Engineering degree - Final year intership report

Deep Reinforcement Learning for soft objects grasping

Author (student):

M. Quentin GALLOUÉDEC
quentin.gallouedec@ecl16.ec-lyon.fr

Technical tutor:

M. Nicolas CAZIN
nicolas.cazin@ec-lyon.fr

School tutor:

M. Emmannuel DELLANDRÉA
emmannuel.dellandrea@ec-lyon.fr

August 26, 2020

Abstract: Reinforcement learning holds the promise of imminent robots capable of learning a wide range of skills with minimal human intervention. Among these skills, grasping soft objects would extend the scope of robotic manipulation to areas in which human dexterity has so far been indispensable. However, reinforcement learning for a manipulative task requires a large number of specific competences such as object identification and segmentation, pose detection, and task hierarchy planning. For this reason, reinforcement learning algorithms for manipulation tasks are often limited to simulated environments and simple tasks. In this document, we first present an introduction to the essential notions of Reinforcement Learning (RL) and establish a state of the art of work concerning the manipulation of soft objects. An interesting method from the literature, Hindsight Experience Replay (HER), is then tested. For a rigid object, we show that in only three hours of training on a conventional computer, a simulated robot reaches a success rate of 93.1% for the pick and place task. Work in progress, whose results are not presented in this document, aims at determining whether HER performs as well with soft objects.

Keywords: Deep Reinforcement Learning, Hindsight Experience Replay, soft objects grasping.

Résumé : L'apprentissage par renforcement constitue la promesse d'imminents robots capables d'apprendre avec un minimum d'intervention humaine un vaste éventail de compétences. Parmi ces compétences, la saisie d'objets mous permettrait d'élargir le champs d'action de la manipulation robotique aux domaines dans lesquels la dextérité humaine est jusqu'à ce jour indispensable. Toutefois, l'apprentissage par renforcement d'une tâche de manipulation requiert un grand nombre de compétences spécifiques comme l'identification et la segmentation d'objets, la détection de pose ou encore la planification d'une hiérarchie des tâches. C'est pourquoi les algorithmes d'apprentissage par renforcement pour les tâches de manipulation sont souvent limitées à des environnements simulés et à des tâches simples. Dans ce document, nous présentons d'abord une introduction aux notions essentielles de l'apprentissage par renforcement et établissons un état de l'art des travaux concernant la manipulation d'objets mous. Une méthode intéressante de la littérature, HER, est ensuite éprouvée. Pour un objet rigide, nous montrons qu'en seulement trois heures d'entrainement sur un ordinateur classique, un robot simulé atteint un taux de succès de 93.1% pour la tâche *pick and place*. Les travaux en cours, dont les résultats ne sont pas présentés dans ce document, ont pour but de déterminer si HER performe aussi bien avec des objets mous.

Mots clés : apprentissage par renforcement profond, expérience rétrospective, saisie d'objets mous.

Preliminary comments

This report presents the steps and results of my final year work which punctuates my 3 years of studies at *École centrale de Lyon* in order to obtain the engineering degree. This 21-week internship is part of my academic project as a preliminary period for the doctorate that I would like to start at the end of my internship. The subject was deliberately chosen to be close to the field I am passionate about: Deep Reinforcement Learning (DRL). This internship is of particular interest since it consists in carrying out a research work that allows me to apprehend the field that should be the one of my thesis.

This internship started on May 11, 2020 and took place in the middle of the health crisis due to the Covid-19 pandemic. In order to respect social distancing measures, the internship took place entirely in telework. The team did its best to continue its research work in this particular context. To ensure regular follow-up of the work, and to maintain a link between the laboratory staff, an informal telephone meeting was organised on a daily basis. In spite of the efforts, some aspects of the daily life of the laboratory were absent from my internship: improvised exchanges, group dynamics, team life...

Everything presented in this report is the result of my first 13 weeks of work. The two oral presentations will take place respectively on September 2, 2020 (week 17 of my internship) and September 18, 2020 (week 18 of my internship). I will try to integrate the most recent results into my presentations. The information in this report may therefore differ from the information presented during these talks.

Summary of notations

The notations used in this report are exactly the same as those used by Sutton and Barto [1] in their book.

Capital letters are used for random variables, whereas lower case letters are used for the values of random variables and for scalar functions. Quantities that are required to be real-valued vectors are written in bold and in lower case (even if random variables). Matrices are bold capitals.

$\doteq$	equality relationship that is true by definition
$\approx$	approximately equal
$\subset$	subset of
$\in$	is an element of
$\sum_{x\in\mathcal{X}}$	discrete sum over the elements of $\mathcal{X}$
$\int_{x\in\mathcal{X}}$	continuous sum over the elements of $\mathcal{X}$
$\prod_{x\in\mathcal{X}}$	product over the elements of $\mathcal{X}$
$\leftarrow$	assignment
$\rightarrow$	$f : \mathcal{X} \rightarrow \mathcal{Y}$ function f from elements of set $\mathcal{X}$ to elements of set $\mathcal{Y}$
$\sim$	$X \sim p$ random variable X selected from distribution $p(x) \doteq \Pr\{X = x\}$
$\|\cdot\|^2$	norm of function on $\mathcal{S}$ i.e, $\|f\|^2 \doteq \sum_{s\in\mathcal{S}} f(s)^2$
∇, $\nabla_{\mathbf{w}}$,$\nabla_{\boldsymbol{\theta}}$	column vector of partial derivatives with respect to $\mathbf{w}$ and $\boldsymbol{\theta}$
α, β	step-size parameters
γ	discount-rate parameter
δ_t	temporal-difference (TD) error at t (a random variable)
ε	probability of taking a random action in an ε-greedy policy
λ	decay-rate parameter for eligibility traces
π	policy (decision-making rule)
$\pi(s)$	action taken in state s under *deterministic* policy π
$\pi(a\|s)$	probability of taking action a in state s under *stochastic* policy π
$\pi_{\boldsymbol{\theta}}$	policy corresponding to parameter $\boldsymbol{\theta}$
$\boldsymbol{\theta}, \boldsymbol{\theta}_k, \boldsymbol{\theta}^-$	parameter vector of target policy
a	an action
$\arg\max_a f(a)$	a value of a at which $f(a)$ takes its maximal value
$\mathcal{A}$, $\mathcal{A}(s)$	set of all actions available, denoted $\mathcal{A}(s)$ if it depends on states
A_t	action at time t

$b(s)$	a baseline function $b : \mathcal{S} \mapsto \mathbb{R}$ for policy-gradient methods
$b(a\|s)$	behavior policy used to select actions while learning about target policy π
e^x	the base of the natural logarithm, $e \approx 2.71828$
$\mathbb{E}[X]$	expectation of a random variable X, i.e., $\mathbb{E}[X] \doteq \sum_x p(x)x$
G_t	return following time t
$G_{t:t+n}$	n-step return from $t+1$ to $t+n$, or to h (discounted and corrected)
G_t^λ	λ-return
$h(s, a, \boldsymbol{\theta})$	preference for selecting action a in state s based on $\boldsymbol{\theta}$
$J(\boldsymbol{\theta})$	performance measure for the policy $\pi_{\boldsymbol{\theta}}$
k	iteration rank
$\log x$	natural logarithm of x
$L(\mathbf{w})$	loss function
$\max_a f(a)$	the value of $f(a)$ at which $f(a)$ takes its maximal value
n	in n-step methods, n is the number of steps of bootstrapping
$\mathbb{N}$	set of natural numbers
$\mathcal{N}(s, a)$	noise over a distribution of states and actions
$N_t(s)$	number of times state s has been visited
$N_t(s, a)$	number of times action a has been selected in state s
p	explorative weight in an upper confidence bound algorithm
$p(r, s' \mid s, a)$	probability of transition to state s' with reward r, from state s and action a
$\Pr\{X = x\}$	probability that a random variable X takes on the value x
$q_\pi(s, a)$	value of taking action a in state s under policy π
$q_*(s, a)$	value of taking action a in state s under the optimal policy
$q_{\mathbf{w}}(s, a)$	approximate action-value of state s given weight vector $\mathbf{w}$
Q, Q_k, Q^A, Q^B	array estimates of action-value function q_π or q_*
r	a reward
$\mathcal{R}$	set of all possible rewards, a finite subset of $\mathbb{R}$
$\mathbb{R}$	set of real numbers
R_t	reward at time t, typically due, stochastically, to S_{t-1} and A_{t-1}
s, s'	states
$\mathcal{S}$	set of all nonterminal states

$\mathcal{S}^+$	set of all states, including the terminal state
S_t	state at time t, typically due, stochastically, to S_{t-1} and A_{t-1}
t	discrete timestep
T	final timestep of an episode, or of the episode including timestep t
U_t	upper bound confidence at time t
$v_\pi(s)$	value of state s under policy π (expected return)
$v_*(s)$	value of state s under the optimal policy
$v_{\mathbf{w}}(s)$	approximate state-value of state s given weight vector $\mathbf{w}$
V, V_k	array estimates of state-value function v_π or v_*
$\mathbf{w}, \mathbf{w}_k, \mathbf{w}^-$	vector of weights underlying an approximate value function
$\mathbb{1}_{predicate}$	indicator function ($\mathbb{1}_{predicate} \doteq 1$ if the *predicate* is true, else 0)

Table 1: Reinforcement Learning notations

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

The manipulation of objects by a robot arm has been a subject studied for many years. While some tasks appear to be extremely simple for humans, they are particularly difficult for a robot arm to perform. Grasping an orange without dropping or crushing it is such an easy task that a child is able to do it in the first few weeks of life. Part of the reason it's so easy for us is that we have innate strengths to succeed:

- the opposition between the thumb and the other fingers,
- the unique sensitivity our hands have in relation to other parts of the body (see Penfield's *homunculus* [2]),
- an area of the brain, called *anterior intraparietal cortex* used for both the control of visually guided actions and the extraction of 3D shapes from objects [3].

Based on our innate gripper ability, the challenge of robotic grasping is one of the fundamental tasks of robotic manipulation. This task involves a wide range of skills. The robot must first recognize the object, its position, shape and pose in space, plan how to position the gripper to grasp the object correctly, choose the sequence of motor commands required to perform a movement, correctly move its gripper to the desired position, estimate the success of this task ... Around these skills, other aspects are essential: which sensors is the robot equipped with? how the robot is controlled (e.g., in torque, speed or position)? what type of gripper is the robot equipped with? These are all aspects that determine the success of the grasping task.

Improving reproducibility in LEARNing physical manipulation skills with simulators using REAListic variations (LEARN-REAL) is a european collaborative project that addresses this challenge. Within LEARN-REAL several aspects of the overall problem are studied. One task aims to provide a tool to generate RGB-D synthetic images of textured objects from a description of the objects containing among others their 6D position. Another task will focus on the simulation of soft objects interacting with a 7 degrees of freedom robotic arm. A final task aims to implement the learning aspect. The learning of the grasping task will be done using RL methods. It is this last task that this paper deals with. Section 2 gives more detailled description of the project.

The first part of the work is to assimilate the key notions of RL. Based largely on the book by Sutton and Barto [1], the lectures by David Silver[1] and Hado van Hasselt[2], the Section 3 presents the key notions of RL. This part is an essential starting point for learning work specifically focused on robotic manipulation. Section 4 presents the state of the art for RL applied to a grasping task. Some related topics are also studied such as the performance of recent tactile sensors and the gap issue from simulation to reality. Section 4 also presents the PANDA from FRANKA EMIKA, which is the robot arm we use for experimentations. A method from the literature called HER (presented Section 4.5) provides a very agile solution to the exploitation *versus* exploration trade-off for multiple goals environments. This method consists in adding hindsight

[1] `https://www.davidsilver.uk/teaching/`

[2] `https://www.youtube.com/playlist?list=PLqYmG7hTraZBKeNJ-JE_eyJHZ7XgBoAyb`

experiments to the replay buffer, corresponding to fictitious successes of the requested task. This ensures that the replay buffer is fed with episodes containing rewards. Initial results show that this method allows to obtain very good results, even for complex tasks (such as pick and place). Our experiments (Section 6) aims to evaluate the performance of HER, and determine if it can be applied to soft object grasping. The results of our experiments show that three hours of training on a conventional computer allows a simulated robot to perform the grasping task with a median success rate of 95%. The results presented are only the beginning of the ongoing series of experiments. Aspects such as the nature of the sensors, the softness of the objects have not yet been integrated. The work in progress, and those planned are presented in Section 7.

2 Context

The work presented in this document is part of an overall project called LEARN-REAL. LEARN-REAL is a european collaborative project supported by Swiss National Science Foundation (SNSF, Switzerland), the *Ministero dell'Istruzione, dell'Università e della Ricerca* (MIUR, Italy), and the *Agence nationale de la recherche* (ANR, France), through the ERA-NET CHIST-ERA 2017 Call ORMR[3] (Object recognition and manipulation by robots: Data sharing and experiment reproducibility). Arès, a joint research laboratory between the Imagine team[4] and Siléane (specialized in industrial robotics) was founded by the ANR. The laboratory works in partnership with two leading european research institutions in robotics, namely Idiap (Switzerland) and Italian Institute of Technology (Italy).

LEARN-REAL aims to enable the learning of robotic handling skills through the simulation of associated components: objects, environments and robots. To do this, three essential tools are proposed:

1. A simulator with realistic rendering. This simulator must allow the creation of a dataset taking into account realistic variations of the environment (e.g., day/night, good/bad weather). This data will be used to evaluate the algorithms in new situations.
2. A virtual reality interface to interact with the robots. This interface will help the robots in the learning process of different tasks such as object manipulation. The development of this tool will be based in particular on existing simulators from the gaming and virtual reality industry.
3. An infrastructure for comparative evaluation of learning performances. Special attention is put on the reproducibility and transparency of the proposed learning algorithms. Development and maintenance will be carried out by Idiap.

The overall workplan is divided into work packages (WPs, see Figure 1).

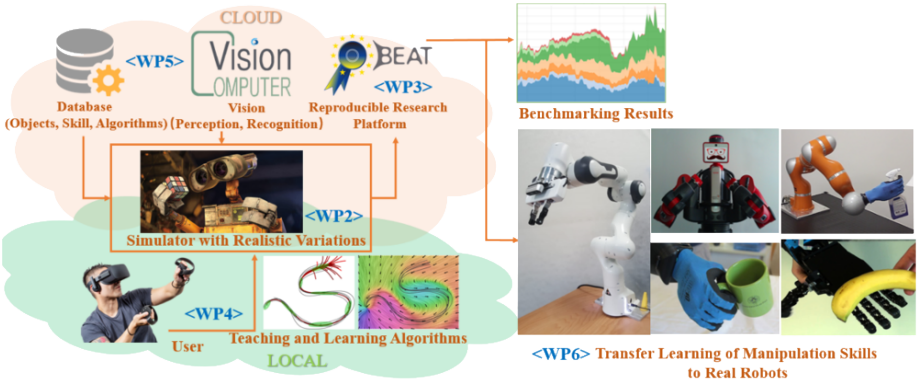


Figure 1: LEARN-REAL work plan

The chosen case study is the recognition and grasping of fruits and vegetables. The objective is to test the tool chain developed in the fruit and vegetable

[3] `https://www.chistera.eu/call-2017-announcement`
[4] at École centrale de Lyon

grasping case study. This includes the processing of camera (or simulated) data, which must be capable of object recognition, instance segmentation, and 6D pose estimation. The grasping and manipulation of these objects is based on the exploitation of the previous processed data. The work carried out has led, among other things, to the creation of a framework to quickly generate an image dataset presenting fruits on their tree, with an associated instance segmentation [4]. This tool should feed the supervised learning algorithms used for semantic segmentation. This segmentation is one of the input components of the model that will need to be trained for the grasping task.

3 Initial learning - introduction to Reinforcement Learning

3.1 Introduction to Reinforcement Learning

The basic idea of RL is *learning what to do*. How an agent should interact with its environment in order to maximize a reward signal. To become familiar with this field, several notions are essential.

3.1.1 Fundamental elements

Environment The black box that receives the actions and modifies its internal state according to the actions (e.g., our world).

Agent The one who interacts with the environment by taking actions (e.g., you). The interaction of the agent and the environment takes place at each discrete timestep $t \in \mathbb{N}$.

State It is a set of variables that describe the environment (e.g., the red car is parked on that street, the player own that item). The set of possible states is denoted $\mathcal{S}^+$. A state can be terminal, which means that the interaction process is finished when the environment is in that state. The set of non-terminal states is denoted $\mathcal{S}$.

Action Action taken by the agent to interact with the environment. The action changes the internal state of the environment. (e.g., move forward, jump). The set of possible action in state $s \in \mathcal{S}$ is denoted $\mathcal{A}(s)$. For the sake of simplicity, it is often stated that actions does not depend on states ($\mathcal{A}$ constant on $\mathcal{S}$). It is therefore denoted just $\mathcal{A}$.

Reward After each action, the agent receives the next state of the environment and a reward. This is a real number and the agent's goal is to maximize this number. The set of possible rewards is denoted $\mathcal{R} \subset \mathbb{R}$.

Markov Decision Process A Markov Decision Process (MDP) is a mathematical object that formalizes the process of interaction between the agent and the environment[5]. At timestep t, the environment is in state $S_t \in \mathcal{S}$. The agent take the action $A_t \in \mathcal{A}(s)$. The environment changes its internal state according to its probability law : $S_{t+1}, R_{t+1} \sim p(r, s' \mid s, a)$. And the process begins again. The strong assumption is made that, in an MDP, the future is independant of the past given the present.

[5]The name Markov Decision Process comes from the Russian mathematician Andrey Markov. MDP are in a way an extension of Markov chains.

Markov Decision Process and Reinforcement Learning The RL problem is largely based on MDPs. There are, however, two major, often overlooked differences between MDPs and a RL problem. First, in most RL problems, the internal state of the environment is not fully known, and the agent only has access to the visible fraction of the MDP (called observation and denoted O_t). Considering that they are quite similar, one will therefore often confuse observation and state ($O_t \approx S_t$). Second, the environment does not return a reward. An interpreter is often used to deduce a reward from the observation. For example, in the case of a robot arm that wants to grasp an object, the palpable world is observable with cameras, sensors... but is never completely known.

Exploration versus exploitation RL widely differs from supervised learning because the agent is not told what is the best baheviour. It has to *explore* its environment by itself and create its experience by is own. Here is the main trade-off of RL : *exploration* and *exploitation*. An agent has to both maximize the reward signal based on its current knowledge of the environment (exploitation) and try new things, in order to discover new sources of reward (exploration).

Policy The agent's behavioral model. Mathematically speaking, it is a function that map states and actions. It is denoted π and can be either *deterministic* or *stochastic*.

- if policy is stochastic, $\pi(a \mid s)$ denotes the probability to choose action $a \in \mathcal{A}(s)$ if the state is $s \in \mathcal{S}$.
- if policy is deterministic, $\pi(s)$ denotes the action choosen if the state is $s \in \mathcal{S}$.

Episode The set of successive states, actions and rewards, until the terminal state :

$$\{S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T\}$$

You might notice that in many cases, the reward of a good action is not immediate. If I play a good chess move, I'm not going to win immediately. Thus, the agent actually tries to maximize the *return* instead of the reward.

Return For a given time t, it is the sum of future discounted reward.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \tag{3.1}$$

Discount factor Denoted $0 \leq \gamma \leq 1$, used to make immediate reward more valuable. $\gamma = 1$ is allowed only if the length of the episodes is bounded from above.

State-value function If the agent is in a state $s \in \mathcal{S}$ and follows the policy π, it can expect to receive a certain return (which will be high if your policy is good). This expectation is the state-value function.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] \tag{3.2}$$

Where the subscript π indicates that all actions are choosen with π.

Action-value function It is like state-value function, except that the next action is not choosen under the policy π.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \tag{3.3}$$

Optimal policy It is a policy that maximizes its value function. In other words, it is a policy that systematically gives the actions that lead to maximum return. Solving an MPD means finding an optimal policy. Let's define the following order relationship.

$$v_\pi \leq v_{\pi'} \Leftrightarrow \forall s \in \mathcal{S} \quad v_\pi(s) \leq v_{\pi'}(s) \tag{3.4}$$

Thus,

$$\pi_* \doteq \arg\max_\pi v_\pi \tag{3.5}$$

Greedy policy Given a action-value function Q, a greedy policy with respect to Q is the policy that chooses the best action according to Q.

$$\text{greedy}(Q) \doteq s \mapsto \arg\max_{a \in \mathcal{A}(s)} Q(s,a) \tag{3.6}$$

Model of the environment It aims for estimating the probability of transition to state s' with reward r, from state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$ denoted

$$p(s', r|s, a) \doteq \Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\} \tag{3.7}$$

A model of the environment allows predictions to be made about how the environment behaves.

Usage of model, value, policy in RL we can use (or not) an explicit model/value function/policy to solve the MDP.

	model	**value function**	**policy**
model-free	✗	-	-
model-based	✓	-	-
value-based	-	✓	✗
policy-based	-	✗	✓
actor-critic	-	✓	✓

Table 2: RL algorithm classes

Planning Using a model to learn, instead of (just) interacting with the environment.

Prediction Evaluate the value function of a given policy.

Control Optimize the policy.

Off-policy and On-policy learning When reinforcement learning is based on interaction with the environment, actions are chosen with what is called behavioural policy, denoted μ. This policy is not necessarily the one that we want to optimize or evaluate (denoted π). If $\pi = \mu$, then the policy to be evaluated is the one used to interract with the environment. The agent *learns on the job*: it is an on-policy algorithm. Otherwise, the agent *looks over someone else shoulder*. It uses a different policy to sample episodes: it is an off-policy algorithm. One of the main advantages of off-policy learning is that every experience can be used: past experience as well as the experience of any another agent.

3.2 Dynamic Programming

Dynamic Programming (DP) describes the set of algorithms for calculating the optimal policy based on the knowledge of a perfect model of the environment. The main idea is to use the value functions defined in equations (3.2) and (3.3). A very interesting property of value functions is that they satisfy a recursive relationship. From any policy π, state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}(s)$, it can be stated that

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \tag{3.8}$$
$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \tag{3.9}$$

These two equations are called the **Bellman expectation equations**. Similar relationships exist for optimal policy:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \mathbb{E}_\pi[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \tag{3.10}$$
$$q_*(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma \max_{a' \in \mathcal{A}(S_{t+1})} q_*(S_{t+1}, a') \mid S_t = s, A_t = a] \tag{3.11}$$

They are called the **Bellman optimality equations**.

3.2.1 Policy evaluation

First, let's see how evaluate a given policy π with the *policy evaluation* algorithm. Equation (3.11) can be rewritten as follows.

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma v_\pi(s')\right) \tag{3.12}$$

The main idea of the algorithm is to iteratively improve the estimation of v_π, denoted V_k. Here is the update rule:

$$V_{k+1}(s) = \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma V_k(s')\right) \tag{3.13}$$

The (V_k) sequence will tend towards its fixed point v_π.

You should notice that the Equation (3.13) uses V_k to update, which is only an approxiation of v_π. Thus, it *learns a guess from a guess*, and the estimation is therefore biased. It's called *bootstrapping*.

The algorithm based on this principle is called *Iterative policy evaluation*.

Algorithm 1: Iterative policy evaluation (for estimating v_π)

Input: π the policy to be evaluated and a model of the environment p
arbitrarily initialize $V(s)$ for all $s \in \mathcal{S}$;
repeat forever
  for $s \in \mathcal{S}$ **do**
    $$V'(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma V(s')\right)$$
  $V \leftarrow V'$

The same principle can also be used for estimating q_π

Algorithm 2: Iterative policy evaluation (for estimating q_π)

Input: π the policy to be evaluated and a model of the environment p
initialize $Q(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$;
repeat forever
  for $s \in \mathcal{S}$ *and* $a \in \mathcal{A}(s)$ **do**
    $$Q'(s, a) \leftarrow \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma \sum_{a' \in \mathcal{A}(s')} \pi(a'|s') Q(s', a')\right)$$
  $Q \leftarrow Q'$

3.2.2 Policy iteration

The idea of policy iteration is to sequentially estimate a policy π and improve it to yield a better policy π'. By starting over a sufficient number of times, you will get π_*. Knowing the action-value function of a policy, the simplest way to find a better policy is to act greedily with respect to its action-value function:

$$\forall s \in \mathcal{S} \quad \pi'(s) = \arg\max_{s \in \mathcal{S}} q_\pi(s, a) \Longrightarrow \pi \leq \pi' \tag{3.14}$$

Algorithm 3: Q-policy iteration (or policy iteration, for estimating π_*)

Input: a model of the environment p
arbitrarily initialize $\pi(s)$ for all $s \in \mathcal{S}$;
while *policy is improving* **do**
  estimate state-value function $Q \approx q_\pi$ with iterative policy evaluation;
  $\pi \leftarrow \text{greedy}(Q)$

3.2.3 Value iteration

The key idea of value iteration is to use the Bellman optimality equation (3.11) deduce this update rule from it:

$$Q_{k+1}(s, a) \leftarrow \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma \max_{a' \in \mathcal{A}(s')} Q_k(s', a')\right) \tag{3.15}$$

By sequentially updating Q, it would get closer to the fixed point of the sequence, i.e., q_*.

Algorithm 4: Q-value iteration (or value iteration, for estimating q_*)

Input: a model of the environment p
arbitrarily initialize $Q(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$;
repeat forever
  for $s \in \mathcal{S}$ *and* $a \in \mathcal{A}(s)$ **do**

$$Q'(s,a) \leftarrow \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma \max_{a' \in \mathcal{A}(s')} Q(s', a') \right)$$

  $Q \leftarrow Q'$

3.2.4 Summary of model-based methods

So far, we have seen one method of DP to do prediction, and two to do control.

Prediction	**Control**
iterative policy evaluation (alg. 1 and 2)	policy iteration (alg. 3) value iteration (alg. 4)

Table 3: Summury of model-free methods based on Dynamic Programming

3.2.5 To get further

Some methods increase the speed of learning by getting rid of useless computation. Here are three of them:

In-place computing systematically uses the most recent value function instead of waiting until the end of the loop to update. Thus, on iterative policy evaluation, the update rule would be:

$$V(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma V(s') \right) \tag{3.16}$$

Prioritized sweeping update some values in priority. For example, we can define how wrong is a value for a state $s \in \mathcal{S}$ with the following norm.

$$\left\| V(s) - \max_{a \in \mathcal{A}} \mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1} \mid S_t = s, A_t = a) \right] \right\| \tag{3.17}$$

Thus, the *wrongest* state-values are updated is priority.

Real-time DP update only relevant states, that have a high probability to occur. By assigning weights to states according to their frequency of occurrence, the value function can be updated with priority over the states that have a high weight, thus saving a lot of non-priority calculations.

3.3 Monte Carlo methods

A very strong assumption is made in DP: there is a reliable model of the environment. However, most of the time this is not the case. A model of the environment is either non-existent or approximate. Monte Carlo (MC) methods does not require a model of the environment: it is a model-free method. It just requires experience. In this section, we study MC methods as the first way to solve a RL problem wihtout needing a model.

3.3.1 Monte Carlo to do prediction

The main idea is to use the return G_t as a approximation of v_π. Let's assume that you have sampled N times your environment and that you have a dataset containing states and the associated return.

$$\langle S_i, G_i \rangle$$

then,

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] \approx \frac{\sum_{i=1}^{N} \mathbb{1}_{S_i=s} G_i}{\sum_{i=1}^{N} \mathbb{1}_{S_i=s}} \tag{3.18}$$

Algorithm 5: Every-visit Monte Carlo (for estimating v_π)

Input: a policy π to be evaluated.
initialize the counter $N(s) = 0$ for each state $s \in \mathcal{S}$;
arbitrarily initialize $V(s)$ for each state $s \in \mathcal{S}$;
repeat forever
  sample episode with π: $S_0, A_0, R_1, \ldots, A_{T-1}, R_T, S_T$;
  for *each timestep t of the episode* **do**
    $N(S_t) \leftarrow N(S_t) + 1$;
    $G_t \leftarrow R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$;
    $V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$

3.3.2 Monte Carlo to do control

Using ε-greedy policy The same algorithm as Algorithm 5 can be used to do control. One way to do this is to approximate the action-value function q_π and then update the policy with the greedy policy with respect to the approximate action-value function. However, for estimating q_π, it is necessary to adopt an exploratory strategy, otherwise certain couples (action, state) would never be visited. The simplest way is to sometimes choose a random action. This is called the ε-greedy policy with respect to Q, where ε is the probability to choose a random action.

$$\varepsilon\text{-greedy}(Q)(s) = \begin{cases} \text{random action } a \in \mathcal{A}(s) & \text{with a probability } \varepsilon \\ \arg\max_{a \in \mathcal{A}(s)} Q(s,a) & \text{with a probability } 1-\varepsilon \end{cases} \tag{3.19}$$

The obtained algorithm is called Greedy in the Limit with Infinite Exploration (GLIE) [5].

Algorithm 6: Greedy in the Limit with Infinite Exploration (for estimating q_*)

arbitrarily initialize $Q(s,a)$ for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$;
$\varepsilon \leftarrow 1$;
$\pi \leftarrow \varepsilon$-greedy$(Q)$;
repeat forever
  $Q \leftarrow \text{MC}(\pi)$ `// use MC for estimating` q_π
  $\pi \leftarrow \varepsilon$-greedy$(Q)$;
  decrease ε ;

It is a simple method. But ε-greedy is not the best way to explore. Instead, you could use Upper Confidence Bounds.

Using Upper Confidence Bounds (UCB) The idea is the give a bonus when the policy explores state it does not know much. This bonus is *upper confidence* $U_t(s,a)$ of a state-action pair. Then, the policy that explores is greedy with respect to $Q+U$:

$$A_t = \arg\max_{a \in \mathcal{A}(S_t)} (Q_t(S_t,a) + U_t(S_t,a)) \tag{3.20}$$

with

$$U_t(s,a) \doteq \sqrt{\frac{-\log p}{2N_t(s,a)}} \tag{3.21}$$

where $p \in]0,1[$ is decreased over time to continue to be explorative.

Algorithm 7: Upper Confidence Bounds (for estimating q_*)

initialize the counter $N(s,a) = 0$ for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$;
arbitrarily initialize $Q(s,a)$ for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$;
initialize $p \in]0,1[$;
initialize $U(s,a) = +\infty$ for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$;
repeat forever
  sample episode $S_0, A_0, R_1, \ldots, A_{T-1}, R_T, S_T$ by acting greedily w.r.t $Q+U$;
  for *each timestep t of the episode* **do**
    $N(S_t,A_t) \leftarrow N(S_t,A_t) + 1$;
    $G_t \leftarrow R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$;
    $Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \frac{1}{N(S_t,A_t)}(G_t - Q(S_t,A_t))$;
    $U(S_t,A_t) \leftarrow \sqrt{\frac{-\log p}{2N_t(S_t,A_t)}}$;
    decrease p ;

3.4 Temporal-Difference Learning

MC methods use G_t as an estimator of $v_\pi(S_t)$. The basic idea of Temporal-Difference (TD) methods is to use another estimation for $v_\pi(S_t)$, from the Bellman expectation equation (Equation (3.8)).

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \quad ((3.8))$$

In TD methods, you want to make the TD-error δ_t as close to 0 as possible:

$$\delta_t \doteq \underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD target}} - V(S_t) \quad (3.22)$$

In addition, since you just use R_{t+1} and S_{t+1} to approximate $v_\pi(S_t)$, you do not need to wait until the end of the episode to learn. You learn *on the fly*.

Important remark: For the sake of simplicity, the algorithms presented in this document do not consider the case where S_{t+1} is terminal. Of course, in practice we cannot bootstrap in this case. So we have to remove the term $\gamma V(S_{t+1})$ from the equation in this case.

	learn from incomplete episode	**variance**	**biased**
MC	no	high	no
TD	yes	low	yes

Table 4: Comparison of Monte Carlo and Temporal-Difference methods

3.4.1 Temporal-Difference to do prediction

The idea here is to use the same principle as Algorithm 5 but to use $R_{t+1} + \gamma V(S_{t+1})$ instead of G_t. This change makes it possible to update the value of V after each interaction with the environment, without having to wait for the end of the episode. The result is presented Algorithm 8.

Algorithm 8: Temporal-Difference (TD(0), for estimating v_π)

Input: a policy π to be evaluated.
arbitrarily initialize $V(s)$ for each state $s \in \mathcal{S}$;
repeat forever
  initialize the environment and get the initial state S_0 ;
  repeat
    choose $A_t \sim \pi(\cdot \mid S_t)$;
    take action A_t and get the reward R_{t+1} and the new state S_{t+1} ;
    $\delta_t \leftarrow R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$;
    $V(S_t) \leftarrow V(S_t) + \alpha\delta_t$
  until *the end of the episode*;

$R_{t+1} + \gamma V S_{t+1}$ is called the 1-step return. Its weakness is that it only looks one step into the future. We therefore introduce the notion of n-step return:

$$\begin{aligned}
&G_{t:t+1} \doteq R_{t+1} + \gamma V(S_{t+1}) && \text{1-step return}\\
&G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) && \text{2-steps return}\\
&G_{t:t+3} \doteq R_{t+1} + \gamma R_{t+2} + +\gamma^2 R_{t+3} + \gamma^3 V(S_{t+3}) && \text{3-steps return}\\
&\dots
\end{aligned}$$

Which approximator is the best ? The larger the n is, the lower the bias and the higher the variance. To find a compromise, we therefore introduce the lambda return:

$$\begin{aligned} G_t^\lambda &\doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n} \\ &= (1-\lambda)\left(G_{t:t+1} + \lambda G_{t:t+2} + \lambda^2 G_{t:t+3} + \cdots\right) \end{aligned} \tag{3.23}$$

You should notice that for $\lambda = 0$, $G_t^0 = R_{t+1} + \gamma V(S_{t+1})$. This is the reason why the previous algorithm is called TD(0).

In the next algorithm, we use an eligibility trace E to spread the reward over the visited states. It's equivalent to use λ-return.

Algorithm 9: Temporal-Difference with λ-return (TD(λ), for estimating v_π)

Input: a policy π to be evaluated.
arbitrarily initialize $V(s)$ for each $s \in \mathcal{S}$;
repeat forever
  initialize eligibility $E(s) = 0$ for each $s \in \mathcal{S}$;
  initialize the environment and get the initial state S_0 ;
  repeat
    choose $A_t \sim \pi(\cdot \mid S_t)$;
    take action A_t and get the reward R_{t+1} and the new state S_{t+1} ;
    $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$;
    for *each state* $s \in \mathcal{S}$ **do**
      $E(s) \leftarrow \lambda\gamma E(s) + \mathbb{1}_{S_t=s}$;
      $V(s) \leftarrow V(s) + \alpha\delta_t E(s)$
  until *the end of the episode*;

3.4.2 Temporal-Difference to do control

As before, we use the Bellman optimality equation to update the approximate optimal action-value function Q. The following algorithm is called State-Action-Reward-State-Action (SARSA)[6].

[6] the name SARSA comes from the fact that every element of the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ is used.

Algorithm 10: State-Action-Reward-State-Action (SARSA(0), for estimating q_*)

arbitrarily initialize $Q(s,a)$ for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$;
$\varepsilon \leftarrow 1$;
repeat forever
  initialize the environment and get the initial state S_0 ;
  choose the first action A_0 by acting ε-greedily w.r.t. Q ;
  repeat
    choose the next action A_{t+1} by acting ε-greedily w.r.t. Q ;
    take action A_t and get the reward R_{t+1} and the new state S_{t+1} ;
    $\delta_t \leftarrow R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$;
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\delta_t$;
    decrease ε ;
  until *the end of the episode*;

Similarly, eligibility traces can be used to replace the 1-step return by the λ-return. The obtained algorithm is called SARSA(λ).

Instead of iteratively evaluate, improve, and so on, Watkins [6] proposed to use the Bellman optimality equation (3.11) to approximate q_*. Here is the new update rule.

$$Q_{k+1}(S_t, A_t) = Q_k(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q_k(S_{t+1}, a) - Q_k(S_t, A_t))$$

This method is called SARSA-max, or Q-learning.

Algorithm 11: Q-learning (for estimating q_*)

arbitrarily initialize Q ;
$\varepsilon \leftarrow 1$;
repeat forever
  initialize the environment and observe the initial state S_0 ;
  repeat
    choose the action $A_t \sim \varepsilon$-greedy(Q) ;
    take action A_t and get the reward R_{t+1} and the new state S_{t+1} ;
    $\delta_t \leftarrow R_{t+1} + \gamma \max_{a \in \mathcal{A}}(Q(S_{t+1}, a)) - Q(S_t, A_t)$;
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\delta_t$;
    decrease ε ;
  until *the end of the episode*;

If Q-learning is responsible of a major breakthrough in RL, it has a important issue: it overestimates the value. The problem comes from improving the policy that is used to sample. One solution is double Q-learning [7]. Double Q-learning uses two value function Q^A and Q^B. You choose randomly which value function is used to sample and the other one is used to improve.

$$a' \leftarrow \arg\max_{a \in \mathcal{A}(S_t)} (Q^A(S_t, a))$$
$$Q^A(S_t, |A_t) \leftarrow Q^A(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q^B(S_{t+1}, a') - Q^A(S_t, A_t)\right) \quad (3.24)$$

or

$$a' \leftarrow \arg\max_{a \in \mathcal{A}(S_t)} (Q^B(S_t, a))$$
$$Q^B(S_t, |A_t) \leftarrow Q^B(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q^A(S_{t+1}, a') - Q^B(S_t, A_t) \right) \quad (3.25)$$

3.5 Off-policy learning

Off-policy learning consists of using a different policy for learning and for sampling. If necessary, the notation of this the policy used to sample is μ.

Off-policy learning enable to learn from any data, regardless of the policy followed to get the data. Many methods are based on replay buffer (or dataset) denoted D that stores past experience. This past experience can be re-used in an off-policy algorithm.

3.5.1 Importance Sampling

One way to transform an on-policy algorithm into an off-policy algorithm is to use Importance Sampling (IS). IS involves multiplying the approximation of the value function by the ratio of the probabilities associated with the actions involved.

	Sampling policy	
	π	μ
MC	G_t	$\prod_{\tau=t}^{T} \frac{\pi(A_\tau \mid S_\tau)}{\mu(A_\tau \mid S_\tau)} \times G_t$
TD	$R_{t+1} + \gamma V(S_{t+1})$	$\frac{\pi(A_t \mid S_t)}{\mu(A_t \mid S_t)} (R_{t+1} + \gamma V(S_{t+1}))$

Table 5: Approximator of $v_\pi(S_t)$ according to the method and policy used for sampling

In practice, IS is never used with MC because it dramatically increases variance. However, it is used with TD. The Algorithm 12 gives an example of this.

Algorithm 12: Importance Sampling for off-policy TD(0) (for estimating v_π)

Input: a policy π to be evaluated.
Input: a policy μ to be followed.
arbitrarily initialize $V(s)$ for each state $s \in \mathcal{S}$;
repeat forever
  initialize the environment and get the initial state S_0 ;
  repeat
    choose $A_t \sim \mu(\cdot|S_t)$;
    take action A_t and get the reward R_{t+1} and the new state S_{t+1} ;
    $\delta_t \leftarrow \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \gamma V(S_{t+1})) - V(S_t)$;
    $V(S_t) \leftarrow V(S_t) + \alpha \delta_t$
  until *the end of the episode*;

3.5.2 Expected SARSA and Generalized Q-learning

Sequences are sampled by following μ and the TD-error becomes

$$\delta_t = R_{t+1} + \gamma \underbrace{\sum_{a \in \mathcal{A}} \pi(a \mid S_{t+1}) Q(S_{t+1}, a)}_{V(S_{t+1})} - Q(S_t, A_t) \tag{3.26}$$

If $b = \pi$, then this is on-policy learning and the obtained algorithm is called **expected SARSA**. If $b \neq \pi$, then it is off-policy learning and it is called **generalized Q-learning**

You should notice that Q-learning (Algorithm 11) is generalized Q-learning with

$$\pi = \text{greedy}(Q)$$
$$b = \varepsilon\text{-greedy}(Q)$$

3.5.3 Summury of model-free methods

So far, we have seen several value-based methods for doing control and prediction without needing a model of the environment. The Table 6 summurizes these algorithms.

Prediction		Control	
On-policy	**Off-policy**	**On-policy**	**Off-policy**
MC		GLIE	
TD(0)	IS for TD(0)	SARSA	Q-Learning
TD(λ)	IS for TD(λ)	SARSA(λ)	

Table 6: Summary of model-free methods

3.6 Deep Reinforcement Learning

The algorithms seen so far work well for environments with a very limited number of states. Beyond a certain threshold, what Richard Bellman calls *the curse of dimensionality* occurs. It means that the computational requirements grows exponentially with the number of states. In many applications that are considered for RL, the state space is huge: it can correspond to data from cameras, or from many sensors. It is impossible for an agent to explore all combinations of pixels (to take the example again) . This is why an alternative to tabular methods must be found. This alternative is Artificial Neural Network (ANN).

The advent of ANNs has created a real revolution in the field of RL. ANNs are fantastic function approximators. They are able to approximate very complex functions, such as those encountered in RL and generalize very well for neighboring states. ANNs architecture will not be discussed in this report. They often have little influence on the results. As Andrej Karpathy says[7]:

[7] https://twitter.com/karpathy/status/822563606344695810

Everything I know about design of ConvNets (resnets, bigger=better batchnorms etc.) is useless in RL. Superbasic 4 layers ConvNets works best.

DRL is RL using ANNs as approximators of functions. The use of ANNs is based on the minimization of a cost function that tells us the distance between the data and the function to approximate. For example, in supervised learning, the goal is to adjust the parameters of the ANN in order to associate the input data with the right label. Using labeled data, the cost function is the gap between the right label and the one predicted by the ANN. In RL, there is no labeled data : we want the agent to learn by itself and not reproduce the behavior of someone else. Thus ANN are not used exactly the same way. First, let's see how to use ANN to approximate value functions.

3.6.1 Value-based Deep Reinforcement Learning

The approximate state-value function (resp. action-value function) is parametrized with $\mathbf{w}$ and is denoted $v_{\mathbf{w}}$ (resp. $q_{\mathbf{w}}$).

Several cost functions to minimize can be choosen. Probably the most intuitive ones are

$$L(\mathbf{w}) = \|G_t - v_{\mathbf{w}}(S_t)\|^2 \tag{3.27}$$

for MC or

$$L(\mathbf{w}) = \|R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)\|^2 \tag{3.28}$$

for TD(0).

Since $v_{\mathbf{w}}$ is differentiable, gradient-based minimazation can be applied. For example, stochastic gradient descent is based on the following update rules.

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \frac{\alpha}{2}\nabla_{\mathbf{w}} L(\mathbf{w}) \\ &\leftarrow \mathbf{w} + \alpha(G_t - v_{\mathbf{w}}(S_t))\nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) \end{aligned} \tag{3.29}$$

for MC and

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(\underbrace{R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)}_{\delta_t})\nabla_{\mathbf{w}}(v_{\mathbf{w}}(S_t) - \gamma v_{\mathbf{w}}(S_{t+1})) \tag{3.30}$$

for TD(0)

In practice, Equation (3.30) is not used because it violate causality principle: the future is independent of the past, knowing the present, so the value function of the next state cannot depend on a reward arrived before. This is why, in practice, we use the following formula, which is called the *semi-gradient*.

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta_t \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t) \tag{3.31}$$

The above calculations are also valid for the action-value function. Using the previous equations, the algorithms 13 and 14 are the versions of SARSA and Q-learning based on function approximators.

Algorithm 13: Episodic semi-gradient SARSA (for estimating q_*)

Input: a differentiable action-value function $q_{\mathbf{w}} : (\mathcal{S}, \mathcal{A}) \to \mathbb{R}$ parametrized with $\mathbf{w}$
randomly initialize $\mathbf{w}$;
$\varepsilon \leftarrow 1$;
repeat forever
  initialize the environment and get the initial state S_0 ;
  choose the action $A_0 \sim \varepsilon$-greedy$(q_{\mathbf{w}}(S_0, \cdot))$;
  repeat
    take action A_t and get the reward R_{t+1} and the new state S_{t+1} ;
    choose the next action $A_{t+1} \sim \varepsilon$-greedy$(q_{\mathbf{w}}(S_{t+1}, \cdot))$;
    $\delta_t \leftarrow R_{t+1} + \gamma q_{\mathbf{w}}(S_{t+1}, A_{t+1}) - q_{\mathbf{w}}(S_t, A_t)$;
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta_t \nabla_{\mathbf{w}} q_{\mathbf{w}}(S_t, A_t)$;
    decrease ε ;
  until *the end of the episode*;

Algorithm 14: Online Deep Q-learning (for estimating q_*)

Input: a differentiable action-value function parametrization $q_{\mathbf{w}} : (\mathcal{S}, \mathcal{A}) \to \mathbb{R}$ parameterized with $\mathbf{w}$
randomly initialize $\mathbf{w}$;
$\varepsilon \leftarrow 1$;
repeat forever
  initialize the environment and get the initial state S_0 ;
  repeat
    choose the action $A_t \sim \varepsilon$-greedy$(q_{\mathbf{w}}(S_t, \cdot))$;
    take action A_t and get the reward R_{t+1} and the new state S_{t+1} ;
    $\delta_t \leftarrow R_{t+1} + \gamma \max_{a \in \mathcal{A}}(q_{\mathbf{w}}(S_{t+1}, a)) - q_{\mathbf{w}}(S_t, A_t)$;
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta_t \nabla_{\mathbf{w}} q_{\mathbf{w}}(S_t, A_t)$;
    decrease ε ;
  until *the end of the episode*;

Online Deep Q-learning is unstable in practice because the target

$$\delta_t \leftarrow R_{t+1} + \gamma \max_{a \in \mathcal{A}}(q_{\mathbf{w}}(S_{t+1}, a)) - q_{\mathbf{w}}(S_t, A_t)$$

is continuously changing with each iteration. To solve this problem, Mnih et al. [8] introduced two new ingredients which are a target network and experience replay. The target network, denoted by parameters $\mathbf{w}^-$, is an old version of the online network with parameters $\mathbf{w}$. Every τ timesteps, the online parameters are copied into the target parameters ($\mathbf{w}^- \leftarrow \mathbf{w}$). Experience replay, initially introduced by Lin [9] is storing observed transitions some time and sampled uniformly from this buffer to update the network. The resulting algorithm is called Deep Q-Networks (DQN).

Algorithm 15: Deep Q-Networks (for estimating q_*)

Input: a differentiable action-value function parametrization
$q_{\mathbf{w}} : (\mathcal{S}, \mathcal{A}) \to \mathbb{R}$ parameterized with $\mathbf{w}$
randomly initialize $\mathbf{w}$ and $\mathbf{w}^-$ such that $\mathbf{w} = \mathbf{w}^-$;
$\varepsilon \leftarrow 1$;
initialize a empty dataset D ;
repeat forever
  initialize the environment and get the initial state S_0 ;
  repeat
    choose the action $A_t \sim \varepsilon$-greedy$(q_{\mathbf{w}}(S_t, \cdot))$;
    take action A_t and get the reward R_{t+1} and the new state S_{t+1} ;
    store the transition $(S_t, A_t, R_{t+1}, S_{t+1})$ into the dataset D ;
    uniformly sample minibatch (s, a, r, s') from D ;
    $\delta_t \leftarrow r + \gamma \max_{a' \in \mathcal{A}}(q_{\mathbf{w}^-}(s', a')) - q_{\mathbf{w}}(s, a)$;
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta_t \nabla_{\mathbf{w}} q_{\mathbf{w}}(s, a)$;
    every τ timesteps, $\mathbf{w}^- \leftarrow \mathbf{w}$;
    decrease ε ;
  until *the end of the episode*;

Unlike the tabular methods, bootstrapping can, under some conditions, cause divergence.

Method	Table lookup	Non-linear
MC	✓	✓
TD	✓	✗
MC	✓	✓
TD	✓	✗

Table 7: When using non-linear function approximator, bootstrapping can cause divergence

3.7 Policy Gradients and Actor Critics

Model-based methods have the advantage of capturing all the information that the data can provide. However, they suffer from the fact that most of the information is not really relevant for maximizing the reward function. Value-based methods are closer to the real objective. Since it is a matter of matching states with their values, it is a kind of regression problem. Nevertheless, these methods also capture a lot of irrelevant information. In some cases, the optimal policy is very simple, but the value function can be extremely complex. In this case, value-based methods have great difficulty in learning properly. To get as close as possible to the true goal, policy-based methods can be used. The learning method consists of increasing the probability of the actions that give a high reward to occur, without trying to calculate that reward. Then, it is possible to combine value-based method and policy-based method. It is called actor-critic

3.7.1 Policy-gradient

First, let us define a stochastic[8] policy, parameterized with $\boldsymbol{\theta}$ and denoted $\pi_{\boldsymbol{\theta}}(a \mid s)$ for all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$. An often used policy is the soft-max policy. It is defined as follows[9].

$$\pi_{\boldsymbol{\theta}}(a \mid s) = \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{a' \in \mathcal{A}} e^{h(s,a',\boldsymbol{\theta})}} \tag{3.32}$$

Where $h(s, a, \boldsymbol{\theta})$ is the preference for selecting action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ based on $\boldsymbol{\theta}$.

The objective function, which means the function we want to maximize, can be defined in many different ways. For the sake of simplicity, let us assume that an episode always start in the same state s_0. The following results remain valid without this assumption, but it dramatically simplify the calculations.

$$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0) \tag{3.33}$$

Defined that way, the objective function J is the expected return under the policy $\pi_{\boldsymbol{\theta}}$. The aim is to find $\boldsymbol{\theta}$ that maximize $J(\boldsymbol{\theta})$.

Policy Gradient method is based of the stochastic gradient ascent.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{3.34}$$

To find an expression of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ that can be sampled, we use what it is sometimes called the *likelihood ratio trick*. The demonstration is not be detailled here. A detailed version is given in [1]. We are just admitting that

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[G_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)] \tag{3.35}$$

Algorithm 16: Monte Carlo Gradient (or REINFORCE, for estimating π_*)

Input: a differentiable policy $\pi_{\boldsymbol{\theta}}$ parametrized by $\boldsymbol{\theta}$
randomly initialize $\boldsymbol{\theta}$;
repeat forever
  sample $\{S_0, A_0, R_1, \ldots, A_{T-1}, R_T\} \sim \pi_{\boldsymbol{\theta}}$;
  for *each time* $t \leq T$ *of the episode* **do**
    $G_t \leftarrow R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$;
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left(\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)\right) G_t$

REINFORCE works on the principle we have just explained: by increasing the likelihood that the policy will take an action as much as the reward it has produced. But, depending on the nature of the reward function, the results can be very different. Take the example of the inverted pendulum. One way to define the reward function is to give a negative reward if the pendulum is down, and a zero reward if the pendulum is up. Thus, using REINFORCE, the probability of all actions will tend irreparably towards 0, which is undesirable.

[8] In practice, stochastic policies are prefered since most environment are partially observable

[9] Note that the denominator is just what is a normalization value used so that the action probabilities in each state sum to 1.

To solve this issue, a value called baseline and denoted $b(s)$ is subtracted from the return[10]. The REINFORCE algorithm with baseline algorithm would use

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left(\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)\right) \left(G_t - b(s)\right) \tag{3.36}$$

What is a good baseline ? A good one is presented in the next section.

3.7.2 Actor-Critic

A good baseline is $v_{\pi_{\boldsymbol{\theta}}}(s)$ since it represents the expected reward from state $s \in \mathcal{S}$. If an action leads to a better reward than expected, one can increase its probability to occur. But, it requires to have an estimation of $v_{\pi_{\boldsymbol{\theta}}}(s)$. To estimate the value of $v_{\pi_{\boldsymbol{\theta}}}$, one can use all the value-based methods seen above, based on MC and TD. Let's denote $v_{\mathbf{w}}$ the parameterized value function that aim to approximate $v_{\pi_{\boldsymbol{\theta}}}$. If you use the one-step return instead of the full return, $\boldsymbol{\theta}$ is updated as following:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left(\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)\right) \left(R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)\right) \tag{3.37}$$

The corresponding algorithm is called Advantage Actor Critic (A2C).

Algorithm 17: Advantage Actor-Critic (A2C, for estimating π_*)

Input: a differentiable actor π parametrized by $\boldsymbol{\theta}$
Input: a differentiable critic $v_{\mathbf{w}}$ parametrized by $\mathbf{w}$
randomly initialize actor parameters $\boldsymbol{\theta}$;
randomly initialize critic parameters $\mathbf{w}$;
repeat forever
  initialize the environment and get the initial state S_0 ;
  repeat
    choose action $A_t \sim \pi_{\boldsymbol{\theta}}(\cdot|S_t)$;
    take action A_t and get the reward R_{t+1} and the new state S_{t+1} ;
    $\delta_t \leftarrow R_{t+1} + \gamma v_{\mathbf{w}}(S_{t+1}) - v_{\mathbf{w}}(S_t)$;
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \delta_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)$;
    $\mathbf{w} \leftarrow \mathbf{w} + \beta \delta_t \nabla_{\mathbf{w}} v_{\mathbf{w}}(S_t)$
  until *until the end of the episode*;

A2C suffers from the same issue as Online Deep Q-learning (Algorithm 14) : the target is continuously changing with each iteration. To solve this problem, Lillicrap et al. [10] introduced DDPG (Algorithm 18) that exploits the same idea used for DQN (Algorithm 15) : experience replay and target network[11]. DDPG can only be used in continuous action spaces. To encourage exploration, an action noise $\mathcal{N}$ is added. The resulting algorithm is called DDPG and is presented Algorithm 18. Note that the algorithm uses a deterministic policy π_θ.

[10] Substracting a baseline is mathematically correct. Indeed, the value of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ remains unchanged

[11] The idea of the target network has been slightly modified: instead of freezing the target network $\boldsymbol{\theta}'$ for a certain number of timesteps, DDPG performs a weighted average (with τ) at each iteration between the target network $\boldsymbol{\theta}'$ and the current network $\boldsymbol{\theta}$.

Algorithm 18: Deep Deterministic Policy Gradient (for estimating π_*)

Input: a differentiable actor π parametrized by $\boldsymbol{\theta}$
Input: a differentiable critic $q_{\mathbf{w}}$ parametrized by $\mathbf{w}$
randomly initialize actor parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^-$ such that $\boldsymbol{\theta} = \boldsymbol{\theta}^-$;
randomly initialize critic parameters $\mathbf{w}$ and $\mathbf{w}^-$ such that $\mathbf{w} = \mathbf{w}^-$;
initialize a empty dataset D ;
repeat forever
  initialize a random process $\mathcal{N}$ for action exploration;
  initialize the environment and get the inital state S_0 ;
  repeat
    select action $A_t = \pi_{\boldsymbol{\theta}}(S_t) + \mathcal{N}$;
    take action A_t and get the reward R_{t+1} and the new state S_{t+1} ;
    store transition $(S_t, A_t, R_{t+1}, S_{t+1})$ in D ;
    sample a random minibatch of N transitions (s_i, a_i, r_i, s'_i) from D ;
    $\delta_i \leftarrow r_i + \gamma q_{\mathbf{w}^-}\left(s'_i, \pi_{\theta^-}(s'_i)\right) - q_{\mathbf{w}}(s_i, a_i)$;
    update $\mathbf{w}$ to minimize $L = \dfrac{1}{N}\displaystyle\sum_{i=1}^{N} \delta_i^2$;
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ with
    $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \dfrac{1}{N}\displaystyle\sum_{i=1}^{N} \nabla_a \, q_{\mathbf{w}}(s,a)|_{\substack{s=s_i \\ a=\pi_{\boldsymbol{\theta}}(s_i)}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s)|_{s=s_i}$;
    $\boldsymbol{\theta} \leftarrow \tau\boldsymbol{\theta}' + (1-\tau)\boldsymbol{\theta}$;
    $\mathbf{w} \leftarrow \tau\mathbf{w}' + (1-\tau)\mathbf{w}$;
  until *the end of the episode*;

The data collected by the algorithm depends on the policy being followed. If the policy suddenly becomes poor, the collected data will become poor too, and the policy will unlearn what it was good at doing before. A frequently used solution is to subtract the Kulleman-Leibler (KL) divergence from the objective function. The KL divergence is a distance between distributions. The obtained algorithm was introduced by Schulman et al. [11] and was called Trust Region Policy Optimization (TRPO).

$$D_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}_k} || \pi_{\boldsymbol{\theta}_{k+1}}) \doteq \mathbb{E}\left[\int_{a\in\mathcal{A}} \pi_{\boldsymbol{\theta}_k}(a|S) \log \frac{\pi_{\boldsymbol{\theta}_{k+1}}(a|S)}{\pi_{\boldsymbol{\theta}_k}(a|S)} \mathrm{d}a\right] \tag{3.38}$$

The new objective function to maximize is $J_{\mathrm{KL}}(\boldsymbol{\theta})$.

$$J_{\mathrm{KL}}(\boldsymbol{\theta}_{k+1}) \doteq J(\boldsymbol{\theta}_{k+1}) - \eta D_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}_k} || \pi_{\boldsymbol{\theta}_{k+1}}) \tag{3.39}$$

where η is the KL penaly and is taken small.

In doing so, from one iteration to the next, the policy *does not change too much*. This solution works well in practice. The measure of divergence is nevertheless a little complicated to calculate. Proximal Policy Optimization (PPO) is an alternative proposed by Schulman et al. [12] that simplifies the objective function and gives similar results.

4 Subject matter grasping

The application of DRL in robotic manipulation has been widely studied in recent years. Effective control of robotic systems is a challenge due to their high dimensionality. DRL provides a model agnostic approach to control such complex dynamic systems. The control of soft object manipulation is one component of the general object manipulation issue. The mastery of this component requires the synergy of other specific skills. In this section, we first present the recent work in terms of robotic learning. Then we focus of the specific skills that improves the mastery of these robotic manipulation, and more specifically soft object manipualtion.

4.1 Robotic learning

Previous work has tried to solve this problem on robotic learning in many ways, and recent learning-based works show encouraging results [13]. Nevertheless, these successes are relative: the tasks are often very specific and require either complex engineering upstream or a very long learning curve. The big question of reinforcement learning applied to object manipulation is how to learn a generalized policy.

An interesting idea could be to artificially reduce the space of possible states by dividing the space into two regions: one in which the model is reliable, and one in which the model is not. It is the idea developped by Lee et al. [14]: they combine the strengths of model-based methods (by model they mean detailed description of how to perform the task, not derived from learning) with the flexibility of learning-based methods. In uncertain regions, a locally learned policy is used. They called their method GUAPO (Guided Uncertainty-Aware Policy Optimization). However, this approach is not entirely satisfactory since it requires a dynamic model created by hand for certain regions.

Let's take the example of a task that consists of picking fruit from a bin. We would like the resulting policy to be able to properly grasp the fruit without dropping or damaging it. Ideally, the policy should also be able to adopt high-level behaviors, such as anticipating which fruit to grasp first or identifying and moving an inconvenient fruit to grasp another. One of the method that give good results is Hierarchical Reinforcement Learning (HRL) [15]. It involves dividing a policy into several policy levels. Levels of abstraction of decisions and time horizons evolve increasingly with the hierarchical level of the layer. A similar idea is also to consider a high-level task as a structured succession of low-level tasks. Thus, specific capabilities such as object recognition are trained upstream of the robotic learning phase. The question to be aswered is: does having priors knowledge facilitate learning to perform manipulation ? A first work led by Piergiovanni et al. [16] suggest that the capability of predicting the future observations (based on an autoencoder) significantly helps training. Another work led by Yen-Chen et al. [17] aimed to evalute the gain of a prior training of the model on a passive vision task. When they adapted it to perform an active manipulation task, the result showed that pre-training on vision tasks significantly improves generalization and sample efficiency for learning to manipulate objects. They obtained very encouraging results since the success rate for a suction task on unseen objects was 91% with only 10 minutes of training.

Another approach is to realize that our innate ability to adopt complex behaviors may well help robots in their learning phase. By showing them how a human being would do this task, the robot would be guided to a policy base that successfully performs the task. This method is called learning by demonstration. Rajeswaran et al. [18] applied learning from demonstration on a multi-fingered hand and used DRL to learn. They showed that model-free DRL can effectively scale up to complex manipulation tasks with a high-dimensional 24 degrees of freedom hand, and solve them from scratch in simulated experiments. Plus, learning from demonstrations significantly improves improves the learning capabilities. With just a few hours of robot experience, the obtained policy is very robust and exhibit very natural movements.

4.2 Tactile perception in a manipulation task

Manipulating objects does not involve just learning. The performance of the grasping task also depends on other parameters such as the ability to feel objects or to make reliable grasping contact with them with a suitable gripper. A suitable grasping organ is required, and an example of an efficient gripper resulting from many years of improvements gives us a good basis for our work: our hand. Many robotic grasping organ are inspired by it [19, 20]. It is also possible to add new capabilities to the gripper. , such as suction. Chin et al. [21] proposed a gripper that combine three gripping modes: suction, parallel jaws and flexible fingers. They showed that conbining these capabilities, such a gripper can perform complex tasks. They acheived to grasp 88% of the objects tested, 14% of which required a combination of grip modes to be grasped.

The ability to feel objects is also a key point in grasping and is inspired by the tactile perception of human beings. This task is performed by tactile sensors. Some works show that taking inspiration from the tactile perception of humans gives good results [22]. Having at first a low number of output data ([23] classic tactile sensor six-axis force/torque), the sensors proposed in recent work are constantly increasing the number and reliability of measurements. PapillArray is an incipient slip sensor introduced by Khamis et al. [24]. They thought that not only tactile forces are important in manipulation task but also friction and the occurrence of incipient slip. Since it is human-inspired it should help to improve manipulation tasks.

Recent work also shows that the performance of these tactile sensors can greatly benefit from advances in deep learning. For example, Fleer et al. [25] were based on organisation of human haptic search behavior, to propose an haptic sensor that uses Recurent Attention Model to explore an object in order to identify objects. Sferrazza et al. [26] also took advantage of recent progress in computer vision and introduced a vision-based tactile sensor based.

Basing the learning of a manipulation task on tactile sensors gives good results. Chebotar et al. [27] trained a model that predict the grasp outcome (whether the grasp task is a success or not). They used they predictor outcome to provide a feedback to the DRL algorithm (that also uses spatio-temporal features extracted from a biomimetic tactile sensor) to estimate the required grasp adjustment and do the regrasping. Allowing 3 regrasps, they were able to get improve the sucess rate on the grasping task from 40.2% to 92.1%

4.3 Grasping what is deformable

The challenge of manipulating soft objects is not such a recent question [19]. As we have just seen, tactile sensors help to better perceive the object to be manipulated and therefore help a lot for a grasping task. This help is all the more essential to perceive the deformation of an object. The work of Sanchez et al. [28] shows that using both the output of a tactile sensor and the deformation model, it is possible to predict the shape of the soft object online. However, when objects and gripper becomes complex, the work necessary to compute a good approximation of the shape of the object becomes very high. Some work focused on making good approximation to lighten the computation: Luo and Xiao [29] modelled the interactions between a rigid and an elastic object by taking into account forces, deformations and frictions. They introduced a new approximation of material deformation adapted to interactive environments. Their approach is valid even when there are multiple contacts. The computing means of that time (2007) allowed them to go up to 1 kHz.

Although the manipulation of soft objects is more complex than the manipulation of hard objects, there are nevertheless some examples of works that have been implemented and that obtain good results. Matas et al. [30] first use DRL on deformable object manipulation. They train the model on simulation, and acheived to deploy it successfully in the real world with domain randomisation. The model performed well even with unseen objects.

4.4 Simulation to real gap

Learning complex policies such as grasping an object requires a lot of training due to data inefficiency of widely-used DRL methods. Most of the best results are obtained using several million steps. [31, 32]. This number is far too high for this training phase to be done on physical systems. This is why the preferred method is to train the robot arm in simulation. The simulation of a trajectory takes much less time than the duration of this trajectory with physical robot. Moreover, in simulation, we can afford to distribute the training. Simulation thus allows to obtain a training capacity much higher than in the physical systems.

The natural question is: once the model is trained in simulation, is it possible to deploy it easily on physical systems? In fact, if certain precautions are not taken, it does not work. This is called the simulation-to-real gap[12]. The gap between the simulation and the real system in terms of dynamics and perception causes a catastrophic drop in performance. Most of the time, even an extremely simple task that is well learned in simulation will not be correctly performed on a physical systems.

To tackle this issue, two methods are often used. The first is to increase the quality of the simulation to get as close as possible to reality. Since observation is often based on cameras, GANs can be used to increase the visual fidelity of the simulated images. The second approach is to make the controller robust to changes in system properties. If the controller is robust to system variations, it will also be robust to system variations when moving from simulation to reality, thus allowing a better transfer. Randomization of certain variables in

[12] or *sim2real*

the simulation can be used to achieve this robustness. For example, one can randomize the dynamics of the system, use a stochastic policy or add noise to the observations. In practice, the two methods are used together [33].

Making the simulation more realistic can be achieved by using self-supervised learning. Recent papers have shown that very good results can be obtained: Kahn et al. [34] combined the approach of using an internal world map (a computationally intensive operation that does not allow for learning from failures) and the learning-based approach (learning from failures but difficult to deploy because of the high complexity of the samples). For this, they introduced a generalized computation graph that uses self-supervised learning to train the model. They evaluate their approach on a real-world remote car and show that it can learn to navigate in a complex indoor environment with a 4-hour fully autonomous training. Hwangbo et al. [35] used self-supervised learning based on real data to simulate the actuator (that map actions to torque) of their legged robot. The training process lasts a several hours on an ordinary desktop PC. Without any modification of filtering, the trained policy was deployed on their real robot and achieved impressive results: it ran faster than ever before, used less energy, and it had never been checked when he had to recover from falling, even in very complex configurations. In a paper from Jeong et al. [36] it is showed that self-supervised domain adaptation (SSDA) method, which uses unlabeled real robot data outperform domain randomization.

4.5 Hindsight Experience Replay

One of the most important challenges of reinforcement learning is to find a compromise between exploration and exploitation to make the training as effective as possible. This challenge is particularly complicated when environments have sparse rewards. To guide the agent to the high reward zones, several methods have been tried [37]. HER was introduced in 2017 by Andrychowicz et al.. The key idea of HER is to see a fail as a success, but with another goal.

To get the intuition, suppose you throw a basketball but miss the basket. If the basket had been where you threw the ball, your shot would have been a hit. In a learning process, you can add this fictitious success to your experience (e.g., a replay buffer). This is the main idea behind HER: it adds simulated hits to the experience.

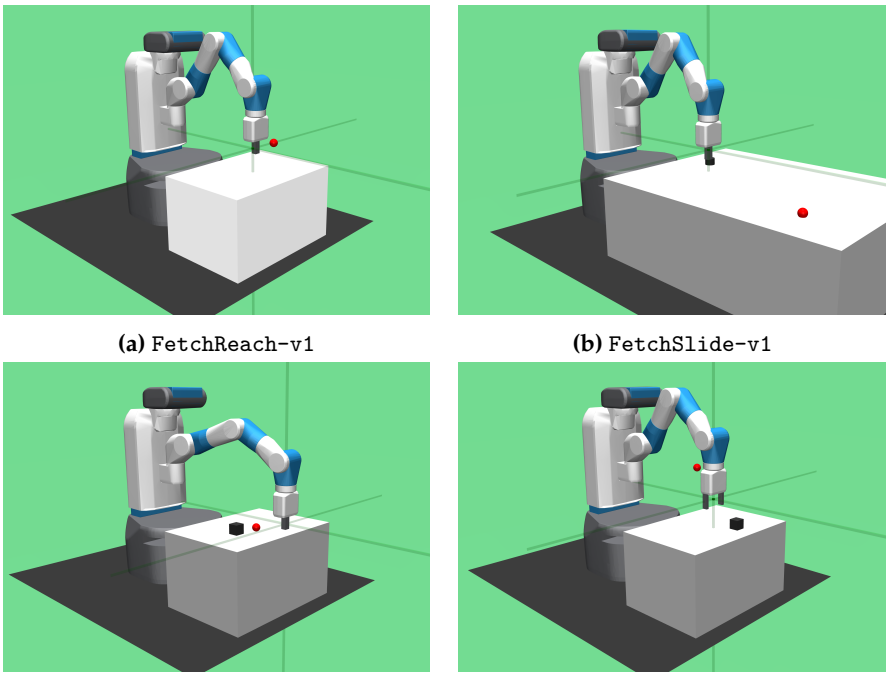
The effectiveness of this approach has been seen in many robotic environments [39]. Even with a binary reward function (success or failure), HER shows very promising results. For complex environments and tasks such as those we encounter in robotics, HER could save us the tedious work of building a good reward function. A simple hit/miss function could be sufficient.

Several strategies were proposed in the original paper to create those artificial goal. The most effective one is called *future*. For a given timestep in an episode, the *future* strategy consist in creating k artifical goals picked randomly from the future states of the same episode.

5 Material and method

5.1 OpenAI/gym Fetch environments

The approach of this work consists of an iterative increase in complexity. First, the experiments will be done on basic robot arm simulation environments. OpenAI/gym [20] proposes a set of 4 robotics environments (Figure 2 to give a starting point to our experiments. These 4 environments for Gym are based on the MuJoCo physics simulation engine. Each environment consists of a robot before performing an elementary task: reaching a point, sliding a cube to a given position, pushing a cube to a given position, picking and placing a cube to a given position.



(a) `FetchReach-v1` **(b)** `FetchSlide-v1`

(c) `FetchPush-v1` **(d)** `FetchPickAndPlace-v1`

Figure 2: Fetch environments

For each of these environments, two reward functions are available: a dense and a sparse. The sparse reward function can take two values : 1 when the task is acheived, and 0 otherwise. The dense reward function can take intermediate values (between 0 and 1) depending how close from the success the agent is. For exemple, for the reaching task, the dense reward function would depend on the distance between the gripper and the goal.

5.2 The Panda robot and its simulator

Once good results have been obtained on the simple OpenAI environments, the objective will be to transfer them to the following simulator. The robot used for the subject is the robot arm PANDA from FRANKA EMIKA (Figure

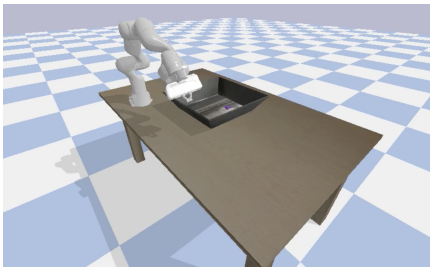
3). It has 7 degrees of freedom, with a payload of 3 kg and a reach of 850 mm. Its grasping organ is a gripped with two parallel fingers. It has a continuous gripping force of 70 N and a stroke of 80 mm.

(a) Panda robot

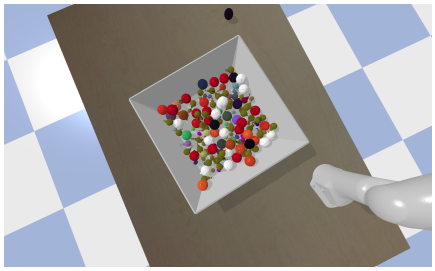
(b) Panda robot in its environment

Figure 3: Panda robot

The training phase will be done in simulation. As explained previously, training in simulation makes it possible to generate many robots in parallel and thus to produce much more data than on a physical robot. In order for the trained model to be deployed on the real robot, the gap between simulation and reality must be anticipated. This is why special attention is paid to the realistic nature of the data produced by the simulator. Indeed, in these real conditions, the light is likely to change, there may be shadows, the fruits are not all the same in shape, weight or texture etc.. The simulator will have to be able to simulate camera renderings faithful to reality and reproduce all these variations in order to limit the gap between simulation and reality as much as possible. These realistic variations also affect other sensors, such as torque sensors or the tactile sensor. Work is in progress to create this realistic rendering simulator. A first version based on the PyBullet [40] physics simulation engine is shown in Figure 4.



(a) Simulated Panda robot



(b) Simulated fruit basket

Figure 4: Panda robotic arm simulation with PyBullet

A second version of the simulator is under construction. This new version should allow faster image rendering and thus increase the simulation speed. This simulation speed is an essential component of the learning process. If the simulation is not fast enough, the algorithm will not be able to learn in a reasonable time. This second version is developed thanks to the ROS[13] and Gazebo[14] frameworks. An overview of the current rendering is presented Figure 5

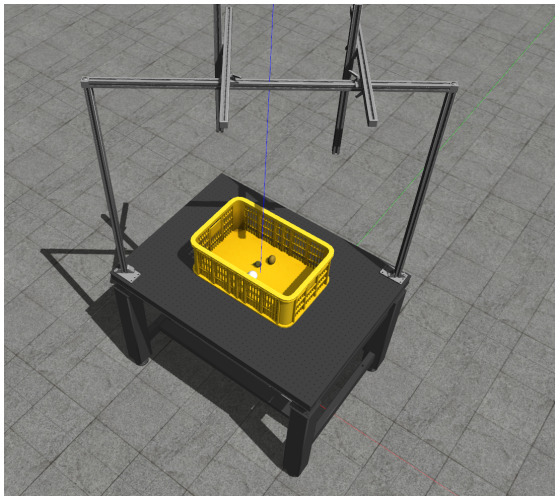


Figure 5: Simulator based on Gazebo

Eventually, this simulator will have to be able to integrate all the essential components of our problem: a realistic rendering of the cameras and other sensors, a simulation of deformable fruits which will have to manage collisions with other objects in the scene (cage, other fruits, etc.).

5.3 Calculation equipment

The configuration of the computer used to do the training is shown in Table 8.

[13] `https://www.ros.org`
[14] `http://gazebosim.org`

Computer	
Operating system	Ubuntu 18.04.5 LTS
OS Type	64 bits
Memory	31.3 Gb
Processor	Intel® Core™ i7-6700HQ CPU @ 2.60Ghz × 8
Graphics card	GeForce GTX 1070/PCIe/SSE2
Software versions	
Python	3.7.8
tensorflow-gpu	1.14
mujoco-py	2.0.2.11
stable-baselines[mpi]	2.10.0
baselines	0.1.6
cuda	10.1

Table 8: Computation equipments

6 Experiments and results

The following experiments evaluate some learning principles found in the literature that could be applied to soft object grasping. Many architectural choices can affect the quality of learning. As we have seen in the Section 4, one can play with the state space, the action space, the task hierarchy, the reward function, the learning algorithm, the reliability of the simulation, the adaptation of domains to move from simulation to reality etc. These aspects all deserve attention, and the development of an overall learning strategy will need to take all these issues into consideration. In this work, priority is given to evaluating learning strategies in simulation. Here we adopt an iterative approach where the complexity of the environment is increasing. We start with a simple robot arm environment that must learn to reach a point with its gripper. In this environment, we evaluate how an elementary DRL algorithm (DDPG, Algorithm 18) behaves (Section 6.1). Then the task becomes more complex: pushing, sliding and finally grasping. As the complexity of the task increases, the rewards become more and more sparse, making learning more complicated. By reproducing Plappert et al. [39] results (Section 4.5), we show that adding a HER helps to solve part of the exploration problem (Section 6.2). Finally, we evaluate more specifically the grasping task (Section 6.3) which is the central task of our subject. We show that after a reasonable learning time, the resulting model successfully completes the pick and place task in almost all the time.

The results presented in this paper are only the beginning of the ongoing experiments. Continuing on this iterative approach of increasing complexity, the next experiments will evaluate the impact of using soft objects, and extending observation space by including data from cameras and tactile sensors. Details of the upcoming experiments are presented in Section 7.

The results of the experiments are represented as learning curves. The experiments were repeated several times. The line represents the median success rate at a given learning epoch and the shaded area represents the interquartile range.

6.1 DDPG for reaching: the criticality of the reward function

We first evaluate the performance of DDPG for the reaching task. The objective for the robot is to move its gripper to a target position. Several reward functions are possible. The first one is to give a $+1$ reward if the gripper is at most a given distance from the target position, 0 otherwise. This is called a *sparse* reward function. A second reward function, called *dense*, is to give a reward between 0 and $+1$, the higher the reward the closer the gripper is to the target position (this function is therefore continuous). We compare the learning curves for these two reward functions (Figure 6).

All hyperparameters are shown in Table 9. The hyperparameters are the default hyperparameters of the stable-baseline implementation. No fine tuning has been done.

The training was done on a classic laptop computer (described in Section 5), and lasted 20 minutes. It was repeated 14 times with a different seed.

One can notice that the task is learned very quickly when the rewards are dense. However, when the rewards are sparse, the algorithm does not learn at all. It is likely that the state space is so large that it is very unlikely that the

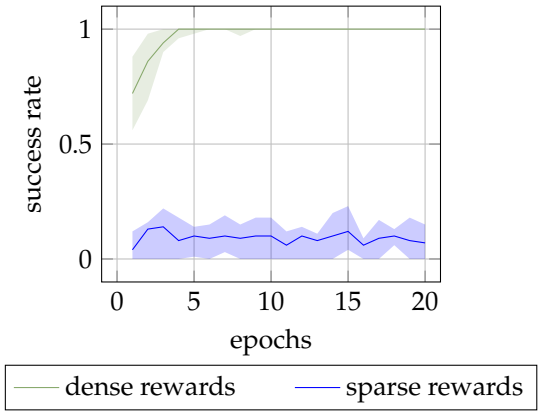


Figure 6: Comparing learning curves of DDPG for dense and sparse rewards. The model was trained on the OpenAI/gym `FetchReach-v1` environment. The training process lasts approximately one hour and was repeated 14 times. Results tends to show that dense rewards are required to learn in this configuration.

expected position will be reached by chance. The rewards would therefore be too infrequent to allow effective learning.

For this simple task, it is easy to imagine a dense function. It can easily be established that the closer the gripper is to the target, the higher the reward should be. Thus, we guide the model's exploration towards areas of high reward. However, for more complex tasks, it is much harder to define this dense reward function. Let's take the example of the task of sliding a cube to a target position. If the same reward function is chosen (the closer the cube is to the target, the higher the reward), this function would be constant on states where the gripper has not touched the cube. Therefore, it would be impossible to guide the model to the high reward areas until the gripper has touched the cube. This example shows that for slightly more complex tasks, the engineering of the reward function is much more complex, and becomes a critical component of learning.

Environment	
Type	`FetchReach-v1`
DDPG	
Number of layers	2
Hidden size	16
Actor and critic learning rate	0.001
Buffer size	10^6
Training	
Total epochs	20
Number of cycles per epoch	100
Number of batches	50
Batch size	256
Number of test rollout	10
Scale of action noise	0.2

Table 9: Hyperparameters used to compare DDPG learning curves for dense and sparse rewards

6.2 Hindsight Experience Replay

In the previsous section, we showed that DDPG manage to solve the reaching tasks as long as the reparation of the rewards is dense. For this simple task it is easy to densify the reward function. However, for more complex tasks such as grasping objects, densifying the reward function is very complex. This requirement is therefore very strong.

As described in Section 4.5, HER presents a clever solution to this problem. In this section, we attempt to reproduce the results presented by Plappert et al. [39] on a rigid cube for the four basic tasks that are : reach, slide, push and pick and place. The experiments should allow us to answer the following questions: (1) What performances can we obtain using a classical learning algorithm? (2) What is the gain of adding HER in the same algorithm? (3) What kind of reward distribution best fits HER?

6.2.1 DDPG and HER

The first tries were done using stable-baselines [41].

DDPG	
Number of layers	2
Hidden size	16
Actor and critic learning rate	0.001
Buffer size	10^6
Training	
Total epochs	20
Number of cycles per epoch	100
Number of batches	50
Batch size	256
Number of test rollout	10
Scale of action noise	0.2
HER (if enabled)	
HER experience per actual experience	4
Observation clipping	$[-5, 5]$

Table 10: Hyperparameters used to evaluate the contribution of DDPG for dense and sparse rewards

First, we notice that adding HER allows the agent to learn very efficiently (from the fourth epoch) a task not learned in the previous section: reach with sparse rewards.

Nevertheless, the learning has been a failure on the other tasks. None of them were learned, and the success rate did not increase at all during training.

Results from Plappert et al. [39] are much better. It is likely that some hyperparameters play a key role in learning tasks using HER. It seems that this lack of learning may be due to the lack of sufficient parallelisation. In the original paper, 19 Message Passing Interface (MPI) workers were working in parallel. Hill et al. [41] suggest that 8 MPI workers are at least necessary to obtain good results. In the following section we will use 8 workers and we will rigorously reproduce the hyperparameters of the initial paper.

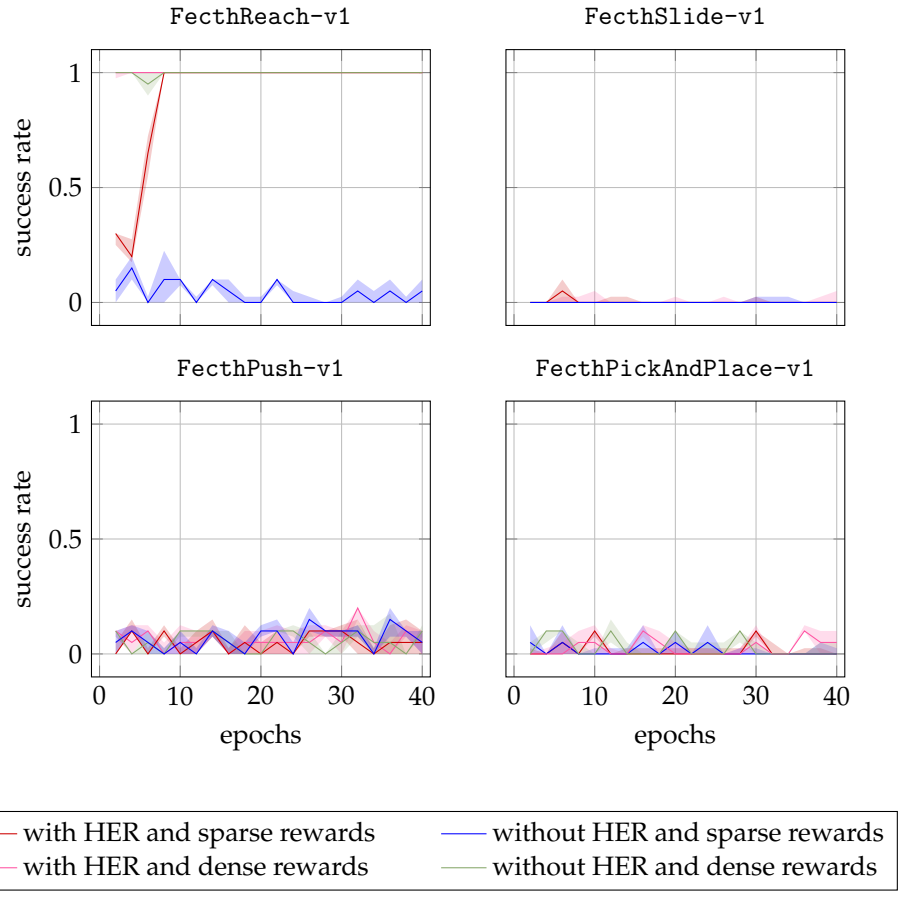


Figure 7: Comparaing learning curves of unparallelized DDPG with and without HER for dense and sparse rewards. The model was trained on the OpenAI/gym Fetch environments. The training process lasts approximately one hour and was repeated 6 times per environment. Results tends to show that this configuration is not sufficient to learn the tasks.

6.2.2 Parallelized learning for HER

To be able to reproduce the results of Plappert et al. [39], it is necessary to match the hyperparameters used as closely as possible. For this, we will use the OpenAI baselines [42] package for the following. This package notably allows to use MPI to parallelise the learning with MPI. It is advised to use at least 8 workers to get good results [41].

The model was trained with and without HER. The hyperparameters used are presented in Table 11 and the learning curve in Figure 8. An illustration of how the agent behaves after 20 training epochs is shown in Figure 9.

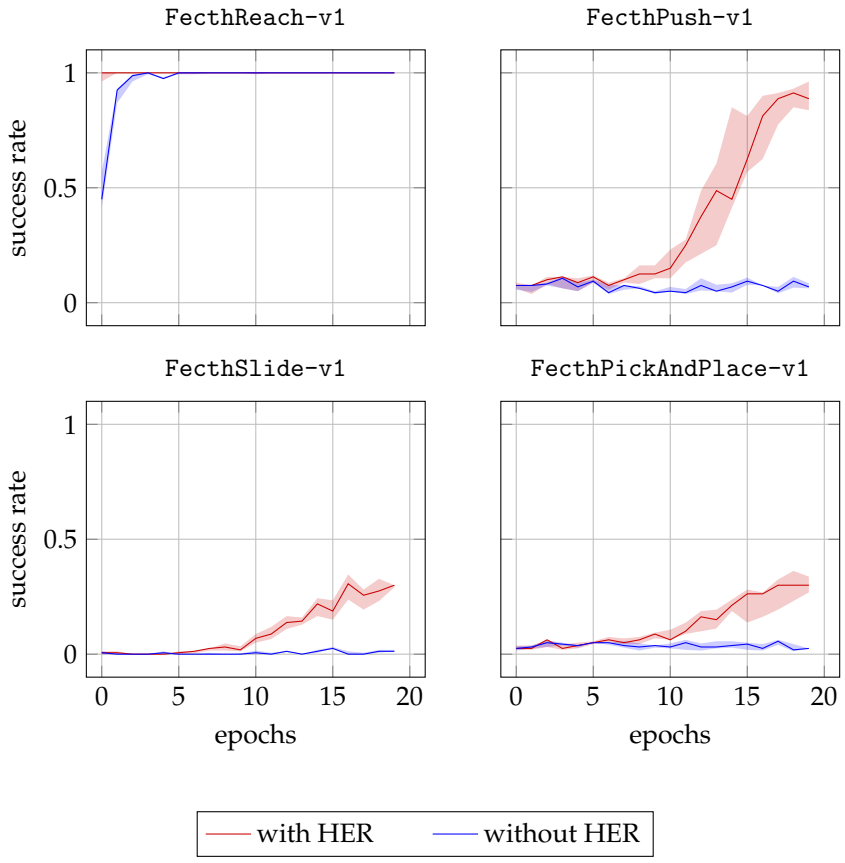


Figure 8: Comparing learning curves of parallelized DDPG with and without HER. The model was trained on the OpenAI/gym Fetch environments. The training process lasts approximately one hour and was repeated 14 times per environment. Results tends to show that this configuration is not sufficient to learn the tasks.

Visually (Figure 9), we can see that the agent has learned the reach and push task well. For example, the reach task is successful from the eighth timestep (image 3). For slide, there are still a lot of failures but the test presented visually shows that the agent is still able to correctly perform the task at this stage of training. Conversely, the pick and place task is not yet learned. We can see that

Environment	
Maximum magnitude of actions	1.0
Rewards	sparse
DDPG	
Number of workers	8
Number of layers	3
Hidden size	256
Actor and critic learning rate	0.001
Buffer size	10^6
Polyak-averaging coefficient	0.95
Quadratic penalty on actions	1.0
Observation clipping (before normalization)	$[-200, 200]$
Training	
Total epochs	20
Number of cycles per epoch	50
Rollout batch size per MPI worker	2
Number of batches	40
Batch size	256
Number of test rollout	10
Probability of taking a random action	0.3
Scale of action noise	0.2
HER (if enabled)	
HER experience per actual experience	4
Observation clipping	$[-5, 5]$

Table 11: Hyperparameters for training on the four Fetch environments with OpenAI/baselines

the agent is laboriously moving the cube to the position but has not yet fully assimilated the action of using his gripper to move the cube. In Section 6.3, we show nevertheless that with a little more training, this task will be perfectly assimilated by the agent.

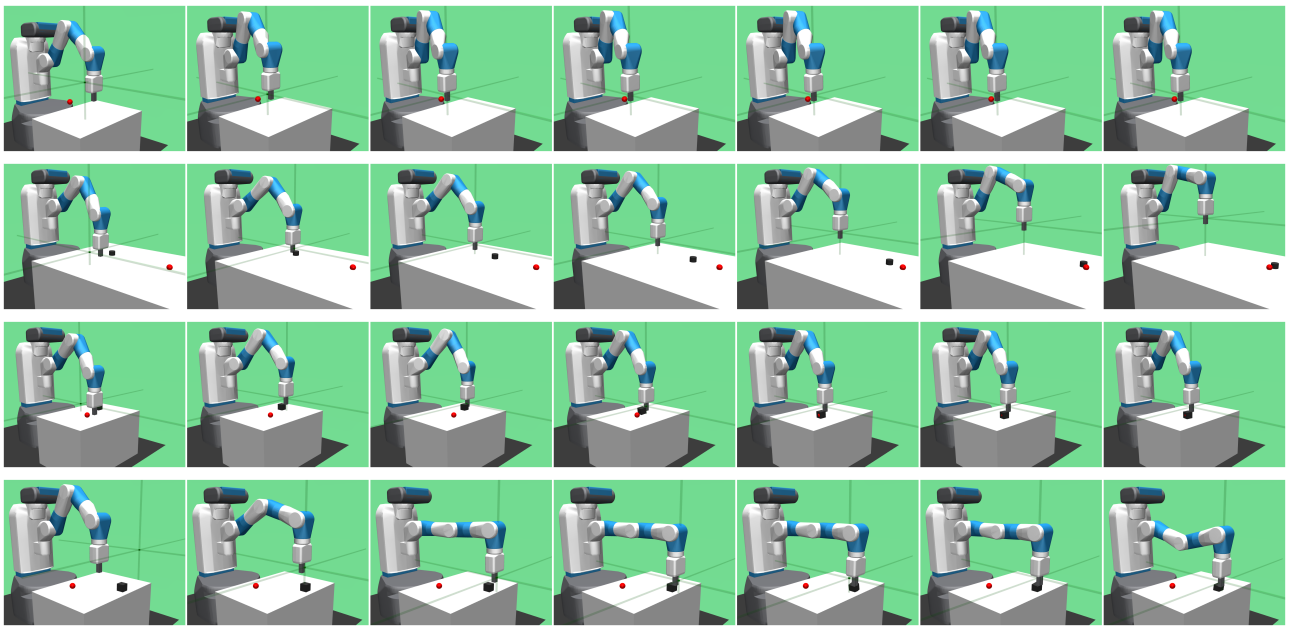


Figure 9: Illustration of agent behavior of Fetch environments after 20 epochs of training. The lines correspond in order to the `FetchReach-v1`, `FetchSlide-v1`, `FetchPush-v1` and `FetchPickAndPlace-v1` environments. Four timesteps separate two successives images.

6.3 Hindsight Experience Replay for the grasping task

In the previous section, we showed that HER is a very effective tool to solve the exploration issue in a multi-goal environment such as robotics. The task we are particularly interested in is the grasping of soft objects. Therefore, in this section we study the learning performance of HER for the grasping task. We also evaluate what success rate is achievable with the DDPG+HER algorithm for the grasping task. With this reference, we will be able to compare the learning results when the cube becomes a soft object.

The hyperparameters used are the same as those in Table 11, except that the training last longer (80 epochs instead of 20). The training lasted approximately three hours and were done on a bassic laptop (see Section 5). It was repeated 15 times. Learning curve is shown Figure 10. Some demonstrations of the obtained results are presented on Figures 11 12 and 13.

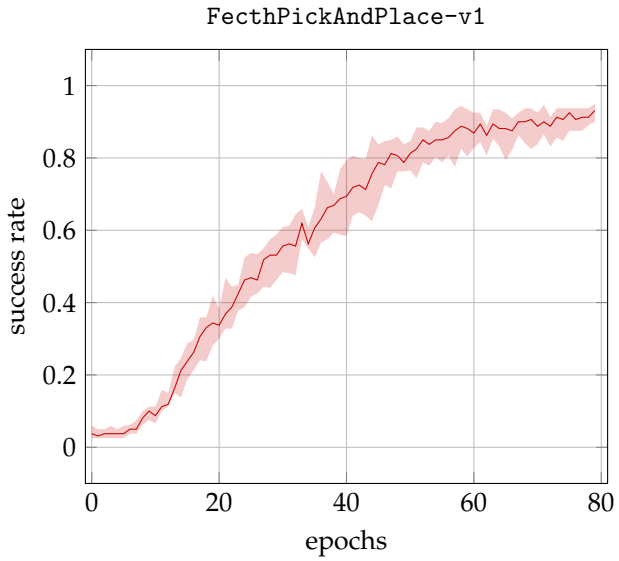


Figure 10: Learning curves of HER in `FetchPickAndPlace-v1`. The model achevies 93.1% success rate. The training lasts approximately 3 hours and was repeated 25 times.

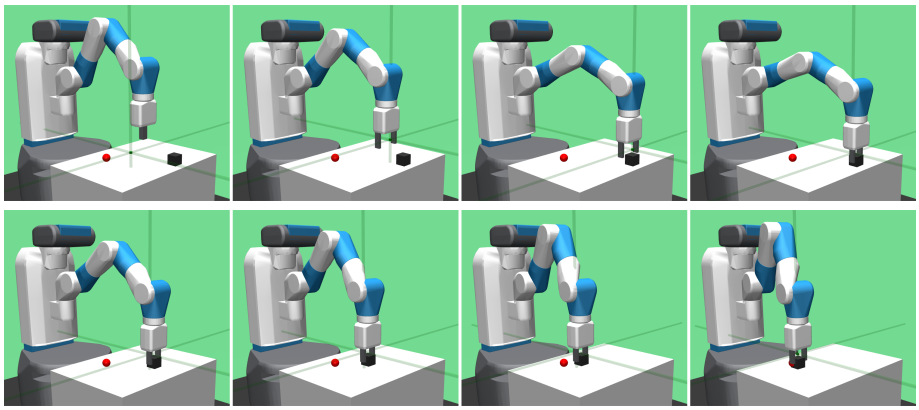


Figure 11: Demonstration of the pick and place policy learned after 80 epochs of training. Two timesteps separate two successive images. The success of the task is achieved from the 15th timestep.

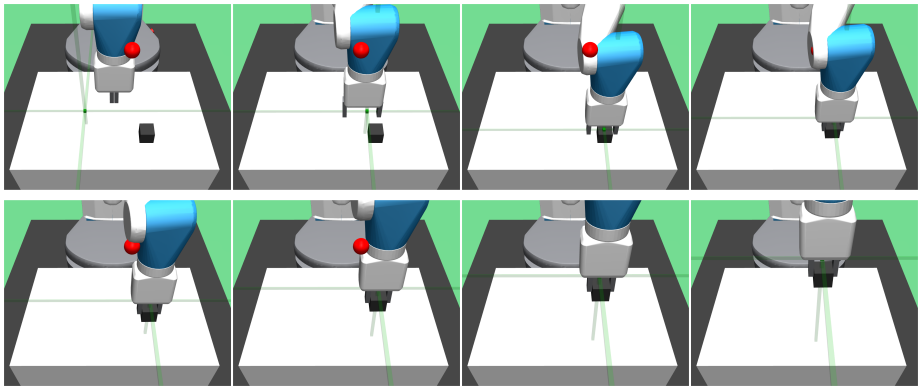


Figure 12: Demonstration of the pick and place policy learned after 80 epochs of training. Two timesteps separate two successive images. The success of the task is achieved from the 15th timestep.

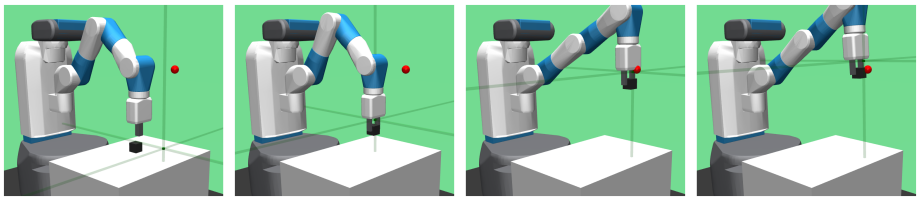


Figure 13: Pick and place failure. The policy was learned during 80 epochs. Eight timesteps separate two successive images.

7 Future work

Initial results are encouraging. They allow us to confirm that the HER method could be applied to learn a grasping task in a reasonable amount of time, saving us the tedious work of creating a good reward function. Nevertheless, we can see that in order to reproduce satisfactory results, it is necessary to parallelize the learning process to feed the replay buffer sufficiently. Without this, policy does not learn when the task is somewhat more complex than the reach. The causes of this discrepancy in results are unclear and could be due to the way OpenAI/baselines aggregates interaction data.Future work will dive into the code to determine the true reason.

Many questions need to be answered now. The real robot is based on data acquisition by cameras, which greatly increases the dimensionality of the observation space. Learning directly from pixels is often not efficient. This is why processing algorithms (such as semantic segmentation, or pose recognition) process the pixel data upstream. RL algorithms based on this digested data have a much better chance of learning correctly. Future work will attempt to determine how learning algorithms respond to this change in observation space.

Next, the objects captured so far are hard and cubic objects. Grasping soft objects is much more challenging since the force with which the object is grasped has to be adapted. This force must be just enough to prevent the object from slipping, but must not be excessive, as this could damage the soft object. A lot of work has been done on the design of the gripper to make it suitable for gripping soft objects. It should be noted that the wider the surface in contact with the soft object, the less this compromise on gripping force is tightened: it is much easier to catch an orange with the whole hand than with just two fingers.

Work in progress aims at making the object to be simulated soft based on simple deformation models. The goal is then to adapt the generation of hindsight experiences so that it can work with soft objects. We will then be able to compare the results obtained with a rigid object and with a soft object.

Then, the goal will be to extend the observation space with simulated cameras and tactile sensors. Using instance segmentation and pose detection algorithms, we should be able to identify the configuration that allows the best learning capability.

Finally, the results obtained would be transferred to a real robot. As we have seen in part 4.4, several methods can be used to overcome the gap between simulation and reality. Initially, we have to find the same results with an environment that simulates our PANDA robot. A simulator based on ROS and Gazebo is under development and should be available by the end of 2020. Then, the planned work aims at comparing methods such as the one based on domain randomization, or self-supervised learning.

8 Conclusion

We started by studying the mathematical foundations and the essential notions of reinforcement learning. Using these concepts, we were able to establish a state of the art in reinforcement learning for soft object manipulation. The skills required are numerous. Robotic manipulation questions the biggest issue in reinforcement learning: how to find a good policy when the dimensionality of the observation space is so important? Many tracks are relevant, and they must be combined. By basing our approach both on the perception of the environment with tactile sensors, the processing of camera data by vision algorithms, the observation space reduces its dimensionality while keeping its richness. Robot simulation learning can use methods emerging from the literature, such as HER. We show in this work that for a complex task, a few hours of training are enough to acquire the competence with a high reliability. Nevertheless, this method still requires to imagine an algorithm that allows to generate retrospective experiences. A conceivable extension of HER could be to integrate the hierarchical version of the goals, replacing the objectives also at higher hierarchical levels.

Eventually, this learning will be based on realistic simulations of the environment and more particularly of its variations. This will allow the gap between simulation and reality to be reduced as much as possible. By combining this with effective methods to overcome this gap, such as domain randomization, the trained agent could be deployed very easily in the real world.

The experiments done in this document only concern rigid objects for the moment. The good results obtained make us confident to extend them to soft objects. The work in progress aims at confirming this intuition.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. URL `http://incompleteideas.net/book/RLbook2020.pdf`.

[2] Wilder Penfield and Edwin Boldrey. Somatic motor and sensory representation in the cerebral cortex of man as studied by electrical stimulation. *Brain*, 60(4):389–443, 1937.

[3] Jean-Baptiste Durand, Koen Nelissen, Olivier Joly, Claire Wardak, James T Todd, J Farley Norman, Peter Janssen, Wim Vanduffel, and Guy A Orban. Anterior regions of monkey parietal cortex process visual 3D shape. *Neuron*, 55(3):493–505, 2007.

[4] Thomas Duboudin, Maxime Petit, and Liming Chen. Toward a procedural fruit tree rendering framework for image analysis. *arXiv preprint arXiv:1907.04759*, 2019.

[5] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3): 287–308, 2000.

[6] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[7] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. 9 2015. URL `https://arxiv.org/abs/1509.06461`.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and others. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[9] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[10] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[11] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. 2 2015. URL `https://arxiv.org/abs/1502.05477`.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. 7 2017. URL `http://arxiv.org/abs/1707.06347`.

[13] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245, 2018.

[14] Michelle A Lee, Carlos Florensa, Jonathan Tremblay, Nathan Ratliff, Animesh Garg, Fabio Ramos, and Dieter Fox. Guided Uncertainty-Aware Policy Optimization: Combining Learning and Model-Based Strategies for Sample-Efficient Policy Learning. *arXiv preprint arXiv:2005.10872*, 2020. URL `https://sites.google.com/view/guapo-rl`.

[15] Andrew G Barto and Sridhar Mahadevan. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13(1):41–77, 2003. ISSN 1573-7594. doi: 10.1023/A:1022140919877. URL `https://doi.org/10.1023/A:1022140919877`.

[16] AJ Piergiovanni, Alan Wu, and Michael S. Ryoo. Learning Real-World Robot Policies by Dreaming. *IEEE International Conference on Intelligent Robots and Systems*, pages 7680–7687, 5 2019. ISSN 21530866. doi: 10.1109/IROS40897.2019.8967559. URL `http://arxiv.org/abs/1805.07813`.

[17] Lin Yen-Chen, Andy Zeng, Shuran Song, Phillip Isola, and Tsung-Yi Lin. Learning to See before Learning to Act: Visual Pre-training for Manipulation. *Icra*, pages 7286–7293, 2020. URL `https://drive.google.com/file/d/1D0d2plVlvdk0ltGSTK7940TCs3CGTgKy/view`.

[18] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. 9 2018. doi: 10.15607/rss.2018.xiv.049. URL http://arxiv.org/abs/1709.10087.

[19] Tong Cui, Jing Xiao, and Aiguo Song. Simulation of grasping deformable objects with a virtual human hand. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 3965–3970, 2008. doi: 10.1109/IROS.2008.4651080.

[20] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. 6 2016. URL https://arxiv.org/abs/1606.01540.

[21] Lillian Chin, Felipe Barscevicius, Jeffrey Lipton, and Daniela Rus. *Multiplexed Manipulation: Versatile Multimodal Grasping via a Hybrid Soft Gripper*. ICRA2020, 2020. ISBN 9781728173955.

[22] Joseph M. Romano, Kaijen Hsiao, Günter Niemeyer, Sachin Chitta, and Katherine J. Kuchenbecker. Human-inspired robotic grasp control with tactile sensing. *IEEE Transactions on Robotics*, 27(6):1067–1079, 2011. ISSN 15523098. doi: 10.1109/TRO.2011.2162271.

[23] G. De Maria, C. Natale, and S. Pirozzi. Force/tactile sensor for robotic applications. *Sensors and Actuators, A: Physical*, 175:60–72, 2012. ISSN 09244247. doi: 10.1016/j.sna.2011.12.042. URL http://dx.doi.org/10.1016/j.sna.2011.12.042.

[24] Heba Khamis, Raquel Izquierdo Albero, Matteo Salerno, Ahmad Shah Idil, Andrew Loizou, and Stephen J. Redmond. PapillArray: An incipient slip sensor for dexterous robotic or prosthetic manipulation – design and prototype validation. *Sensors and Actuators, A: Physical*, 270: 195–204, 2018. ISSN 09244247. doi: 10.1016/j.sna.2017.12.058. URL http://dx.doi.org/10.1016/j.sna.2017.12.058.

[25] Sascha Fleer, Alexandra Moringen, Roberta L. Klatzky, and Helge Ritter. Learning efficient haptic shape exploration with a rigid tactile sensor array. *PLoS ONE*, 15(1):1–22, 2020. ISSN 19326203. doi: 10.1371/journal.pone.0226880. URL http://dx.doi.org/10.1371/journal.pone.0226880.

[26] Carmelo Sferrazza, Thomas Bi, and Raffaello D'Andrea. Learning the sense of touch in simulation: a sim-to-real strategy for vision-based tactile sensing. 3 2020. URL http://arxiv.org/abs/2003.02640.

[27] Yevgen Chebotar, Karol Hausman, Zhe Su, Gaurav S. Sukhatme, and Stefan Schaal. Self-supervised regrasping using spatio-temporal tactile features and reinforcement learning. *IEEE International Conference on Intelligent Robots and Systems*, 2016-Novem:1960–1966, 11 2016. ISSN 21530866. doi: 10.1109/IROS.2016.7759309.

[28] Jose Sanchez, Carlos M. Mateo, Juan Antonio Corrales, Belhassen Chedli Bouzgarrou, and Youcef Mezouar. Online Shape Estimation based on Tactile Sensing and Deformation Modeling for Robot Manipulation. *IEEE International Conference on Intelligent Robots and Systems*, pages 504–511, 2018. ISSN 21530866. doi: 10.1109/IROS.2018.8594314.

[29] Qi Luo and Jing Xiao. Contact and deformation modeling for interactive environments. *IEEE Transactions on Robotics*, 23(3):416–430, 6 2007. ISSN 15523098. doi: 10.1109/TRO.2007.895058.

[30] Jan Matas, Stephen James, and Andrew J. Davison. Sim-to-Real Reinforcement Learning for Deformable Object Manipulation. (CoRL), 2018. URL http://arxiv.org/abs/1806.07851.

[31] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[32] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. 6 2018. URL http://arxiv.org/abs/1806.10293.

[33] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[34] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation. 9 2017. URL http://arxiv.org/abs/1709.10489.

[35] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. Technical report, 2019. URL http://robotics.sciencemag.org/.

[36] Heejin Jeong, Brent Schlotfeldt, Hamed Hassani, Manfred Morari, Daniel D. Lee, and George J. Pappas. Learning Q-network for Active Information Acquisition. *IEEE International Conference on Intelligent Robots and Systems*, pages 6822–6827, 2019. ISSN 21530866. doi: 10.1109/IROS40897.2019.8968173.

[37] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[38] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):5049–5059, 2017. ISSN 10495258.

[39] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. 2 2018. URL http://arxiv.org/abs/1802.09464.

[40] Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning, 2019. URL http://pybullet.org.

[41] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable Baselines. https://github.com/hill-a/stable-baselines, 2018.

[42] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. OpenAI Baselines. https://github.com/openai/baselines, 2017.

Acronyms

ANN Artificial Neural Network. 24, 25

DDPG Deep Deterministic Policy Gradient. 6, 7, 29, 30, 39–45

DP Dynamic Programming. 6, 7, 15, 17, 18

DRL Deep Reinforcement Learning. 1, 2, 6, 24, 25, 31–33, 39

GLIE Greedy in the Limit with Infinite Exploration. 7, 18, 19, 24

HER Hindsight Experience Replay. 1, 6–9, 34, 39, 42–45, 47, 49

IS Importance Sampling. 7, 23

KL Kulleman-Leibler. 30

LEARN-REAL *Improving reproducibility in LEARNing physical manipulation skills with simulators using REAListic variations*. 6, 8, 10

MC Monte Carlo. 6, 7, 18–20, 23–25, 27–29

MDP Markov Decision Process. 12–14

MPI Message Passing Interface. 42, 44, 45

PPO Proximal Policy Optimization. 30

RL Reinforcement Learning. 1, 5–8, 12–14, 18, 22, 24, 25, 49

SARSA State-Action-Reward-State-Action. 7, 21, 22, 24–26

TD Temporal-Difference. 6, 7, 19–21, 23–25, 27, 29

TRPO Trust Region Policy Optimization. 30

ÉCOLE CENTRALE LYON

membre de : UNIVERSITÉ DE LYON

FINAL YEAR INTERNSHIP

Academic year 2019-2020

This form should be inserted after the cover page of the FYI Report. It certifies that the Report has been validated by the firm and can be submitted as it stands to the Registrar's Office of Ecole Centrale de Lyon.

COMPANY VALIDATION OF FYI REPORT

Final Year Internship references

Student's name: Quentin GALLOUÉDEC

Report title: Deep reinforcement learning for soft object grasping

Firm: LIRIS

Name of Company Tutor: Nicolas CAZIN

Name of School Tutor: Emmanuel DELLANDRÉA

The firm acknowledges awareness of the above Report and authorizes communication to Ecole Centrale de Lyon (ECL).

- The company authorizes the release of the report on the internet
- ~~The company authorizes the release of the report exclusively on the intranet of ECL~~

(strike out inapplicable part)

Firm's representative

Name: Emmanuel Dellandréa

Position:

Responsible for LIRIS at ECL

Date: 26/08/2020

Signature and stamp:

Laboratoire LIRIS CNRS UMR 5205
ECOLE CENTRALE DE LYON
Département Mathématiques - Informatique
36, avenue Guy de Collongue
69134 ECULLY CEDEX