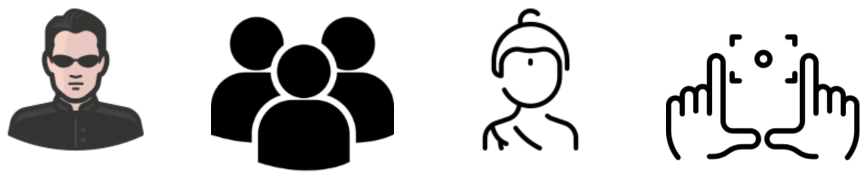
Limitation: Unstable learning

Solution:

Two models:

- 1st model used for simulations
- 2nd model updated frequently

The 1st model is occasionally updated directly with the new weights of the 2nd model.d



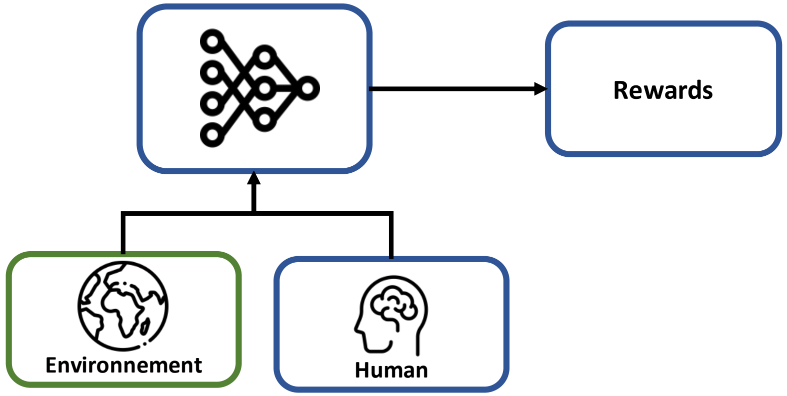
Deep Reinforcement Learning

Agent can learn the world (model based).

Hindseight experience replay : learn from rare and low rewards

Learn from both good and bad episodes

Train a general AI capable of tackling multiple problems



Inverse Reinforcement Learning

Objective: Learning the objective rather than the task

Input: Environmental states and actions chosen by an expert
Output: The rewards to be predicted