Autonomous Learning in Robotics

Charles Wilmot

Supervisor: Prof. Jochen Triesch

Department of Computer Sciences
Goethe Universität Frankfurt

This dissertation is submitted for the degree of
Doctor of Philosophy

FIAS February 2022

I would like to dedicate this thesis to my girlfriend Trupti, to my parents Philippe and Isabelle, and to my friends Alexandre, Antoine, Felix, Guillaume, Killian, Paul B, Paul R, Pierre and Quentin, whom I could always count on.

> So great is our innate love of learning and of knowledge that no one can doubt that man's nature is strongly attracted to these things even without the lure of any profit. - Cicero

> The actual science of logic is conversant at present only with things either certain, impossible, or entirely doubtful, none of which (fortunately) we have to reason on. Therefore the true logic for this world is the calculus of probabilities, which takes account of the magnitude of the probability which is, or ought to be, in a reasonable man's mind. - James Clerk Maxwell

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Charles Wilmot
February 2022

Acknowledgements

I would like to acknowledge my supervisor Jochen Triesch, for his trust and support, and my friends and colleagues, Max Murakami, Lukas Klimasch, Alexander Lelais, Bruno Del Papa, Diyuan Lu, Julias Taylor, Markus Ernst and Jonas Mahn for making me feel welcomed in Frankfurt.

Abstract

Recent advances in artificial neural networks enabled the quick development of new learning algorithms, which, among other things, pave the way to novel robotic applications.

Traditionally, robots are programmed by human experts so as to accomplish pre-defined tasks. Such robots must operate in a controlled environment to guarantee repeatability, are designed to solve one unique task and require costly hours of development.

In developmental robotics, researchers try to artificially imitate the way living beings acquire their behavior by learning. Learning algorithms are key to conceive versatile and robust robots that can adapt to their environment and solve multiple tasks efficiently.

In particular, Reinforcement Learning (RL) studies the acquisition of skills through teaching via rewards. In this thesis, we will introduce RL and present recent advances in RL applied to robotics. We will review Intrinsically Motivated (IM) learning, a special form of RL, and we will apply in particular the Active Efficient Coding (AEC) principle to the learning of active vision. We also propose an overview of Hierarchical Reinforcement Learning (HRL), an other special form of RL, and apply its principle to a robotic manipulation task.

Abstrakt

Jüngste Fortschritte auf dem Gebiet der Künstlichen Neuronalen Netze ermöglichten die rasche Entwicklung neuartiger Lernalgorithmen, die unter anderem den Weg zu neuartigen Anwendungen von Robotern ebnen.

Üblicherweise werden Roboter von menschlichen Experten programmiert, um vordefinierte Aufgaben zu erfüllen. Dabei müssen die Roboter in einer kontrollierten Umgebung arbeiten, um Wiederholbarkeit zu gewährleisten. Darüber hinaus, sind sie meist lediglich für eine einzige Aufgabe konzipiert und müssen kostspielig und zeitaufwändig konfiguriert werden.

Im Bereich "Developmental Robotics" versuchen Forschende das Lernverhalten von Lebewesen künstlich zu imitieren. Die daraus resultierenden Lernalgorithmen sind der Schlüssel zur Entwicklung vielseitiger und robuster Roboter, die sich an ihre Umgebung anpassen und mehrere Aufgaben effizient lösen können.

Insbesondere das Reinforcement Learning (RL) untersucht den Erwerb von Fähigkeiten, die allein durch Belohnungs- und Bestrafungssignale vermittelt werden. In dieser Arbeit werden wir RL eingehend behandeln und aktuelle Fortschritte in der Anwendung von RL in der Robotik präsentieren. Wir geben einen Überblick über das intrinsisch motivierte Lernen (IM), eine spezielle Form des RL, und wenden speziell das Prinzip des Active Efficient Coding (AEC) auf das Lernen des aktiven Sehens an. Wir geben auch einen Überblick über das hierarchische Verstärkungslernen (HRL), eine andere spezielle Form des RL, und wenden dessen Prinzip auf eine Robotermanipulationsaufgabe an.

Kurzfassung

In den frühen 70er Jahren kombinierten Seymour Papert und Marvin Minsky einen Computer, eine Kamera und einen Roboterarm und versuchten, einen allgemeinen Algorithmus zu konzipieren, der Türme aus Würfeln bauen könnte. Zuerst versuchten sie, Routinen und von diesen ausgeführte Unterroutinen manuell zu programmieren.

Schnell erkannten sie, dass der Ansatz unpraktisch ist und dass ein cleverer Algorithmus Fähigkeiten von selbst lernen sollte. Dieses Experiment inspirierte sie zu ihrer berühmtesten Theorie, der 1988 veröffentlichten "Society of Mind". Sie beschreibt, wie Intelligenz aus nicht intelligenten, einfacheren Komponenten entstehen kann. Insbesondere erklärt er, dass Fähigkeiten wiederverwendbar und in einer "Heterarchie" (im Gegensatz zu einer Hierarchie) organisiert sein müssen. Zum Beispiel setzt sich die Fertigkeit "einen Würfel auf dem Turm stapeln" aus einfacheren Fertigkeiten zusammen wie "einen freien Würfel zum Stapeln anschauen", "nach dem Würfel greifen", "greifen", "heben", "den Arm bewegen" , "fallen lassen" usw. Jeder *Skill* wiederum kann andere *Skills* aufrufen, höher oder niedriger in der Hierarchie.

Obwohl es logisch erscheint, das Problem der Robotermanipulation in einfachere, elementare Aufgaben aufzuteilen, bleibt das Problem, die einfacheren Aufgaben auf eine Weise zu lösen, die sich gut auf neue, unsichtbare Versuchsanordnungen übertragen lässt. Während sich die heutigen Computer hervorragend zum Lösen fortgeschrittener Berechnungen eignen, sind ihre Fähigkeiten sehr begrenzt, wenn es darum geht, selbst einfache Manipulationsaufgaben auszuführen. Dies hat der Robotiker Hans Moravec wie folgt formuliert:

> Es ist vergleichsweise einfach, Computer dazu zu bringen, bei Intelligenztests oder Damespielen Leistungen auf Erwachsenenniveau zu erbringen. Es ist jedoch schwierig oder unmöglich, ihnen die Fähigkeiten eines Einjährigen in Bezug auf Wahrnehmung und Mobilität zu vermitteln.

Warum sind einfache Dinge für Computer schwerer?
Marvin Minsky bemerkte:

> An jedem Punkt in dieser Welt der Blöcke, wo wir gezwungen waren, genauer als gewöhnlich hinzuschauen, fanden wir ein unerwartetes Universum der Komplika-

> tionen. Betrachten Sie nur das scheinbar einfache Problem, bereits eingebaute Blöcke im Turm nicht wiederzuverwenden. Für eine Person scheint dies einfach gesunder Menschenverstand zu sein: Verwenden Sie kein Objekt, um ein neues Ziel zu erreichen, wenn dieses Objekt bereits an der Erreichung eines früheren Ziels beteiligt ist. Niemand weiß genau, wie der menschliche Verstand das macht.

Und er kommt zu dem Schluss,

> Im Allgemeinen wissen wir am wenigsten, was unser Verstand am besten kann.

Diese einfachen Aufgaben, die wir Menschen unbewusst erledigen, programmgesteuert zu lösen, ist auch heute noch eine große Herausforderung im *Machine Learning* und in der künstlichen Intelligenz.

Heutzutage werden Roboter in der Industrie typischerweise gesteuert, indem die Gelenkwinkel numerisch berechnet werden, die eine gewünschte Endeffektorposition erzeugen würden, und dann jeder Motor betätigt wird, um den Zielgelenkwinkel gemäß der Dynamik eines PID-Reglers zu erreichen. Der Weg, den der Endeffektor nimmt, wird dann so beschrieben, dass er eine Liste von Zwischenpunkten durchläuft und optional Aktionen auf dem Weg auslöst (wie Schweißen, Greifer schließen, öffnen usw.). Dies impliziert, dass ein menschlicher Experte zuerst die PID-Regler kalibrieren und dann das Skript schreiben muss, dem der Roboter folgt. Die Umgebung, in der sich der Roboter entwickelt, muss stark kontrolliert werden, um die Wiederholbarkeit und Sicherheit der Ausführung zu gewährleisten.

Die Herausforderung der nächsten Jahre in der Robotik wird darin bestehen, Steueralgorithmen zu entwerfen, die sich an ihre Umgebung anpassen können und die ein Minimum an menschlichem Eingreifen erfordern, um sich selbst auf die Aufgaben zu kalibrieren. Die Frage ist also, wie können wir Roboter entwerfen, die sich an ihre Umgebung anpassen können, um ein breiteres Spektrum einfacher Aufgaben zu erfüllen?

Die Antwort von Robotikern ist, sich von der Art und Weise inspirieren zu lassen, wie Lebewesen ihr Verhalten erwerben: durch Lernen.

In Abschnitt 2.2.1 der Dissertation stellen wir den Formalismus der “Markow-Entscheidungsprobleme” vor, der die Interaktion zwischen einem Agenten und seiner Umgebung und die Qualität seines Verhaltens darin durch die Definition einer Quantität namens Belohnung beschreibt. Das Ziel für den Agenten ist es, seine Interaktionen mit der Umgebung auszunutzen, um eine Funktion π zu finden, die seine Beobachtungen auf motorische Befehle abbildet, um die Gesamtmenge an Belohnung, die er erfährt, zu maximieren.

Reinforcement Learning besteht darin, die optimale Funktion π zu finden.

Die Anwendung von Reinforcement Learning auf Robotersteuerungsaufgaben ist eine Herausforderung und wirft einige Probleme auf. Das erste Hindernis liegt in der kontinuierlichen Natur der Zustände und Aktionen von Robotern, was es erforderlich macht, sorgfältig zu entscheiden, wie genau die Kontrolle ist, die wir über den Roboter benötigen, ob wir Diskretisierung oder Annäherungen an kontinuierliche Funktionen verwenden und mit welcher Häufigkeit Aktionen durch den Roboter generiert und angewendet werden sollen. Die zweite Schwierigkeit ist der "Fluch der Dimensionalität" *(curse of dimensionality)*. Er beschreibt, wie mit zunehmender Dimensionalität des Zustandsraums oder des Aktionsraums die Komplexität des Problems exponentiell zunimmt. Wenn wir davon ausgehen, dass ein Agent n kontinuierliche Aktionen zur Verfügung hat, und wenn wir uns entscheiden, diese Aktionen in 10-Bits zu diskretisieren, beträgt die Anzahl der verschiedenen Kombinationen von Aktionen 10^n. Schließlich ergibt sich Komplexität aus der inhärenten Physik der realen Welt und des Roboters. Roboter sind oft spröde und können daher ihre Umgebung nicht energetisch erkunden, Reparaturen sind teuer und bedeuten oft lange Wartezeiten. Die Versuchsbedingungen können auch während der Trainingszeit variieren. Einige Motoren reagieren empfindlich auf Temperaturänderungen, Lichtverhältnisse können variieren und die vom Roboter empfangenen visuellen Informationen beeinflussen, der Roboter kann sich im Laufe des Trainings abnutzen usw. Eine bemerkenswerte Schwierigkeit, die durch physische Einschränkungen verursacht wird, ist die Verzögerung zwischen der Erzeugung vom Befehl, seine Ausführung und das sensorische Feedback, das der Agent erhält. Bei der Anwendung auf Robotikaufgaben müssen Reinforcement-Learning-Algorithmen all diese Hindernisse berücksichtigen.

In dieser Arbeit haben wir versucht, die Erfolge von Reinforcement Learning Algorithmen zu überprüfen, die auf Robotersteuerungsaufgaben angewendet werden.

Das Problem, Entscheidungen in einer Umgebung zu treffen, bringt natürlich die Frage nach der Repräsentation des Wissens des Agenten über die Welt mit sich, die Gegenstand des Kapitels 3 ist.

Wenn um uns herum ein Ereignis eintritt, erfassen wir nicht alle durch dieses Ereignis generierten Informationen. Unser Gehirn nimmt davon nur einen Bruchteil auf, und zwar gleichzeitig über verschiedene Sinneskanäle: Sehen, Hören, Riechen, Schmecken, Tasten, aber auch Propriozeption, Nozizeption, Lokalisation im Raum usw. Nehmen wir an, dass ein Ereignis in der Welt gesehen und gehört wurde, wie zum Beispiel ein Teller, der auf dem Boden zersplittert. Um ein scharfes mentales Bild des Ereignisses zu konstruieren, muss das Gehirn die Informationen, die gesehen wurden, und die Informationen, die gehört wurden, miteinander verschmelzen. Diese Verschmelzung wird dadurch ermöglicht, dass einige Informationen über das Ereignis gleichzeitig durch beide Sinne gehen. In der Infor-

mationstheorie wird dies als gegenseitige Information oder Transinformation bezeichnet. Dabei ist zu beachten, dass letzteres statistisch definiert ist und nicht existiert, wenn ein einzelnes Ereignis oder eine einzelne Wiederholung betrachtet wird. Indem es lernt, diese gegenseitigen Informationen zu erkennen, kann das Gehirn die Informationen, die von den beiden sensorischen Modalitäten kommen, zusammenführen. Beim Brechen des Tellers hält das Sehen zusammen mit der Propriozeption die Information über die Lokalisierung des Ereignisses im Bezugssystem des Körpers. In ähnlicher Weise leitet unser Gehirn durch spektrale Transformationen an eingehenden Geräuschen die horizontale und vertikale Lokalisierung des Ereignisses ab. In der wechselseitigen Information, die visuelle und akustische Modalitäten teilen, ist also das höchst relevante Wissen enthalten, wo der Teller im Bezugssystem des Körpers zerschmettert wird. In dieser Arbeit versuchen wir, die Fragen zu beantworten, ob die gegenseitigen Informationen algorithmisch isoliert werden können und welche Art von Wissen wir über die Welt darin erwarten können.

In Kapitel 4 führen wir in das Thema der "intrinsischen Motivationen" (IM) ein. Intrinsische Motivationen existieren innerhalb des Individuums und werden durch die Befriedigung interner Belohnungen angetrieben, anstatt sich auf äußeren Druck oder extrinsische Belohnungen zu verlassen. In Anlehnung an die von Gianlucca Baldassare eingeführte Taxonomie präsentieren wir 3 Kategorien intrinsischer Motivationen: Vorhersagebasiert, Neuheitsbasiert und Kompetenzbasiert.

In Kapitel 5 untersuchen wir insbesondere *Active Efficient Coding*, eine IM, die, wie wir zeigen, den Erwerb von aktivem Sehen vorantreiben kann. Das AEC zugrunde liegende Prinzip ist eng mit dem in Kapitel 3 untersuchten informationstheoretischen Mechanismus verwandt. Unsere Implementierung des AEC-Prinzips ist durch biologische Beobachtungen motiviert, und der resultierende Agent lernt, Objekte richtig um sich herum zu fixieren, um eine Art der Fixierung durchzuführen, die als Zyklovergenz bezeichnet wird, und um Objekte reibungslos in der Zeit zu verfolgen. Kurz gesagt zeigen wir zunächst, dass das Maß des Rekonstruktionsfehlers eines Autoencoders, der binokulare Bildpaare codiert, als Maß für die Redundanz in dem Paar dienen kann. Aus dieser Beobachtung leiten wir dann eine intrinsische Belohnung ab und trainieren damit einen Agenten. Schließlich messen wir quantitativ die Leistungen des resultierenden Verhaltens.

Schließlich beschäftigen wir uns in Kapitel 6 mit dem Thema *Hierarchical Reinforcement Learning* (HRL). Nach einem kurzen Überblick über die Literatur bringen wir *Feudal RL*, *Options RL* und *Movement Primitives* (MP) unter denselben Formalismus. Wir schlagen dann einen inkrementellen Ansatz zum Aufbau eines Algorithmus vor, der *Movement Primitives* lernen kann. Wir wenden es dann auf eine Robotermanipulationsaufgabe an und vergleichen den hierarchischen Ansatz mit dem traditionellen Ansatz. Insbesondere zeigen wir, dass

das Problem des “Fluchs der Dimensionalität” eher in Bezug auf die Dimensionalität des “Explorationsraums” als auf die des Aktionsraums definiert wird. Wir zeigen, dass es möglich ist, eine komprimierte Darstellung der *Movement Primitives* zu lernen, und darüber hinaus, dass die Durchführung von Erkundungen in diesem Raum die Geschwindigkeit erhöht, mit der der Agent seine Kontrollstrategie verbessert.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 The First Thinking Machines

In 1770, the Hungarian inventor Wolfgang von Kempelen unveiled his most famous machine, *The Turk*, to the Vienna court. It is named after the appearance of the automaton which is dressed as an oriental sorcerer. It stands in front of a cabinet, on top of which lays a chess board. During the exhibition, guests were proposed to challenge the automaton at a chess game. While not being completely undefeatable, the automaton often beats his opponents within 30 minutes and his playing style was described as aggressive.

Before each exhibition, von Kempelen would open the front doors of the cabinet to show its interior to the spectators. The left part exhibited a complex machinery, while the right part revealed a red cushion and some brass structures. The design was made in such a way that an operator could hide inside and stay invisible while the presenter opened the doors, thus maintaining the illusion. While that story might seem ludicrous nowadays, this spectacle was very successful in the 18^{th} century. *The Turk* made a tour in whole Europe and even played against Benjamin Franklin in Paris when he was serving as United States ambassador. The fascination for this hoax comes from the projection of human attributes on the machine, emphasised by the presence of the animated mannequin. The first true chess playing machine came only 150 years later, in 1912. Called *El Ajedrecista*, this device could win a rook-against-king endgame. Although less spectacular than *The Turk* - it could not play full games and a human operator had to move the pieces - the device generated great enthusiasm at its debuts at the University of Paris.

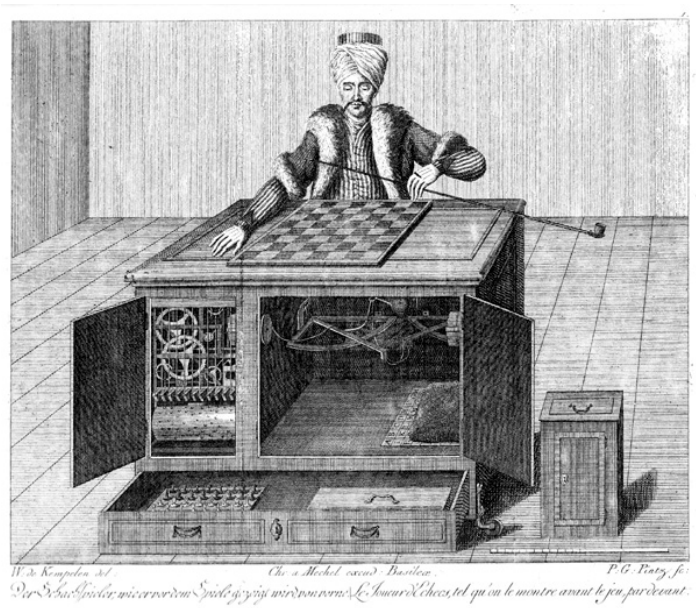


Fig. 1.1 A copper engraving of *The Turk*

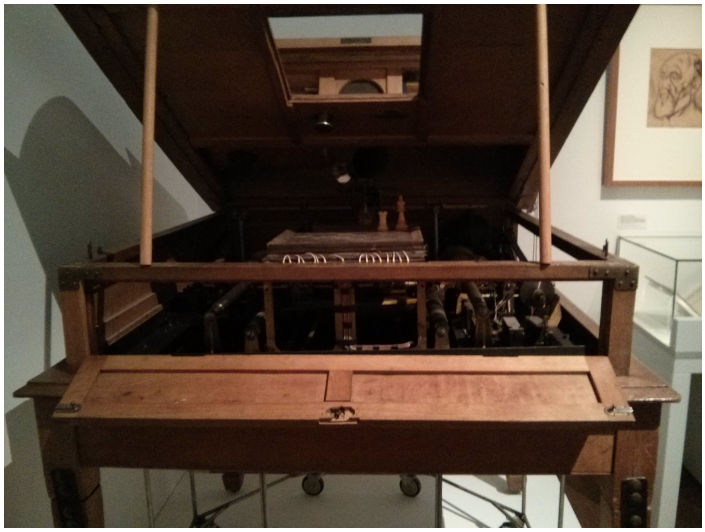


Fig. 1.2 El Ajedrecista front view on display at the *Colegio de Ingenieros de Caminos, Canales y Puertos* in Madrid.

1.2 The First Intelligent Machines

We have to wait until 1996 to see for the first time IBM's famous chess machine *Deep Blue* defeating Garry Kasparov. While this is an incredible achievement for that epoch, requiring one of the most powerful computers of that time (the 259th according to the TOP500), the machine was only following instructions given by human programmers and did not learn to play chess by itself. Further facilitating the task, the computer did not have to control a robotic arm to play. The pieces were moved by a human experimenter.

Autonomously learning algorithms are much more recent. The most famous is probably DeepMind's *Alpha Zero* [128], designed to learn by itself to evaluate the quality of a chess position and to learn which moves are interesting in a given position. The newest version of this algorithm, *Mu Zero* [123], goes a step further by also learning the rules of the game. Those learning machines are made possible by the recent development of neural networks and deep learning.

According to the Oxford dictionary, intelligence is *"the ability to learn, understand and think in a logical way about things; the ability to do this well"*. This is well reflected by the history of chess engines. *Deep Blue* was programmed to logically think and take decisions, by solving a two player game using extensive Monte-Carlo tree search. *Alpha Zero* was given the additional ability to have intuitions about the quality of a chess position and about the relevance of a move, by learning from experience. Finally, *Mu Zero* was designed to understand the rules of the game by itself, purely from interactions.

Even though playing chess might seem to be an advanced intellectual task for humans, it is nowadays a trivial task for a computer. Contrarily, the task of physically moving the chess pieces on the board is easy for even young children, while robots still don't learn it properly. This has been formulated by the roboticist Hans Moravec as follows:

> It is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility.

Why are easy things harder for computers?

Marvin Minsky, AI scientist and co-founder of the Massachusetts Institute of Technology's AI laboratory, together with his collaborator Seymour Papert, were the firsts to combine a mechanical arm, a numerical camera and a computer, in order to build towers from building cubes. They discovered the incredible complexity of everyday problems:

> At every point, in that world of blocks, when we were forced to look more carefully than usual, we found an unexpected universe of complications. Consider

> just the seemingly simple problem of not reusing blocks already built into the tower. To a person, this seems simple common sense: don't use an object to satisfy a new goal if that object is already involved in accomplishing a prior goal. No one knows exactly how human minds do this.

Minsky concludes,

> In general, we're least aware of what our minds do best.

Solving programmatically those simple tasks, which us humans achieve unconsciously, is today still a major challenge in machine learning and artificial intelligence.

1.3 Outline of the Thesis

Computers are nowadays able to solve complex problems like playing chess or go, and can furthermore learn to do so solely from interaction with the game and self-play. However, open-ended acquisition of simple motor skills like reaching, grasping or locomotion is not yet understood.

In this thesis, we will try to address the problem of artificially learning simple skills. More precisely, we will take a developmental perspective on the acquisition of skills by robots. After introducing the key concepts and reviewing existing work in Chapter 2, we will address the topic of (multimodal) representation learning in Chapter 3. Next, in Chapter 4 we will introduce Intrinsic Motivations (IM). In Chapter 5, we focus on one type of IM in particular, to derive an bio-inspired algorithm that autonomously learns to perform accurate vergence, cyclovergence and tracking movement. Finally, the last Chapter addresses the topic of robotic manipulation under the scope of Hierarchical Reinforcement Learning.

As an illustration of the interest of the scientific community for the topics addressed in this thesis, Fig. 1.3 shows the proportion of papers which title contains the terms *Intrinsic Motivation* or *Hierarchical Reinforcement Learning* within all papers of the *artificial intelligence* category on the arXiv platform containing the term *Reinforcement Learning* from 2010 to 2022.

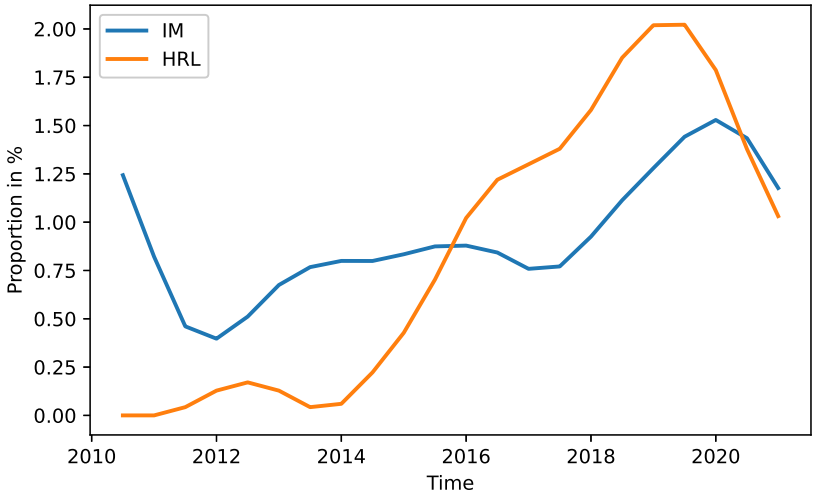


Fig. 1.3 Proportion of papers which title contains the terms *Intrinsic Motivation* or *Hierarchical Reinforcement Learning* within all papers of the *artificial intelligence* category on the arXiv platform containing the term *Reinforcement Learning* from 2010 to 2022. (Curves smoothened for improved readability).

Chapter 2

Overview of learning algorithms for robotic control

2.1 The First Learning Robotic Setups

In the early 70's, Papert and Minsky combined a computer, a camera and a robotic arm, and tried to conceive a general algorithm that could build towers from cubes. At first, they tried to manually program routines, and subroutines triggered by the firsts. Quickly they realized that the approach is impractical, and that a clever algorithm should learn skills by itself. This experiment inspired him his most famous theory, the *Society of Mind* [80] published in 1988. It describes how intelligence can arise from non intelligent, simpler components. In particular, he explains that skills must be reusable and organized in a *heterarchy* (as opposed to a hierarchy). For example, the skill of *stacking a cube on the tower* is composed of simpler skills like *looking at a free cube to stack*, *reaching for the cube*, *grasping*, *lifting*, *moving the arm*, *dropping*, etc. In turn, each skill can call other skills, higher or lower in the heterarchy.

As of this day, there exists no successful realization of Minsky's *Society of Mind*.

We will now introduce concepts and methods developed in order to build intelligent agents, capable of learning elementary skills like these required for stacking toy bricks. We will first define the conceptual framework used for the study of learnt robotic control called Markov Decision Process (MDP) and present evidences that similar mechanisms exist in the brain. Secondly we will introduce *Deep Learning*, a technique which recently witnessed success in the field of Machine Learning and serving as the cement of diverse learning algorithms. We will show how Deep Learning can be used specifically for robotic control.

2.2 Extrinsically Motivated Reinforcement Learning

2.2.1 Definition

Learning tasks are generally grouped in 3 categories.

Supervised learning addresses the problem of, given a data-set of (x, y) pairs, learning the mapping from x to y such that the resulting system can generalize to unseen x. This includes for example classification of images, in which case the x are images and the y are labels representing the content of the picture.

Unsupervised learning aims at finding and exploiting structures in a set of data. This covers for example the tasks of finding clusters of similar data points, the task of learning an abstract compressed representation of the data, or the task of generating fake new data, different yet not differentiable from the original data points.

Finally, Reinforcement Learning is a more specific class of problems involving an agent evolving in an environment. The task is formalized as a Markov Decision Problem (MDP) as follows:

In the environment, an agent observes *states* $s \in S$, starting from an initial state sampled from the distribution $I(s)$, and can perform *actions* $a \in A$. The environment has an unknown transition dynamic, not necessarily deterministic, described by the transition probability density function $T(s'|s,a)$ indicating the probability of transitioning from state s to state s' when performing action a. Finally, upon transition between two states, the agent receives a measure of the quality of his behavior called reward. It is determined by the function $R(r|s,a,s')$ denoting the probability of getting the reward r when transitioning from state s to s' via action a.

The task of Reinforcement Learning is for the agent to learn which actions to take depending on the states, in order to maximize the total amount of reward received in its lifetime. We commonly define a policy function $\pi(a|s)$ quantifying the probability of taking action a while in state s. Solving the task for the agent thus means finding the function π that will maximize the expected total reward

$$R^{\pi} = \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right] \tag{2.1}$$

summing the rewards r_t observed when starting from state $s_0 \sim I$ and sampling actions from the policy π thereafter. This quantity is referred to as the return.

Note however that this infinite sum is neither guaranteed to converge, nor to be finite. It is common in Reinforcement Learning to introduce a discounting term to enforce convergence. The discounted return is defined

$$R_\gamma^\pi = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{2.2}$$

and depends on the coefficient $\gamma \in [0,1]$ called discount factor controlling how much the agent must anticipate.

Sometimes, the tasks to be solved are better described by *Goal-based* MPDs. In Goal-based MDPs, the task is parameterized by a goal $g \in G$ and the reward r follows a distribution $R(r|s,a,s',g)$. A typical use of *Goal-based* MDP is to define the goals such that they can be compared against the states, and the reward indicates how close the goal g and the state s are, thus encouraging the agent to reach the goal. The policy is also parameterized by the goal g and the distribution of the next action a when in state s and pursuing goal g is $\pi(a|s,g)$. *Goal-based* MDPs can be transposed into regular MDPs by interpreting the states as the union of the states and the goals.

Two functions called *value functions* are central in all approaches to solving MDPs.

- The state value function $V^\pi(s)$ representing the expected (and possibly discounted) return observed when applying the policy π starting from state s.

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi\right] \tag{2.3}$$

- The state-action value function $Q^\pi(s,a)$ representing the expected (and possibly discounted) return observed when applying the policy π starting from state s and after applying the first action a.

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right] \tag{2.4}$$

We will now derive two equations from the value functions called the Bellman equations which define a recursive relationship between the value in one state and the value in the next state:

- For the state value function,

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^{t} r_t | s_0 = s\right] \\
&= \mathbb{E}\left[r_0 + \gamma\sum_{t=1}^{\infty}\gamma^{t-1} r_t | s_0 = s\right] \\
&= \mathbb{E}\left[r_0 | s_0 = s\right] + \gamma\mathbb{E}\left[\sum_{t=1}^{\infty}\gamma^{t-1} r_t | s_0 = s\right] \\
&= \mathbb{E}\left[r_0 | s_0 = s\right] + \gamma\int_{a\in A}\pi(a|s)\int_{s'\in S} T\left(s'|a,s\right)\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^{t} r_t | s_0 = s'\right] da\, ds' \\
&= \mathbb{E}\left[r_0 | s_0 = s\right] + \gamma\int_{a\in A}\pi(a|s)\int_{s'\in S} T\left(s'|a,s\right) V^{\pi}\left(s'\right) da\, ds' \\
&= \int_{a\in A}\pi(a|s)\int_{s'\in S} T\left(s'|a,s\right)\int_{r\in\mathbb{R}} R\left(r|s,a,s'\right)\left(r + \gamma V^{\pi}\left(s'\right)\right) da\, ds'\, dr \qquad (2.5)
\end{aligned}$$

The Bellman equation for the state value function can also be expressed using the expected value operator

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{a\sim\pi(.|s),s'\sim T(.|a,s),r\sim R(.|s,a,s')}\left[r + \gamma V^{\pi}\left(s'\right)\right] \\
&= \mathbb{E}_{a,s',r\sim p^{\pi}(.|s)}\left[r + \gamma V^{\pi}\left(s'\right)\right] \qquad (2.6)
\end{aligned}$$

with $p^{\pi}(a, s', r|s)$ the probability of taking action a, and transitioning to state s' with a reward r given that we start from state s and follow policy π.

- Similarly for the state-action value function we can show that

$$\begin{aligned}
Q^{\pi}(s,a) &= \int_{s'\in S} T\left(s'|a,s\right)\int_{r\in\mathbb{R}} R\left(r|s,a,s'\right)\left(r + \gamma V^{\pi}\left(s'\right)\right) ds'\, dr \\
&= \mathbb{E}_{s'\sim T(.|a,s),r\sim R(.|s,a,s')}\left[r + \gamma V^{\pi}\left(s'\right)\right] . \qquad (2.7)
\end{aligned}$$

The Bellman equations are useful for deriving rules to find an approximation of the optimal policy denoted π^* that maximizes the (discounted) return. The optimal policy is defined

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\,\mathbb{E}_{\pi,s\sim I}\left[\sum_{t=0}^{\infty}\gamma^{t} r_t | s\right] . \qquad (2.8)$$

Knowing that π^* is optimal, tells us that in each state it chooses the action that maximizes the future return

$$\begin{aligned}\pi^*(s) &= \underset{a\in A}{\operatorname{argmax}}\, \mathbb{E}_{\pi^*}\left[\sum_{t=0}^{\infty}\gamma^t r_t | a, s\right] \\ &= \underset{a\in A}{\operatorname{argmax}}\, Q^{\pi^*}(s,a)\ . \end{aligned} \tag{2.9}$$

If we now apply the Bellman equation we get

$$\pi^*(s) = \underset{a\in A}{\operatorname{argmax}}\, \mathbb{E}_{s'\sim T(.|a,s),r\sim R(.|s,a,s')}\left[r+\gamma V^{\pi^*}(s')\right]\ . \tag{2.10}$$

Given either one of Q^{π^*} or V^{π^*}, equations 2.9 or 2.10 enable us to derive the optimal policy π^*. Reinforcement Learning algorithms try to construct approximations of those functions, from which good policies can be obtained.

Before presenting methods for building these approximations, we will quickly review how the concepts explained above relate to biological mechanisms in the brain.

2.2.2 Analogy with Reward Mechanisms in the Brain

The Markov Decision Problem formulation is very general, and was modeled to describe a class of problems that learning beings face continuously in their life. Understandably, the ability to maximize *"the quality"* of a behavior in an environment represents a big evolutionary advantage, as it is increasing the species' adaptation power. Learning through rewards is one of the ways animals adapt their behavior throughout their life, and neuroscientists recently discovered new mechanisms in the brain involved in such learning.

In particular, measurement in primates primary somatosensory, pre-motor and motor cortex, show evidences that the brain also makes its own estimation of the return [126, 107, 108, 79]. [136] for example recorded the activity of a population of neurons in M1 during a Cued Centered Out task (CCT) where monkeys have to place an object on a certain target to get a reward. Measurements show that as the monkeys get used to the task, some neurons' activity pattern in the population being recorded tend to resemble the value functions. Even more surprisingly, other neurons' activity correlates well with the so-called TD-error, a measure of the value functions' error.

Inspiring from advances in the field of Reinforcement Learning, [25] goes a step further, showing evidences that the brain represents TD-errors not as a single scalar quantity, but rather as a distribution. The computational counterpart of this model is called Distributional

Reinforcement Learning (DRL). In DRL, agents make multiple estimations of the return with varying levels of optimisms. To quantify the resemblance between the biological observations and the computational model, scientists tested several predictions of DRL using single-unit recordings in the ventral tegmental area of mice performing tasks with probabilistic rewards. Specifically, they predicted that different dopaminergic neurons encoding the TD-error should have different reversal points (threshold at which the neuron encodes a zero TD-error). By controlling the distribution of the quantity of sugar delivered and comparing with the cells activity measurements, they discovered a range of TD-error-encoding cells, from optimistic cells that reversed between the two smallest rewards to others reversing between the two biggest rewards.

Finally, *Prefrontal cortex as a meta-Reinforcement Learning system* [145] proposes an advanced model of Reinforcement Learning in the prefrontal cortex (PFC) and shows how biological neural networks might not only implement RL via the tuning of the synaptic strengths, but also through the activity of the cells. In short, as recurrent neural networks can learn algorithms, they can learn a Reinforcement Learning algorithm. This metatheatre is referred to in the RL literature as *meta Reinforcement Learning*. The learnt RL algorithm shows different properties than the traditional one. In particular, meta Reinforcement Learning shows quicker learning and more adaptability to unseen tasks.

In the last decade, neuroscience and the field of Machine Learning with neural networks inspired each other. The interface between both remains thin, it is however today recognized that independently of the biological plausibility of the algorithm chosen, Reinforcement Learning methods constitute a good way to model cognitive processes involving learning through rewards.

We will now present two methods to construct an approximation of the value functions V^π and Q^π.

2.2.3 The Approximation of the Value Functions

Monte-Carlo Method

A simple method to solve MDPs is called Monte-Carlo (MC). It is applicable in the case of episodic tasks, i.e. when the lifetime of the agent is limited to a constant N. MC methods aim at building an approximation of the state-action value function Q^π for a policy π by simulating the agent's behavior when following π, and then update the policy according to the current estimate of Q^π, according to equation 2.9.

In one iteration of MC, starting from k initial states sampled from I, k episodes are simulated following the current policy π and the consecutive states, actions and rewards are

recorded. For each episode, the true discounted return can be computed from the knowledge of the subsequent rewards. The data samples can then be grouped by (s,a) pairs and averaged, giving an approximation of Q^π, from which we can compute an improvement of π, according to equation 2.9.

MC methods have a few limitations however. First, they are practicable only for small and finite MDPs. A second problem is that the convergence when the returns have a high variance is slow.

Temporal Difference Method

A second method to solve MDPs is called Temporal Difference learning (TD). TD-learning builds an approximation of value functions by taking advantage of the Bellman equations 2.6 and 2.7.

Indeed,

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(.|s),s'\sim T(.|a,s),r\sim R(.|s,a,s')}\left[r+\gamma V^\pi\left(s'\right)\right], \tag{2.6}$$

therefore $r+\gamma V^\pi(s')$ is an unbiased estimate of $V^\pi(s)$ (they have the same expected value). We can thus iteratively construct an approximation of V^π denoted $\tilde{V}^\pi$ by applying the update rule

$$\tilde{V}^\pi(s) \leftarrow (1-\alpha)\tilde{V}^\pi(s) + \alpha\left[R(s,\pi(s)) + \gamma\tilde{V}^\pi\left(s'\right)\right], \tag{2.11}$$

meaning that $\tilde{V}^\pi(s)$ is slowly moving toward its own, more accurate estimate. The corresponding approximation method also exists for the Q^π function. In that case, the update rule is given by

$$\tilde{Q}^\pi(s,a) \leftarrow (1-\alpha)\tilde{Q}^\pi(s,a) + \alpha\left[R(s,a) + \gamma\tilde{Q}^\pi\left(s',\pi\left(s'\right)\right)\right]. \tag{2.12}$$

TD methods are fundamentally different from MC methods in that they can update the approximation of the value function and the policy at the same time. TD methods are performing better than MC methods when the return has a high variance. They are also practicable for bigger MDPs.

An intermediate approach between Temporal Difference learning and the Monte-Carlo methods builds more accurate estimations by computing the so-called n-steps return estimate

$$\tilde{R}_n(t) = \sum_{i=t}^{t+n-1} \gamma^{i-t} r_{t+i} + \gamma^n V^\pi(s_{t+n}) \text{ for the state value function and} \tag{2.13}$$

$$\tilde{R}_n(t) = \sum_{i=t}^{t+n-1} \gamma^{i-t} r_{t+i} + \gamma^n Q^\pi(s_{t+n}, a_{t+n}) \text{ for the state-action value function} \tag{2.14}$$

In that case, the update rules for the value function's estimates become

$$\tilde{V}^\pi(s) \leftarrow (1-\alpha)\tilde{V}^\pi(s) + \alpha\left[\sum_{i=t}^{t+n-1} \gamma^{i-t} r_{t+i} + \tilde{V}^\pi(s_{t+n})\right] \text{ and} \tag{2.15}$$

$$\tilde{Q}^\pi(s,a) \leftarrow (1-\alpha)\tilde{Q}^\pi(s,a) + \alpha\left[\sum_{i=t}^{t+n-1} \gamma^{i-t} r_{t+i} + \tilde{Q}^\pi(s_{t+n}, a_{t+n})\right]. \tag{2.16}$$

Both MC methods and TD methods rely on the ability of numerically approximating a function. Technically, Reinforcement Learning algorithms make use of numerical methods for doing so. Recently, a method called *Deep Learning (DL)* or *Differentiable Computing* gained in popularity. Interestingly, Deep Learning shares common features with biological information processing in the brain.

We will now introduce the general principles of Deep Learning.

2.2.4 From Linear Regressions to Differentiable Computing

In 1957 , the American psychologist Frank Rosenblatt published for the first time a learning algorithm called *perceptron* which aims at reproducing the behavior of neurons in computers [111]. Five years later, he published a book called *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* [112], in which he describes how this algorithm enabled him to classify images, estimate the size, distance or position of objects, or to detect relations between objects. The artificial neurons were binary units, organized in two successive layers. His research was described by the traditional press as a revolutionary technological advancement.

It however met sharp criticism. Marvin Minsky and Seymour Papert jointly published a book in 1969 called *Perceptrons: An introduction to computational geometry* [81], where they prove that perceptrons can only approximate linear functions and are therefore not suited for more complex non-linear problems. *A sociological study of the official history of the perceptrons controversy* [90] claims that this book steered the research funding away from neural-imitation based algorithms, back to a more mainstream, symbolic AI research line.

With the allocation of more research credit to AI in the early 80's, new researchers regained interest in nero-based learning techniques. Quickly, Rosenblatt's binary neurons got replaced by continuous versions and other changes made it possible for such networks to approximate non-linear functions. Today, these function approximators find applications everywhere in Machine Learning, and in particular in Reinforcement Learning, where they are used to learn the V^π or Q^π value functions.

Let us review briefly the development of these techniques before applying them to the resolution of MDPs.

A neuron in a neural network achieves something similar to a linear regression. Linear regressions aim at modeling linearly the relationship between variable quantities and a scalar response. Let $\{x_{i0}\dots x_{ip}\}_{i<n}$ be a collection of n samples of the p variable quantities and $\{y_i\}_{i<n}$ be the observed, associated scalar response. Performing a linear regression consists in optimizing coefficients $\{w_0\dots w_p\}$ and b such that the relation $y'_i = b + \sum_j w_j x_{ij}$ approximates best the scalars y_i for all i. There exists many methods to find the w and b parameters, among which the maximum likelihood estimation [113], the method of moments, Bayesian approaches or the method of least squares [67].

The biological interpretation of linear regressions sees the x_js as the excitation of the dendrites of the neuron, which are then re-scaled by the synaptic weights w_js. Electric impulses are then summed at the cell body, and the bias term b is added thus symbolizing the intrinsic excitability of the cell. The processed signal y' is then transmitted along the axon. The tuning of the parameter w corresponds to synaptic plasticity and that of b to changes in the cell's intrinsic excitability.

Multiple scalar values responding to the same variable quantities can be linearly approximated by multiple one-dimensional linear regressions in parallel. This can be interpreted as multiple neurons with different synaptic weights w all excited by the same electric signal x. Such a stack of neurons is commonly referred to as a *layer*. Formally, this is done by replacing the vector w by a matrix W and the scalar b by a vector:

$$y'_i = Wx_i + b \tag{2.17}$$

This simple construct can efficiently approximate multidimensional linear functions. In order to approximate non-linear functions, the ensemble of neurons must be given more expressibility. The principle of recent approaches is - much like in the brain - to construct a network of neurons, where each sends its information to neighbouring units. Note however that if we dispose neurons in two successive layers like Rosenblatt did, the outputs of the firsts being inputs of the second,

$$y_1 = W_1 x + b_1, \tag{2.18}$$
$$y_2 = W_2 y_1 + b_2 \tag{2.19}$$

the overall equation of the network can be rewritten

$$y_2 = W_2 W_1 x + W_2 b_1 + b_2, \tag{2.20}$$
$$y_2 = W_{1\times 2} x + b_{1\times 2} \tag{2.21}$$

as it is the composition of two linear functions. The resulting equation is still linear and the network did not gain in representation power. The discovery that enabled to construct much more expressive networks is to add a non linear transformation of the signal between the two layers. Initially, in order to constrain the neurons output in $]0,1[$, scientists used a sigmoidal transformation $\sigma(x) = \frac{1}{1+e^{-x}}$:

$$y_1 = \sigma(W_1 x + b_1), \tag{2.22}$$
$$y_2 = W_2 y_1 + b_2 \tag{2.23}$$

The use of a non linearity also required to change the neuron's training mechanism. Most modern approaches are based on gradient descent: given a differentiable objective function L of the network output (called loss) which we want to minimize, one defines a training step as the update of all free parameters θ according to the rule

$$\theta \leftarrow \theta - \lambda \frac{\partial L}{\partial \theta}(x) \tag{2.24}$$

where λ is a coefficient called *learning rate* and θ is in the present case the union of W_1, W_2, b_1 and b_2.

Research about this type of neural networks quickly discovered that the expressivity increases as more layers are stacked sequentially. The number of layers is referred to as the *depth* of the network. Recently (2016), [103] showed that the global curvature of a deep neural network increases exponentially with the depth of the network under certain conditions. Until the end of 2015 however, architectures were limited by a problem known as vanishing gradient. As the derivative is computed from the output to the input, it tended to

diminish down exponentially fast to values nearly equal to zero, thus rendering the learning impossible. This issue got resolved completely with the use of a different non-linearity between layers called ReLU [106], the use of *Adam*, a more efficient update rule than the one showed in equation 2.24 [55] and a normalization technique which resembles a homeostasis mechanism of the intrinsic excitability of biological neurons called *batch normalization* [48].

In parallel, scientists explored new ways of connecting together neurons in order to form networks. The connectivity of a neural network is commonly referred to as its architecture or its topology. Notably, a special kind of layer has been designed to efficiently process data with a spatial structure such as images. Called convolutional neural networks [16], the resulting networks exploit spatial locality and shift-invariance in the input data, tremendously reducing the number of required free parameters. An other axis of research focused on the development and study of techniques to process time-series with neural networks. By incorporating so called *backward connections* between neurons, enabling to create loops of information, networks are given a form of short term memory. Much like traditional neural networks approximate functions, these network called recurrent neural networks approximate algorithms. Long-Short Term Memory neural networks are the most notorious kind of recurrent networks [45]. They were designed specifically to extend the duration in time of the network's short term memory.

These successes enabled researchers to design new algorithms solving more and more complex tasks. Recent advances include generative models like Generative Adversarial Networks [34, 2, 54], abstract representation learning with Variational Auto-Encoders [1, 44, 110], Neural Turing Machines [35] learning to read and write in differentiable memory, and various robotic control algorithms, subject of the next sections.

2.2.5 Deep Reinforcement Learning

In Reinforcement Learning, neural networks serve as powerful function approximators for learning the V^π or Q^π value functions. Let's have a closer look at recent Reinforcement Learning algorithms.

Deep Q-Network (DQN)

Deep Q-Network (DQN) [83] is one of the simplest RL-algorithms. It is a TD method where the value function Q is approximated by a deep neural network. It is applicable when the set A of actions is discrete and finite. We will thus write $A = \{a_i\}, i \in \mathbb{N}_n$. DQN aims at approximating the state-action value function Q^π using 1-step TD estimates (Eq 2.12). To this end, a neural network takes as input the states s and returns one value per action

$$Q_\theta(s) = \begin{pmatrix} q_0 \\ \vdots \\ q_n \end{pmatrix} . \tag{2.25}$$

where each q_i is meant to approximate $Q^\pi(s, a_i)$. For simplicity, we will also define the function of two variables

$$Q_\theta(s, a_i) = q_i \text{ for each } a_i \in A . \tag{2.26}$$

The policy is defined with respect to Q_θ as

$$\pi_\theta(s) = \underset{a}{\operatorname{argmax}} Q_\theta(s, a) \quad , \tag{2.27}$$

meaning that it chooses the action corresponding to the highest predicted return.

To train the network Q_θ, we must collect data in the environment and construct the return estimates $\tilde{Q}^\pi(s, a)$. The data collection is made following a second policy, called the *exploration policy* and defined

$$\pi_e(s) = \begin{cases} \text{random action} & \text{with probability } \varepsilon \\ \pi_\theta(s), & \text{otherwise} \end{cases} \tag{2.28}$$

Its purpose is to ensure that all actions are tried in each state, such that all q_is can converge to their target value $Q^\pi(s, a_i)$.

Following the exploration policy, we can construct *trajectories* of transitions (s, a, r)

$$(s_0, a_0, r_0), (s_1, a_1, r_1), \ldots, (s_N, a_N, r_N) \tag{2.29}$$

from which we can compute the 1-step TD return estimates

$$\tilde{Q}^\pi(s_i, a_i) = r_i + \gamma Q_\theta(s_{i+1}, a_{i+1}) \tag{2.30}$$

The neural network is then trained in order to minimize the loss

$$L(s_i, a_i, \theta) = \left(\tilde{Q}^\pi(s_i, a_i) - Q_\theta(s_i, a_i)\right)^2 \tag{2.31}$$

with respect to the weights θ, bringing the network output closer to the 1-step return estimate.

We described here the DQN algorithm as it is in its original publication [83]. It has been shown however that, in this formulation, the DQN algorithm is unstable. Indeed, algorithms

based on Q-learning have been proved to be unstable in many environments. This is a direct consequence of 1-step TD learning, prone to divergence as the Q function is updated according to its own estimate in the next iteration (see eq. 2.12 and 2.30). To addresses this problem, the authors propose in a second publication [84] to use different function approximators for learning (a) the function Q and (b) the 1-step target value towards which Q converges. In practice, the weights of the second Q network (called the target Q network) are not trained with gradient descent, but rather slowly follow the weights of the Q network.

The 1-step TD return estimate from equation 2.30 therefore becomes

$$\tilde{Q}^{\pi}(s_i, a_i) = r_i + \gamma Q_{\theta'}(s_{i+1}, a_{i+1}) \tag{2.32}$$

with $Q_{\theta'}$ the target network, of which the weights θ' follow the update rule given by

$$\theta' \leftarrow \tau\theta + (1-\tau)\,\theta' \tag{2.33}$$

with τ a small number defining the speed at which the weights θ' are following θ. This dampening has been shown to prevent divergence due to the unstable nature of the 1-step TD learning rule.

DPG, DDPG and TD3

DQN is not practicable for continuous action spaces as it requires a global maximization of $Q^{\pi}(s,a)$ with respect to a at every step. A solution to this problem is to directly learn the policy function $\pi_\theta(a|s)$. Algorithms of this sort are called *actor-critic* methods, as opposed to *value-based* methods like DQN.

In particular, Deterministic Policy Gradient (DPG) [129] is the root of a family of actor-critic methods. Let us now briefly review the history of the DPG family.

Assuming that the agent's policy π is parameterized by a vector $\theta \in \mathbb{R}^n$, we can define a measure of the performance of π_θ as

$$J(\pi_\theta) = \mathbb{E}_{s_0 \sim I}\left[V^{\pi_\theta}(s_0)\right] . \tag{2.34}$$

It has been shown in [134] that the derivative of the policy's performance with respect to its parameters - the policy gradient - can be expressed

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^{\pi}, a\sim\pi_\theta(.|s)}\left[Q^{\pi_\theta}(s,a)\,\nabla_\theta \log \pi_\theta(a|s)\right] \tag{2.35}$$

where $\rho^{\pi}(s)$ is the improper discounted stationary state distribution

$$\rho^{\pi}\left(s'\right) = \int_{s\in S} \sum_{t=0}^{\infty} \gamma^t I\left(s\right) p\left(s \rightarrow s', t, \pi_\theta\right) ds \tag{2.36}$$

with $p\left(s \rightarrow s', t, \pi_\theta\right)$ the probability of transitioning from state s to state s' in t iterations.

DPG further assumes that the policy π_θ is deterministic and can therefore be expressed as a function $\mu_\theta\left(s\right) = a$. They then show that under this condition, the deterministic policy gradient can be expressed

$$\begin{aligned} \nabla_\theta J\left(\mu_\theta\right) &= \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_a Q^{\mu_\theta}\left(s,a\right)\nabla_\theta \mu_\theta\left(s\right)\right] \\ &= \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta Q^{\mu_\theta}\left(s,\mu_\theta\left(s\right)\right)\right] . \end{aligned} \tag{2.37}$$

This implies that training the parameters θ of μ_θ to maximize the value of $Q^{\mu_\theta}\left(s,\mu_\theta\left(s\right)\right)$ at data points s sampled from ρ^μ increases the performance of the policy μ as defined in equation 2.34.

The DPG algorithm exploits equation 2.37 by training the networks $Q_w\left(s,a\right)$ and $\mu_\theta\left(s\right)$ to minimize the losses

$$\begin{aligned} L_Q &= \left(Q_w\left(s,a\right) - \tilde{Q}^{\mu_\theta}\left(s,a\right)\right)^2 \\ &= \left(Q_w\left(s,a\right) - \left(r + \gamma Q_w\left(s', \mu\left(s'\right)\right)\right)\right)^2 \qquad \text{with respect to } w \text{ and} \qquad (2.38) \\ L_\mu &= -Q_w\left(s,\mu_\theta\left(s\right)\right) \qquad \text{with respect to } \theta. \qquad (2.39) \end{aligned}$$

where $\tilde{Q}^{\mu_\theta}\left(s,a\right) = r + \gamma Q_w\left(s', \mu\left(s'\right)\right)$ is the one-step TD estimate of the state-action value function, meaning that the policy's actions are pushed in the direction of increasing return estimate.

In practice, it is more efficient to - like for the DQN - collect data from the environment using an exploratory policy different from the behavior policy. Typically the exploratory policy is defined with respect to the behavior policy as

$$\pi\left(a|s\right) = \mathcal{N}\left(a|\mu_\theta\left(s\right), \sigma\right) \tag{2.40}$$

where σ is a parameter to control the level of exploration around the behavior policy.

In its original publication, DPG was presented as a proof of concept, showing that policies could be obtained using the deterministic policy gradient from equation 2.37. It however

has been shown that DPG is too unstable for solving complex MDPs, partly due to the impossibility to use large function approximators for Q and μ.

The DDPG algorithm [71] is an extension of DPG which solves this issue by bringing two enhancements to the approach.

The first is to make the use of a replay buffer from which the training data is sampled. Most optimization techniques for neural networks assume that training samples are independently and identically distributed. In the case of Reinforcement Learning however, as the data is being generated by the agent following its policy, this assumption is wrong. The solution to this problem is to place the data in a buffer after it has been collected from the environment, and then sample from it to train the function approximators. This way the networks are presented with data coming from different episodes and distant in time. This technique is also more sample efficient, as it enables to train the neural networks multiple times on the same data. This is done by increasing the frequency at which we sample from the buffer, and the number of data points sampled in each iteration.

As the DQN algorithm, the algorithms of the DPG family are based on Q-learning, and therefore suffer from instabilities as the Q function is updated according to its own estimate in the next iteration (equation 2.12). Similarly to the DQN [84], the authors of the DDPG algorithm propose to use different function approximators for learning Q and for learning the target value towards which Q converges. The loss from equation 2.38 therefore becomes

$$\begin{aligned} L_Q &= \left(Q_w(s,a) - \tilde{Q}^{\mu_\theta}(s,a)\right)^2 \\ &= \left(Q_w(s,a) - \left(r + \gamma Q'_{w'}\left(s', \mu'_{\theta'}\left(s'\right)\right)\right)\right)^2 \end{aligned} \tag{2.41}$$

where $Q'_{w'}$ and $\mu'_{\theta'}$ are referred to as the critic and actor target networks. They are slower-moving version of the networks Q_w and μ_θ whose weight update rules are given by

$$w' \leftarrow \tau w + (1-\tau) w' \text{ and} \tag{2.42}$$

$$\theta' \leftarrow \tau \theta + (1-\tau) \theta' \tag{2.43}$$

with τ a small number defining the speed at which the weights w' and θ' are following w and θ.

DDPG is shown to be much more efficient than DPG and has been applied to challenging problems like complex robotic tasks as well as continuous control tasks from raw pixel data similar to [83]. However it also revealed its own limitations. In particular, like for

many others Q-learning based algorithms, DDPG tends to overestimate the action state value function. Indeed, unavoidable inaccuracies in the approximation of Q^{μ_θ} lead to favor the choice of actions exploiting these inaccuracies. It results that the 1-step TD estimate is biased towards optimistic estimations, which are then propagated backward in time due to the nature of the TD update rule.

The TD3 algorithm [31] proposes a solution to the overestimation bias, by inspiring from an older algorithm called Double Q-learning [42]. In TD3, the estimation of the action state value function Q^{μ_θ} is defined as the most pessimistic estimation of an ensemble of networks. In practice, an ensemble of size 2 is sufficient to counteract the overestimation bias. The loss function for the critic Q_w becomes

$$\begin{aligned} L_{Q_{w_i}} &= \left(Q_{w_i}(s,a) - Q^{\mu_\theta}(s,a)\right)^2 \\ &= \left(Q_{w_i}(s,a) - \left(r + \gamma \min_{i=1,2} Q'_{w'_i}\left(s', \mu'_{\theta'}(s')\right)\right)\right)^2 \end{aligned} \tag{2.44}$$

2.3 Applications

We introduced the reinforcement-learning class of problems, and presented theoretical and technical solutions to solve MDPs. We will now present applications of Reinforcement Learning to robotic control. However, DQN and the DPG-inspired algorithms explained above constitute only a small fraction of all Reinforcement Learning algorithms and we will therefore not limit ourselves to these algorithms. Other algorithms include Soft Actor-Critic (SAC) [41]; Natural Actor-Critic (NAC) [100] ; Trust Region Policy Optimization (TRPO) [124]; Proximal Policy Optimization (PPO) [125]; Relative Entropy Policy Search (REPS) [99] and Hierarchical Relative Entropy Policy Search (HiREPS) [26]; Continuous Actor-Critic Learning Automaton (CACLA) [141]; Asynchronous Advantage Actor-Critic (A3C) [82]; Normalized Advantage Function (NAF) [37].

Applying Reinforcement Learning to robotic control tasks is challenging and poses a few problems. *Reinforcement Learning in Robotics: A Survey* [59] details the difficulties in porting Reinforcement Learning algorithms to real-world robotic agents. The first obstacle presented in the survey resides in the *continuous* nature of the states and actions of robots, necessitating to carefully decide how fine is the control that we need over the robot, whether to use discretization or continuous function approximations and at what frequency should actions be generated and applied by the robot. The second difficulty presented in the survey is that of the *Curse of Dimensionality* [13]. It describes how, as the dimensionality of the state

space or of the action space increases, the complexity of the problem increases exponentially. Indeed, if we assume that an agent has n continuous actions available, and if we decide to discretize these actions in 10 bits, the number of different combinations of actions is 10^n. Finally, complexity arises from the inherent physics of the real-world and of the robot. Robots are often brittle and therefore can't take risks when exploring their environment, repairs are expensive and often imply long waiting times. The experimental conditions may also vary during the time of training. Some motors are sensitive to changes in temperature, lighting conditions may vary and affect the visual information received by the robot, the robot may wear out over the course of training etc. One notable difficulty induced by physical constraints is the delay between the generation of a command, its execution and the sensory feedback received by the agent. When applied to robotics tasks, Reinforcement Learning algorithms must take into account all these obstacles. It results that only few publications relate pure applications of Reinforcement Learning algorithms to robots. Most incorporate additional optimizations specifically designed to solve the aforementioned complications. However, original publications presenting new Reinforcement Learning algorithms - in particular algorithms for continuous state and action spaces - often showcase the performance on a simulated robotic task. Simulated robots offer multiple benefits over real robots, although simulations do not capture the full complexity of real-world applications. It enables to leave out the complexity arising from physics and to simulate robots faster than real-time at a much lower financial cost.

We will try to review here the successes of Reinforcement Learning algorithms applied to robotic control tasks, both on real robots and in simulations, limiting ourselves to publications minimizing the additional complexity induced by optimizations specific to robotic problems. We will group publications by the nature of the robotic task being solved.

2.3.1 Reaching

The most canonical robotic control task is without a doubt the *reaching* task. In this case, the task is parameterized by a target location in the environment that the agent must reach with its *end-effector*. The reward for that task can be defined in two ways. In the simpler version, it is defined as the negative distance to the target location. The reward is thus non zero all the time and it is easy for the agent to determine after each motion if the action affected the reward positively or negatively. The second option is to reward the agent only once it reached the target location. This task is harder as the information about the quality of the motion is delayed. In the first case we describe the reward as dense, as it changes value at each iteration, in the second case the reward is described as being sparse, meaning that the

reward is constant and negative all the time but upon successful completion where the reward is positive.

Reaching tasks often serve as benchmarking tasks for Reinforcement Learning algorithms. More complex variants include the avoidance of obstacles in the task.

In [38, 72], the reward is expressed in its dense variation and the agent is given successful demonstrations from which it must learn. Both incorporate obstacles that the agent must avoid in order to reach the target location. These early work used Gaussian Mixtures Model obtained from the demonstrations as movement primitives, the reward additionally contains a term encouraging movements close to the mean of the gaussian distribution. The approach is applied on real robots. The Reinforcement Learning algorithm used in [38] is the NAC algorithm [100].

In [76], the authors propose a comparison of dense and sparse rewards and investigate the benefit of iteratively making the task more complex, a technique known as *curriculum learning*. A video of the learnt policy is available following this link[1]. In the video, you will see the robot arm getting close to objects in a so-called *pre-reaching* state. The agent then hands over the control to a second algorithm designed specifically to perform grasping. Finally the robot arm moves to a second *drop* position using the learnt policy, and the object is released. The Reinforcement Learning algorithm used is a variant of DDPG [71] that the authors named PCCL.

In [101], the agent learns from a dense reward specifically shaped to avoid collisions. The motivation is that collisions between the robot and the objects in the environment often result in either failures of the robot or degradation of the objects. The authors compare multiple strategies for shaping the reward and are able to successfully learn the reaching task without colliding with the environment. They propose a new Reinforcement Learning algorithm called OptLayer, derived from the TRPO algorithm [124]. The approach is validated on a real robotic setup.

In [28], multiple rewards are compared. Notably the agent is trained with a simple dense reward, a sparse reward, and with a more complex sparse reward shaped to exploit previously learned eye-hand coordination. In this research, the agent learns to express the position of the end-effector and that of the target location to be reached in the reference frame of the eyes. By then comparing them, an additional term can be added to the reward of the agent which facilitates learning of reaching. The algorithm used is the DDPG algorithm [71]. The approach is validated on a real robotic setup. A video of the agent performing reaching is available under this link[2].

[1] https://www.youtube.com/watch?v=WY-1EbYBSGo&feature=youtu.be

[2] https://www.frontiersin.org/articles/10.3389/frobt.2018.00110/full#supplementary-material

In [36], the authors investigate the benefit of parallelizing the Reinforcement Learning algorithm to enable training on multiple robots at the same time. The motivation is that RL algorithms often need a lot of environment interactions to converge. The approach is tested on multiple tasks including reaching. The algorithm used is called NAF [37], a video of the agents is available under this link[3].

2.3.2 Picking and Placing

In the *pick and place* task, a robotic arm must displace an object from a known original position to an other known target position. The initial and target positions are randomized in the general form of the task. Usually, the operation of grasping the object is left out by simplifying the interaction between the robotic hand and the object. One way of doing so is to employ an *electro-magnet* type of end-effector which simply attracts the objects to it. An other way of doing is to let the robot push and pull the object on a table. Finally, when the robots are required to grasp the objects in *pick and place* tasks, their grippers are often mechanically designed such that the grasping operation does not require a precise positioning of the end-effector and the set of object to be displaced does not match the variability in shape, size and weight of real world objects.

In the broad sense, *pick and place* tasks include many real world skills like opening a door, screwing a cap on a bottle or stacking blocks like Marvin Minsky did in the early 70's.

[53] proposes a solution specifically designed to (1) take into account the compliance of the robot and (2) speedup learning via expert demonstrations. The robot is controlled in torque mode, as opposed to position or velocity control. The aim is to develop compliant actuation that can operate safely in human environment. The Reinforcement Learning algorithm used is called PI^2 [138] and it is trained from a dense reward. Their approach is applied on two tasks on a real robot: the first task consists in opening a door, the second consists in lifting a pen off a table. This work is one of the first to successfully learn a torque controller using Reinforcement Learning.

The original publications about DDPG and NAF [71, 37] include results of their algorithms when trained on simulated robotic tasks. In both papers, the agent learns to control a 2D robotic arm in order to pick and place objects. In each, two versions of the task are presented. In the first version, the base of the 2D arm is fixed to the ground and the agent must control the joints in order to first reach for an object and then place it at a specified target location. In the second version, the base of the arm is not fixed anymore, and the agent

[3]https://sites.google.com/site/deeproboticmanipulation/

must learn to move it, as well as the joints of the arm, in order to accomplish the same task. The reward given to the agent is not specified in the papers.

In [36], the authors present an asynchronous parallelization schema for Reinforcement Learning algorithms that they apply on the NAF algorithm [37]. Results are presented for training on simulated robots as well as on real hardware. They show that by using multiple physical robots in parallel, it is possible to teach them to perform picking and placing tasks in under 3 hours. The first task is a *door pushing and pulling* task, performed in simulated environments and on the physical robots. The reward function is dense and is composed of two parts: the proximity of the end-effector to the door handle, and a measure of how much the door is opened in the right direction. The second task is a pure *pick-and-place* task. In this setup, a stick is positioned in the air close to the robot's end-effector. The robot must learn to grasp it and move it to a target location. Again, the reward is dense and composed of multiple parts: the distance between the *grip position* and the stick, the distance between each of the fingers' tip position and the stick, and the distance between the stick and the target position. A video of the learnt policies is available under this link[4].

In [69], the authors show that it is possible to learn a range of different *pick-and-place* tasks on a real robot by supplying the robot with visual information, thus forming visuo-motor policies, meaning that the visual information is transformed directly into torque commands. The tasks solved by the robot include placing a hanger on a clothes rack, inserting objects in a shape sorting cube, screwing a cap onto bottle or placing the teeth of a hammer around a nail.

2.3.3 Grasping

The reason why grasping is often left out of *pick and place* tasks is because it is a very challenging problem for robots. Grasping is often learned independently from picking and placing. It constitutes a fully-fledged robotic control task. The difficulties are two-fold. First, there is a big variability of grasping approaches, depending is the size, shape or weight of the object to be grasped and on the intended use of that object. Secondly, grasping is tremendously facilitated by the haptic feedback of the contact, which lacks both to robots and simulators.

The earliest attempt at applying Reinforcement Learning to robotic grasping dates back to 2010 [61]. It uses a hybrid approach intending to reduce the dimensionality of the state and action spaces for faster convergence. The authors combine a custom Reinforcement Learning algorithm (Continuum Gaussian Bandits) generating 6-dimensional hand-poses

[4]https://sites.google.com/site/deeproboticmanipulation/

specifying where to grasp the object, and a DMP controller responsible for the action execution determining how to perform the motion. The 6-dimensional hand-pose can thus be interpreted as an abstract, high level action.

In 2011, similarly to the aforementioned paper performing picking and placing [53], [133] applies the PI² algorithm to the robotic grasping task. The RL algorithm optimizes 2 DMPs to learn to perform (1) a prereach motion and (2) a subsequent grasping motion while keeping the end-effector fixed. The agent learns in simulation from a sparse reward and using a 3-fingered gripper.

In 2018, [104] proposes a benchmark for vision-based grasping algorithms and measures the performance of 6 algorithms including DDPG and Deep Q-Learning. Two tasks are presented: in the first case, a simulated robotic arm must grasp any object from a bin containing 5. In the second, the arm must grasp a particular object out of the 5. The agent receives a reward if the correct object is held in the gripper at the end of a fixed-length episode. The objects are diverse in shape and colors, but not in size. The authors decided to focus on off-policy Reinforcement Learning algorithms and trained them on a fixed, pre-generated dataset of grasps. This enables them to present an interesting comparison of the sensitivity of the algorithms to hyper-parameters, as online learning would have required too many expensive environment interactions. The agent's actions correspond to x, y and z end-effector velocities as well as a ϕ wrist angular velocity around the z axis. The improvements over the previously mentioned older methods are that (1) reaching and grasping are not differentiated anymore in this approach and (2) the policy is learned end-to-end from visual information. A video of the learnt policies is available under this link[5].

[51] proposes a simplistic architecture based on Deep Q-Networks performing grasping both in simulation and on a real robot. The robot has 2 cameras, one overhead projecting over the scene and a second in the wrist. The authors achieve quick and reliable grasping with a clamp gripper of cubic spherical and cylindrical objects.

[86] proposes an application of TRPO to grasping using an anthropomorphic hand with 16 degrees of control. This simulated hand is part of the SouthHampton Hand Assessment Procedure (SHAP) test suite developed to assess the effectiveness of upper limb prostheses. In their approach, the agent only knows about the current position and speed of its joints and 3 numbers denoting the position of the object. They used a dense reward composed of the distance between the palm and the object and a sparse term upon successful grasping. A video of the learnt policies is available under this link[6].

[5]https://sites.google.com/view/grasping-icra2018/home
[6]https://rctn.github.io/deephapticsgrasp/

Finally, the problem of grasping cluttered or densely packed objects on a table is addressed by [154]. They propose a technique to learn to perform pushing movements intending to facilitate subsequent grasping, from visual data. RGB-D information is first converted to a 3D representation of the scene. One Q-Learning network then transforms the latter in a 2D map evaluating for each pixel the quality of accomplishing a pushing motion. A second such network outputs a 2D map evaluating for each pixel the quality of accomplishing a grasping motion. Both networks are trained jointly as a single Q-Network and the choice between pushing or grasping is determined by the highest Q-value. In this approach, the authors leverage the process of learning the pushing and grasping motions, replacing them by Inverse Kinematics based movement primitives: for each pixel in the 2D maps corresponds a push and a grasp. A video of the learnt policies is available under this link[7].

2.3.4 In-Hand Manipulation

Following to grasping, attempts have been made at performing *in-hand manipulation*, that is to say performing rotation of an object in one hand by controlling the movement of the fingers and the wrist. The same way grasping is a useful skill to perform picking and placing, in-hand manipulation is useful to achieve accurate and reliable grasping. Like for grasping, one of the main difficulties with dexterous manipulation is the lack of haptic feedback in virtual reality. Another difficulty lies in the fact that hand manipulations involve a complex contact dynamic with the object which is hard to simulate reliably. Finally, like human hands, robotic hands offer a large number of degrees of control, often greater than 16. Large action spaces have been shown to be much more complex to solve due to the curse of dimensionality. An open line of research consists in finding good dimensionality reductions that can couple joints without loosing of the robot's capabilities.

The very first application of Reinforcement Learning to in-hand manipulation dates back to 2016 [63]. In this research, a robotic hand composed of 24 joints learns to rotate elongated objects using it's fingers. The Reinforcement Learning algorithm used differs slightly from the RL formalism explained earlier in this thesis. The agent does not learn actions to be performed but rather complete sequences, each action in the sequence being a linear transformation of the current state. This results in a sequence of linear models, of which the weights are learned using Reinforcement Learning. A video of the learnt policies is available under this link[8].

A following paper [105] focuses on a simulated robot in order to learn manipulation of tools and re-orientation of object. It investigates the benefit of showing demonstrations

[7] https://vpg.cs.princeton.edu/
[8] https://www.youtube.com/watch?v=Q9VLr_UGjBI

of the task to the agent. The demonstrations are recorded using a mechanical glove that enable to interact with the simulated environment. They exhibit grasping and pick-and-place movements, as well as in hand manipulation. For each task, 25 demonstrations are provided. The RL algorithm proposed is derived from the Natural Policy Gradient algorithm (NPG) and is called Demonstration Augmented Policy Gradient (DAPG). This algorithm is compared against DDPG and the use of reward shaping is properly reported. A video of the learnt policies is available under this link[9].

Finally, [93] published in late 2018 is the first research able to accomplish successful in-hand rotation of a cube using a real robot. It applies PPO with recurrent neural networks (LSTM) in simulation to a robotic hand similar to the hand in [63] offering 24 joints. The agent trained in simulation is then directly ported onto the real robot. This transferability is made possible by a technique called domain randomization, which aims at improving the agent's generalization ability by training it on a wide spectrum of slightly different environment. A video of the learnt policies is available under this link[10].

2.3.5 Locomotion

An other typical robotic control task is the learning of bipedal and quadrupedal locomotion. The difficulty in locomotion tasks often resides in the variability of the geometry of grounds on which the robot must walk, and in the combination of information from very different sensory modalities. Learning locomotion involves learning equilibrioception, which is the result of the visual system, the proprioception and the inner ears working together. The research in RL for locomotion control is mainly driven by robotics companies such as Boston Dynamics or Agility Robotics.

Quadrupedal Locomotion

Similarly to pick-and-placing and grasping[53, 133], the PI² algorithm has been applied to quadrupedal locomotion in 2010 [139]. In this experiment, a quadrupedal, dog-like robot must learn to jump over a gap. To facilitate learning, the algorithm is seeded with an initial behavior learned from demonstration. Additionally, the agent is not only rewarded for the jumped distance but also for keeping the correct roll and yaw angles. Like in [53] and [133], the approach uses DMPs as movement primitives.

Recently, deeper models have successfully been applied to four-legged locomotion. For example [40] applies a modified Soft Actor-Critic (SAC) algorithm to both simulated and real

[9] https://sites.google.com/view/deeprl-dexterous-manipulation
[10] https://www.youtube.com/watch?v=jwSbzNHGflM&feature=youtu.be

robots. The algorithm is chosen to be sample efficient and the authors accomplished training on a real robot in about 2 hours. The proposed modification automates the tuning of the temperature parameter of the SAC algorithm. In this research, the action space is composed of the extension of each leg and their swing angle, resulting in 8 dimensions. The authors found that the latencies in the hardware and the partial observation of the environment make the system non-Markovian. They propose to overcome this problem by augmenting the observation space to include a history of the last five observations and actions. A video of the learnt policies is available under this link[11].

[131] proposes a new approach to transfer a locomotion behavior learned in simulation to a real robotic platform. By decomposing the simulated robot's motion in principal components with a PCA, they define kinematic motion primitives that can be later used on the real robot. Again, the action space is composed of the extension and swing angle of each leg. The RL algorithm used is the PPO algorithm and the reward is composed of a term for maximizing the speed and a second for minimizing the metabolic cost. A follow-up paper [17] investigates in simulation the advantage of having a flexible spine for running. Finally, a third paper [60] investigates the advantage of using Bezier polynomials to output a complete step trajectory instead of a single next foot position. Using this technique, a full stepping motion can be described in 8×4 dimensions. By exploiting conditions on continuity of motion and symmetries of the trotting gait, this dimensionality is further reduced to only 8. Again, videos for each of these 3 publications are available: [17][12] [131][13] and [60][14].

Finally, the most efficient approach as of 2020 [65] presents a novel solution demonstrating zero-shot generalization from simulation to natural environment. Moreover, the agent does not need to have access to any cameras or LIDARs as the behavior is learned solely from proprioceptive information. The first contribution is to replace the policy's Multi-Layer Perceptron (MLP) with a Temporal Convolution Network (TCN) [5]. In this approach, the TCN outputs actions given a history of the proprioceptive states. The second contribution is the use of a technique called privileged learning [23]. The authors report that directly training the Reinforcement Learning agent on challenging terrain without giving it any information about the ground it is stepping on fails to learn locomotion with reasonable time budget. The solution they propose is to decompose this training process in two stages: first they train a teacher agent by giving it ground truth knowledge of the terrain (the privileged information). Second they train a student policy using only sensors available to the real robot, guiding its training with the teacher's policy. Additionally to using TCN and privileged learning, the

[11]https://sites.google.com/view/minitaur-locomotion/
[12]https://www.youtube.com/watch?v=INp4aa-8z2E&feature=youtu.be
[13]https://www.youtube.com/watch?v=kiLKSqI4KhE&feature=youtu.be
[14]https://www.youtube.com/watch?v=aFGM_xWeh3U&feature=youtu.be

authors implemented an automated curriculum. Curriculum learning consists in progressively increasing the difficulty of the task and enables quicker learning of more complex skills. In this particular approach, the difficulty of 3 different types of terrain can be adjusted (hills, steps and stairs) by tuning the width and height of the stairs and steps, and the roughnes, frequency and amplitude of the hills. The agent learns to follow a command received from the experimenter, indicating the desired direction of motion and desired turning direction around the center of mass. To facilitate portage from the simulation to the real world, the simulation uses a learned dynamics model of the robot's joint PD controller following ideas introduced in [47]. This enables them to directly deploy the proprioceptive controller on the real robot with no additional fine-tuning. Importantly, the robot's motion is generated using an hand-crafted leg rhythm. In short, the foot-tips are following a predefined trajectory and the controller outputs a time offset determining if the agent should lag-behind or precede it, and a space offset determining a foot position residual with respect to the predefined trajectory. The resulting foot-tip position is tracked using analytic inverse kinematics (IK). The Reinforcement Learning algorithm used is the TRPO algorithm. A video of the learnt policies is available under this link[15]. The resulting behavior is spectacular in 2020's standards. Notably, a foot-trapping reflex emerges from the training. When failing to climb over a high step, the agent *remembers* it and its next leg movement will reach higher in order to successfully pass the obstacle.

Bipedal Locomotion

The first successful realization of bipedal locomotion using deep Reinforcement Learning dates from 2017 [43]. In this research, a 28 DoF simulated humanoid robot is trained to run over a challenging terrain, involving climbing and jumping over obstacles. The algorithm is a plain implementation of PPO with minimal additional optimizations. The agent is provided with proprioceptive information, as well as extroceptive information like the position with respect to the center of the track it must run on as well as the profile of the terrain ahead. The tracks are designed such that their difficulty increases as the agent progresses towards the end, forming a natural curriculum learning mechanism. The reward consists of 4 components. The first is a term proportional to the velocity along the x-axis, encouraging the agent to make forward progress along the track, the second is a small term penalizing torques, the third penalizes deviations from the center of the track, and the humanoid receives an additional reward per time-step for not falling. A video of the learnt policies is available under this link[16].

[15]https://leggedrobotics.github.io/rl-blindloco/
[16]https://www.youtube.com/watch?v=hx_bgoTF7bs&feature=youtu.be

A series of publications tracks the progress of a research group about bipedal locomotion learning on the Cassie robot [150–152]. Similarly to the previous paper [43], [150] is a plain application of PPO to the Cassie platform, with the nuance that the robot is rewarded for following a nominal trajectory - a reference motion - given by the experimenter. The advantage of this technique is that "the reference motion can be a crude sketch of the desired motion, i.e., it can be unphysical, kinematically inconsistent or having body and foot velocities that are mutually incompatible". Typically, reference motions are obtained from tracking recordings. In this paper, all experiments are conducted in simulation only. The resulting behavior exhibits undesirable motions like shaking of the pelvis. In the follow-up papers [151] and [152], the authors address this issue using a method for refining the learned policy iteratively. This technique enables them to achieve a more stable gait and a reduced energetic demand. Additionally, they show that is is possible to compress the trained policy neural network to a much smaller size using supervised learning, while not hindering the agent's performance. The policies learned in the simulated environment are transferred directly onto the real robotic setup. The authors report that they did not need to learn any dedicated models of the actuators using data collected from the physical robots like it is for example proposed in [47, 153, 65]. Videos are available for each of the publications: [150][17], [151][18] and [152][19].

Deep Reinforcement Learning for bipedal locomotion also finds applications in computer graphics, where the task is to find a physically plausible gait for 3D animation or for video games. For these applications, the physics simulation does not need to be completely accurate. [149] proposes a technique to learn bipedal or quadrupedal locomotion in simulation for varying body shapes. In their approach, the shape of the different body parts is provided in the agent's state, enabling to vary the robot's dimension on the fly during performance of the movement. Similarly to the previous examples of bipedal locomotion, the agent is encouraged to follow a reference motion, as well as matching a target velocity instructed by the experimenter. Finally, a third reward term penalizes motor torques to encourage smoother joint control. Similarly to the previous examples, the actions are joint positions tracked by a PD controller and the Reinforcement Learning algorithm used is PPO. A video of the learned policies is available under this link[20].

Finally, deep Reinforcement Learning has been successfully applied to the control of a virtual model of the entire human body [66]. This enables to simulate musculoskelatal conditions like muscle contractures or bones malformations, to simulate the improvement in

[17]https://www.youtube.com/watch?v=z3DMKQwt68Y
[18]https://www.cs.ubc.ca/~van/papers/2019-CORL-cassie/index.html
[19]https://www.youtube.com/watch?v=TgFrcrARao0
[20]http://mrl.snu.ac.kr/publications/ProjectMorphCon/MorphCon.html

gait resulting from surgery and to design better prostheses. In this research, the body model is composed of 284 muscles connected to a tree of rigid bones. Behavior learning through Reinforcement Learning on a 284 dimensional action space in impracticable, however the reference motion tracking technique described above considerably reduces the part of the action space that must be explored. Surprisingly, this is sufficient to successfully learn locomotion in 36 hours of training on a computer. Again, the Reinforcement Learning algorithm used is the PPO algorithm. Overall, the approach is similar to the ones described above, however the authors propose a 2-levels architecture to coordinate the muscles. The first level consists in a policy transforming the skeletal state and the reference motion in a desired skeletal pose. A PD control mechanism translates this pose into joint accelerations. The second level transforms the muscular state of the agent and the desired joint accelerations into a muscle activation command. A video of the learned policies is available under this link[21].

A table summarizing all the publications referenced in this section is available in the appendix (see table A.1).

2.4 Conclusion

In this chapter, *Overview of learning algorithms for robotic control*, we introduced the Markov Decision Problem formalism and presented evidences that similar mechanisms take place in the brain. We then briefly introduced the concepts of Deep Learning and Differentiable Programming, also relating it to neural processes. We then showed how Deep Learning can be used to solve Markov Decision Processes. Finally, we presented the state of the art in Reinforcement Learning specifically applied to robotic control tasks. The analysis of the literature shows that as of 2020, the promise of versatile robotic arms has still to be implemented. However, recent advances enabled researchers for the first time to successfully achieve robotic locomotion, on 2 or 4 legs, without the use of LiDARs nor cameras.

Learning to manipulate objects in the environment is significantly more challenging than learning locomotion. The comparison is however unfair: arms and hands serve hundreds of different purposes (reaching, grasping, throwing, pushing, pulling, clapping, squeezing, scratching, cleaning, ...) while legged locomotion is only one of the possible uses of robotic legs (like jumping, kicking in a ball ...). The difficulty in solving robotic manipulation comes from the wider spectrum of tasks, but also from the fact that important pieces of information required for solving a task are often unavailable to the agent (precise shape, weight) or must be abstracted from sensor data (distance, brittleness). In that respect, the absence of

[21] http://mrl.snu.ac.kr/research/ProjectScalable/Page.htm

haptic feedback constitutes an additional difficulty yet to be addressed. Finally, additional complexity comes from the fact that most manipulation tasks involve to structure the behavior in sequences of different skills (reaching, grasping) or in a hierarchy of skills (find a block - look, evaluate; stack a block - reach, grasp, lift, stack), thus requiring a way to *abstract* sensory information.

In the next chapter, *Learning Abstract Representations through Lossy Compression of Multi-Modal Signals*, we will study how information from different sensory modalities can be integrated together and how, by doing so, meaningful sensory representation can be obtained.

Chapter 3

Learning Abstract Representations through Lossy Compression of Multi-Modal Signals

3.1 Introduction

When an event occurs around us, we do not capture all the information generated by this event. Our brain receives only a fraction of it, and does so through different sensory channels simultaneously: the sight, hearing, smell, taste, touch, but also proprioception, nociception, localization in space etc. Let's say that an event in the world has been seen and heard, like for example a plate shattering on the floor. In order to construct a sharp mental image of the event that occurred, the brain needs to fuse together the information that has been seen and the information that has been heard. This coalescence is made possible by the fact that some information about the event goes simultaneously through the two senses. In information theory, this is referred to as the *mutual information*. Note that the latter is statistically defined and that it does not exist when considering a single event or a single repetition. By learning to recognize this mutual information, the brain can coalesce together the information coming from the two sensory modalities. In the case of the plate shattering, the vision, together with the proprioception, hold the information about the localization of the event in the referential of the body. Similarly, by performing spectral transformations to incoming sounds, our brain deduces the horizontal and vertical localization of the event. Thus the highly relevant piece of knowledge indicating where the plate shattered in the body's referential frame is contained in the mutual information, shared by the visual and acoustic modalities. Can the mutual

information be algorithmically isolated, and what kind of knowledge about the world can we expect to find in it?

Let us take an information theoretic perspective on the multimodal perception problem. We consider two variables, X and Y, representing the data from two sensory modalities, and we assume that some information, the mutual information, is shared by both modalities X and Y. That means that the knowledge of X gives us also some prior knowledge about Y, and reciprocally. In information theory, scientists are interested in the size, measured in bits or in nats, of the most compact code of a variable. The size of the most compact code representing the variable X is called the entropy of X and is noted $H(X)$. The mutual information is defined as the size of the most compact code that represents the prior knowledge about Y contained in X, which equals the size of the prior knowledge about X contained in Y. It is noted $MI(X,Y)$. An other way to define the mutual information requires to introduce a third variable Z representing the union of the two variables X and Y. Since some information is repeated in X and Y, the most compact code representing Z is shorter than the sum of the two codes representing X and Y. This is noted $H(Z) \leq H(X) + H(Y)$. The difference in code length is called the mutual information:

$$MI(X,Y) = H(X) + H(Y) - H(Z) \quad \text{, also noted} \tag{3.1}$$

$$MI(X,Y) = H(X) + H(Y) - H(X,Y) \tag{3.2}$$

The special case when $H(Z) = H(X) + H(Y)$, and thus $MI(X,Y) = 0$, corresponds to the case where the two variables are statistically independent, i.e. $P(X,Y) = P(X)P(Y)$ or $P(X|Y) = P(X)$ and $P(Y|X) = P(Y)$.

Similarly to how we defined Z as the union of the variables X and Y, we can define a variable W that would encompass only the information which is contained in both X and Y. We can also define the two remaining variables $X_{\setminus Y}$ and $Y_{\setminus X}$ which respectively represent the information which is in X, but not in Y and reciprocally. By definition, $MI\left(W, X_{\setminus Y}\right) = MI\left(W, Y_{\setminus X}\right) = 0$, which means that W, $X_{\setminus Y}$ and $Y_{\setminus X}$ are statistically independent. Thus these 3 variables are holding information about independent aspects of the event that took place, or in other words they represent independent factors of variation of the event. If we go back to the example of the plate shattering, the localization in space, which is present in both modalities, is strictly independent from the shape of the object, which is only seen, or from the loudness of the sound it produced, which is only heard. And these indeed constitute independent properties of the event that took place.

In the research presented in this chapter, we hypothesize that the information that is contained in multiple sensory modalities at a time is statistically independent from the information contained in only one modality, and thus that it must correspond to an independent factor of variation of the world. We believe that a representation of the multimodal sensory information that would identify and isolate the independent factors of variation of the world is highly relevant in the context of RL.

3.2 Related Work

Within the machine learning community, our work belongs to the field of *representation learning* [15, 30, 14, 87]. Representation learning consists in learning a representation of the data that makes it easier to extract *useful information*. Historically, in supervised learning, scientists were hand-crafting feature extractors which would for example, when applied to image classification, detect vertical, horizontal edges, curved lines, crosses etc. Typically, linear classifiers would then use these features as inputs. With the regain of interest for neural networks, engineered feature extractors have been replaced with deep non-linear network architectures. Arguably, if we slice open a deep neural network trained in a supervised manner, we have access to a latent representation of the information. Using various techniques, and notably through a technique known as *feature visualization* [89], scientists have been able to better understand how these learned representations are constructed. They found that neural networks have the ability to abstract the information, and that the learned features often correspond to real-world concepts (see [89]). Potentially, representation learning could find uses in RL.

First, by operating on an abstract *view* of the world, a RL agent would improve its ability to generalize its behavior to other environments. The lack of generalization constitutes one of the major flaws of current RL algorithms [75, 146]. Although the perspective of achieving better generalization in RL is very attractive, past attempts met limited success: when the learning of the representation is completely decoupled from the reward signal, the benefit in generalization is often mitigated by slower convergence speed of the algorithm and lower overall performance. See for example the experiments in the appendix of [132].

Secondly, representation learning might be used in Reinforcement Learning in order to generate rewards for the agent. By using an abstract representation of a state as a goal that the agent must reach, one can then compare it against the abstract representation of the state in which the agent finds itself, and reward the agent according to the distance to the goal, encouraging the agent to get closer. By comparing together only the abstract representation of the states, one can discard the specific details in the sensory information and focus rather

on the abstract features that determine the agent's environment. It is clear that for this application, a sensory representation that has isolated the independent factors of variation of the environment is particularly relevant. Abstract representation learning for goal definition is closely related to hierarchical Reinforcement Learning and goal-based Reinforcement Learning [62, 142], which are the subjects of Chapter 6.

The field of representation learning is also closely linked to that of *multimodal integration*. Multimodal integration designates the study of the coalescence of the information from the different sensory modalities into one, which essentially consists in learning a joint representation of the different sensory modalities [50, 88].

The rest of this chapter heavily draws on a publication in *IEEE Transactions On Cognitive And Developmental Systems* published in 2021 and titled *Learning Abstract Representations through Lossy Compression of Multi-Modal Signals* [148].

3.3 Methods

Our general approach comprises three processing steps. The first step consists in generating multimodal sensory data. In order to have more control over the amount of mutual information among the different sensory modalities, we present a way to generate the latter from noise. We call this type of data synthetic as it has no real significance. For this we use random multi-layer neural networks that map independent information sources x onto different "views" y seen by different sensory modalities. This models the process how multimodal sensory information is generated from unobserved causes in the world. We also validated our model using synthetic data on *real* data, obtained from a robot simulator. In the second step, we train a neural network autoencoder with varying capacity, i.e., size of the central bottleneck, to learn a compressed (lossy) representation z from the concatenation of the multimodal inputs y. This models the process of a developing agent learning an abstract representation from multimodal sensory information. In the third step we analyze the learned representation z and measure how much information it retains that is unique to individual modalities versus shared among multiple modalities. For this we train a third set of neural networks to reconstruct the original information sources x from the latent code z. The reconstruction error is a proxy for how much information has been lost during the encoding process. We now explain the three processing steps in detail.

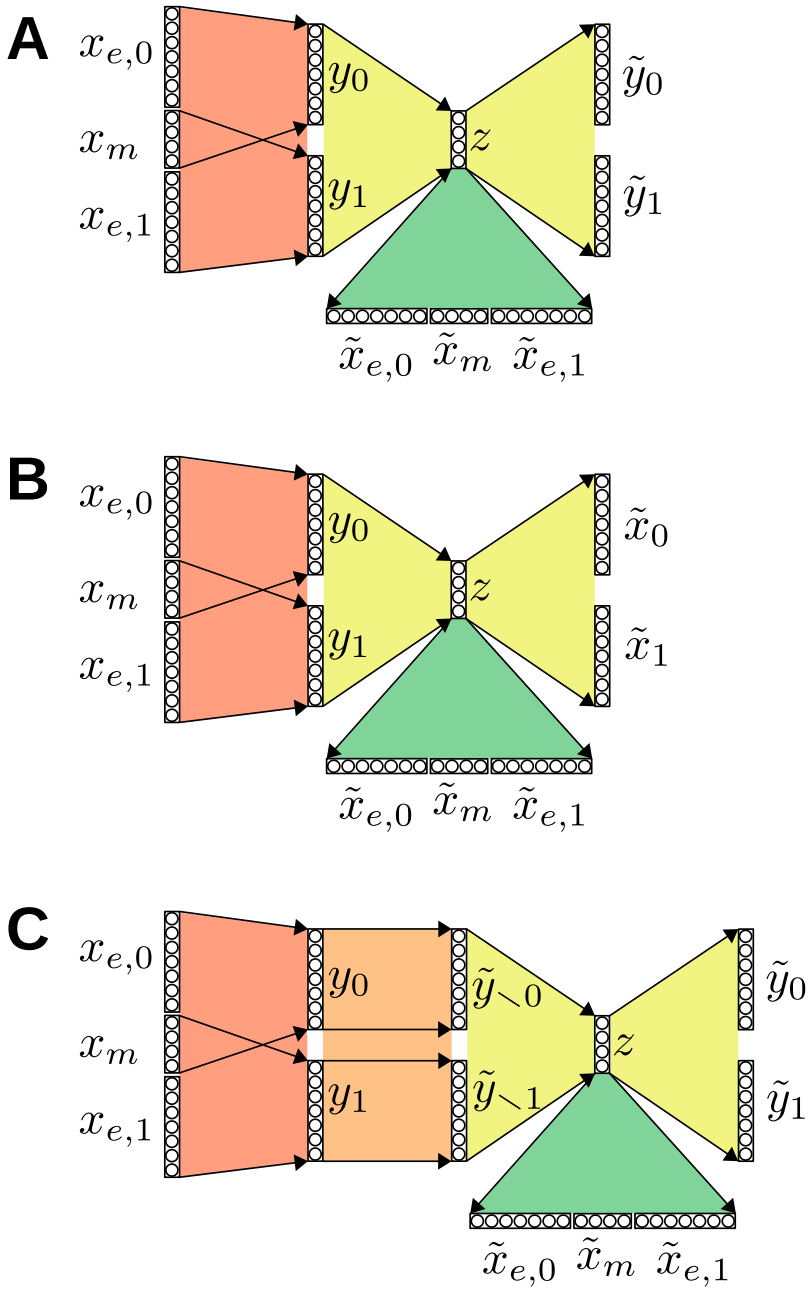


Fig. 3.1 Overview of the approaches, assuming only 2 modalities ($n = 2$). **A.** baseline experiment, jointly encoding the dependent vectors y_i. **B.** control experiment, jointly encoding the dependent vectors y_i but reconstructing the original data x_i. **C.** cross-modality prediction experiment, jointly encoding the predicted vectors $\tilde{y}_{\setminus i}$. In each schema, the red areas represent the random neural networks generating dependent vectors (section 3.3.1), the yellow areas represent the encoding and decoding networks (section 3.3.2), the green areas represent the readout networks (section 3.3.3), and the orange area represents the cross-modality prediction networks (section 3.3.4).

3.3.1 Step 1: Generating Synthetic Multimodal Input

We produce the multimodal sensory data in such a way that we can precisely control the amount of mutual information between the different sensory modalities. We first define a distribution p_m from which we sample information that is shared by all sensory modalities. p_m therefore represents independent information sources in the world that affect multiple sensory modalities. We define it as a d_m-dimensional distribution of independent standard normal distributions. We then define a second distribution p_e, from which the information exclusive to each modality is sampled. We define p_e as a d_e-dimensional distribution of independent standard normal Gaussians. For a shared vector $x_m \sim p_m$ and n vectors $x_{e,i} \sim p_e$, $i \in \mathbb{N}_{n-1}$ we can create the vectors $x_i = x_{e,i} \oplus x_m$ carrying the information of each modality, where $\oplus$ is the concatenation operation.

Our sensory modalities do not sample the underlying causes directly (e.g., objects and light sources), but indirectly (e.g., images provided by the eyes). To mimic such an indirect sampling of the world without making any strong assumptions, we generate the sensory inputs y that the learning agent perceives via random neural networks. Specifically, we define the input to modality i as:

$$y_i = \frac{C\left(x_i, \theta_{C,i}\right) - \mu_i}{\sigma_i}\,, \tag{3.3}$$

where C and $\theta_{C,i}$ are the input construction network and its weights for the modality i and μ_i and σ_i are constants calculated to normalize the components of y_i to zero mean and unit variance.

Tuning the amount of mutual information between the vectors y_i is done by changing the dimensionalities d_m and d_e of the vectors x_m and $x_{e,i}$, respectively. The amount of information preserved from the vectors x_i in the vectors y_i depends on the dimension d_y of the vectors y_i. We define d_y to be proportional to the dimension $d_x = d_m + d_e$:

$$d_y = k \times d_x\,, \tag{3.4}$$

where $k \gg 1$. This ensures that the sensory inputs y_i essentially retain all information from the sources x_m and $x_{e,i}$.

3.3.2 Step 2: Learning an Abstract Representation of the Synthetic Multimodal Input via Autoencoding

Taken together, the set of vectors $\{y_i\}_{i\in\mathbb{N}}$ carries once the information from each $x_{e,i}$ and n times the information from the mutual vector x_m. To show that a lossy-compression algorithm

achieves a better encoding when favoring the reconstruction of the repeated information, we train an autoencoder to jointly encode the set of the y_i. We therefore construct the concatenation $y = y_0 \oplus \cdots \oplus y_{n-1}$ to train the autoencoder:

$$z = E(y, \theta_E) \tag{3.5}$$

$$\tilde{y} = D(z, \theta_D) \ , \tag{3.6}$$

where E and θ_E are the encoding network and its weights and D and θ_D are the decoding network and its weights. Tuning the dimension d_z of the latent representation z enables us to control the amount of information lost in the encoding process. The training loss for the weights θ_E and θ_D is the mean squared error between the data and its reconstruction, averaged over the component dimension and summed over the batch dimension:

$$L_{E,D} = \sum_{\text{batch}} \frac{1}{n d_y} (y - \tilde{y})^2 \ . \tag{3.7}$$

$$\tag{3.8}$$

As a control condition, we also study a second encoding mechanism, where, instead of reconstructing the dependent vectors $y_0 \oplus \cdots \oplus y_{n-1} = y$, the decoder part reconstructs the original data $x_0 \oplus \cdots \oplus x_{n-1} = x$. The loss for training the encoder and decoder networks E and D from equation 3.7 then becomes:

$$L_{E,D} = \sum_{\text{batch}} \frac{1}{n d_x} (x - \tilde{x})^2 \ . \tag{3.9}$$

3.3.3 Step 3: Quantifying Independent and Shared Information in the Learned Latent Representation

Finally, in order to measure what information is preserved in the encoding z, we train readout neural networks to reconstruct the original data x_m and the vectors $x_{e,i}$:

$$\tilde{x}_m = R_m(z, \theta_m) \tag{3.10}$$

$$\tilde{x}_{e,i} = R_e(z, \theta_{R,i}) , \tag{3.11}$$

where R_m and θ_m are the mutual information readout network and its weights, and the R_e and $\theta_{R,i}$ are the *exclusive information* readout networks and their weights.

The losses for training the readout operations are the mean squared errors between the readout and the original data summed over the batch dimension and averaged over the component dimension:

$$L_m = \frac{1}{d_m} \sum_{\text{batch}} (\tilde{x}_m - x_m)^2 \text{ and} \tag{3.12}$$

$$L_{e,i} = \frac{1}{d_e} \sum_{\text{batch}} (\tilde{x}_{e,i} - x_{e,i})^2 \ . \tag{3.13}$$

Finally, once the readout networks trained, we measure the average *per data-point* mean squared errors

$$r_m = \frac{1}{d_m} \mathbb{E}\left[(\tilde{x}_m - x_m)^2\right] \text{ and} \tag{3.14}$$

$$r_e = \frac{1}{d_e} \mathbb{E}\left[(\tilde{x}_{e,i} - x_{e,i})^2\right] \ , \tag{3.15}$$

serving as a measure of the portion of the mutual and exclusive data retained in the encoding z.

3.3.4 An Alternative to Step 2: Isolating the Shared Information

We also compare the previous approach to an alternative architecture, designed specifically to isolate the information shared between the modalities. Let (A,B) be a pair of random variables defined over the space $\mathscr{A} \times \mathscr{B}$ with unknown joint distribution $P(A,B)$ and marginal distributions $P(A)$ and $P(B)$. Determining the mutual information between the variables A and B consists in finding either one of $P(A,B)$, $P(A|B)$ or $P(B|A)$. With no other assumptions, this process requires to sample many times from the joint distribution $P(A,B)$. We propose to make the strong assumption that the conditional probabilities are standard normal distributions with a fixed standard deviation σ

$$P(B = b|A = a) = \mathscr{N}(\mu(a), \sigma, b) \tag{3.16}$$

We can then try to approximate the function $\mu(a)$ with a neural network $M(a, \theta_M)$ maximizing the probability $P(B = b|A = a)$. $\mu(a)$ thus represents the most likely b associated with a, under the standard normal assumption. Training the network is done by minimizing the mean squared error loss

$$L_M = -\mathbb{E}_{a,b\sim P(A,B)}\left[\log\left(\mathcal{N}(\mu, \sigma, b)\right)\right] \tag{3.17}$$

$$= \mathbb{E}_{a,b\sim P(A,B)}\left[(\mu - b)^2\right] \cdot K_1 + K_2 \tag{3.18}$$

with K_1 and K_2 constants depending on σ.

More concretely and using the notation from the first architecture, we define for each modality i a neural network $M(y_i, \theta_{M_i})$ learning to predict all other modality vectors $y_j, j \neq i$. The loss for the weights θ_{M_i} is defined

$$L_{M_i} = \sum_{\text{batch}} \frac{1}{(n-1)\, d_y} \left(y_{\setminus i} - \tilde{y}_{\setminus i}\right)^2 \tag{3.19}$$

with

$$y_{\setminus i} = \bigoplus_{j\neq i} y_j \tag{3.20}$$

the concatenation of all vectors y_j for $j \neq i$ and $\tilde{y}_{\setminus i}$ the output of the network. We then consider the concatenation of the $\tilde{y}_{\setminus i}$ for all i as a high-dimensional code of the shared information. This code is then compressed using an autoencoder, similarly to the description in Section 3.3.2. We vary the dimension of the encoder's latent code. Finally, similarly to the first approach, we train readout networks from the compressed latent code to determine how mutual and exclusive information are preserved in the process.

Overall, this way of processing the data is analogous to the baseline experiment in that the cross modality prediction networks and the subsequent auto-encoder, when considered together, form a network that transforms the vectors y_i into themselves. Together, these two components can thus be considered as an auto-encoder, subject to a cross-modality prediction constraint.

3.3.5 Neural Network Training

In the following, we compare three architectures against each other (compare Fig. 3.1):

- The baseline architecture, simply auto-encoding the vectors y_i jointly (cf. Fig. 3.1A).

Network	M	C	E	D	R_m	R_e
Dimension	$(n-1)\times d_y$	d_y	d_z	$n\times d_y$	d_m	d_x

Table 3.1 Dimension of the last layer of each neural network

- The control condition with a simpler encoding task, where the vectors y_i are encoded into a latent code z, from which the decoder tries to reconstruct the original vectors x_i, from which the inputs y_i were generated (cf. Fig. 3.1B).
- The alternative architecture, where for each modality, a neural network tries to predict all other modalities and then all resulting predictions are jointly encoded, similarly to the baseline architecture (cf. Fig. 3.1C).

We will now describe the training procedure and implementation details. In order to show the nature of the information preferably preserved by the encoding process, we measure the quality of the readouts obtained as we vary the dimension of the latent vector d_z. To this end, for each dimension $d_z \in [1; d_{z,max}]$, we successively train the cross modality prediction networks (experiment C only), the autoencoder weights θ_E and θ_D and the readout weights θ_m and $\theta_{R,i}$. Once training is completed, we measure the average mean squared error of the readouts $\tilde{x}_m$ and $\tilde{x}_{e,i}$.

We choose the distributions of the vectors x_m and $x_{e,i}$ to be multivariate standard normal with a zero mean and unit variance. Therefore, a random guess would score an average mean squared error of 1. Each experiment is repeated 3 times and results are averaged.

The neural networks for the input construction C, cross-modality prediction M, encoding E, decoding D, mutual readout R_m, and exclusive readout R_e all have three fully-connected layers. The two first layers always have a fixed dimension of 200 and use a ReLU as non-linearity. The final layer is always linear, its dimension for each network is reported in Table 3.1.

For each model architecture A, B, or C, we show the effect of varying the ratio between mutual and exclusive data and that of varying the number of modalities. The default experiment used $d_e = 4$, $d_m = 4$, $n = 2$, $k = 10$. We then varied $d_e \in \{4, 10, 16, 22\}$ or $n \in \{2, 3, 4, 5\}$, keeping all other parameters fixed.

Each network is trained on 2500 batches of data of size 128 with a learning rate of 10^{-3} and using the Adam algorithm [55].

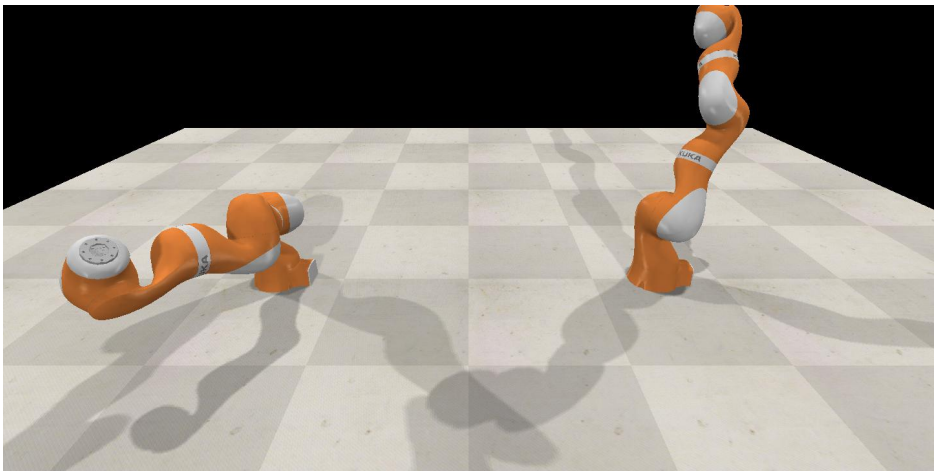


Fig. 3.2 High resolution image of the 2 arms in the simulated environment. The images in the dataset have a resolution of 32 by 64 pixels only.

3.3.6 Step 1: Generating Realistic Multimodal Input

In order to better assess our approach, we also tested it on realistic data generated from a robot simulator, in which we placed two 7 degrees of freedom robotic arms side by side (see figure 3.2). We then generated a dataset comprised of pictures of the 2 arms, representing the visual modality, and of the joints positions and speeds, representing the proprioceptive modality. Note how the position information is present both in the visual and proprioceptive modalities. The velocity information however is present only once, in the proprioception. So as to also having at our disposal an information stream that is uniquely present in the visual modality, we decided to provide the encoding networks solely with the proprioceptive information from one of the two arms (the right arm). Thus, the position information of the other arm (left arm) is only available through the visual information. Furthermore, by doing so, the velocity information about the left arm is present in neither of the two modalities and thus serves as a control factor. Finally, the dataset also contains records of the end effectors positions of the two arms. We consider the end effector position of the right arm as being implicitly part of the proprioceptive modality, as it is deducible from the position information, while not directly feeding it into the networks. A summary of the information available to each modality is provided in figure 3.3. In section 3.3.1, we named the vectors representing the different modalities y_i. When dealing with the realistic data, we will use $y_0 = y_v$ for the visual modality and $y_1 = y_p$ for the proprioceptive one. The y_p is z-scored, i.e. it has a 0 mean and a standard deviation of 1. The y_v vector is normalized such that the pixel values are in $[-1, 1]$.

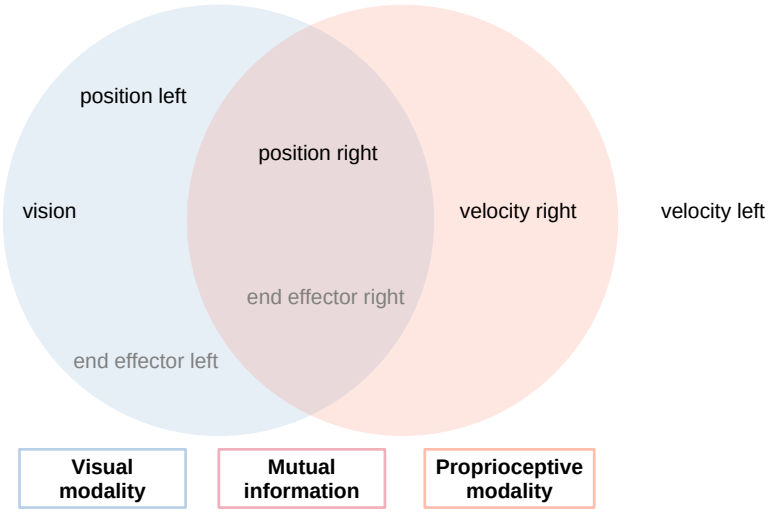


Fig. 3.3 Schema representing the information available to each modalities for the realistic data dataset. Note that the positions and velocities of the left arm are not part of the proprioceptive modality. This way, the information about the position of the left arm is available only through the vision sensor. It also results that the velocity information of the left arm is present in neither of the two modalities, and thus serves as a control factor in our experiments.

We will now redefine the steps 2, 3 and the alternative to step 2 for this dataset. The main difference with the synthetic dataset lies in the fact that the visual information must be processed with convolutional neural networks. Moreover, we propose to compare 2 ways of jointly encoding the modalities which we call *options*. The first option is analogous to the way the synthetic data is encoded, with the difference that the visual information is processed by a convolutional neural network. The second option consists in learning a latent representation of the visual information with an convolutional autoencoder prior to jointly encoding it with the proprioceptive information.

3.3.7 Step 2: Learning an Abstract Representation of the Realistic Data

Similarly to section 3.3.2, the y_v and y_p vectors are jointly encoded and decoded with an autoencoder $(E, \theta_E, D, \theta_D)$. This time however, the encoding and decoding steps are divided in two parts

$$z_{pre} = E_v\left(y_v, \theta_{E_v}\right) \oplus E_p\left(y_p, \theta_{E_p}\right) \tag{3.21}$$

$$z = E_{pre}\left(z_{pre}, \theta_{E_{pre}}\right) \tag{3.22}$$

for the encoder and

$$z_{post} = D_{post}\left(z, \theta_{D_{post}}\right) \tag{3.23}$$
$$\tilde{y}_v = D_v\left(z_{post,v}, \theta_{D_v}\right) \tag{3.24}$$
$$\tilde{y}_p = D_p\left(z_{post,p}, \theta_{D_p}\right) \tag{3.25}$$

for the decoder, where $z_{post,v}$ and $z_{post,p}$ form a partition of z_{post}. The index within z_{post} at which the split occurs is an hyper-parameter. The loss is then defined

$$L_{E,D} = \frac{1}{2d_p}\sum_{\text{batch}}\left(\tilde{y}_p - y_p\right)^2 + \frac{1}{2d_v}\sum_{\text{batch}}\left(\tilde{y}_v - y_v\right)^2 \tag{3.26}$$

with d_p and d_v the size of the proprioception and vision tensors.

Dividing the encoding and decoding process in two parts enables to use convolutional and deconvolutional networks E_v and D_v to encode and decode the visual information. We did not find any difference when pre-encoding the proprioceptive information with a MLP compared to directly feeding it to the E_{pre} network, we will therefore report the results for $E_p = \text{id}$ and $D_p = \text{id}$. In the case of option number 2, the y_v is a compressed representation of the visual information and there is no need to process it with a convolutional neural network. In that case, we also set $E_v = \text{id}$ and $D_v = \text{id}$, meaning that the architecture of the networks in that case is the same as for the synthetic dataset.

3.3.8 Step 3: Deciphering the Latent Code

Similarly to section 3.3.3, we train readout neural networks to decipher the information contained in the encoding z. This time however, since we don't have access to the original vectors x which induced the vectors y_v and y_p, the readout operation aims at reconstructing the proprioceptive information from both arms $y_{target} = y_{pos,l} \oplus y_{vel,l} \oplus y_{ee,l} \oplus y_{pos,r} \oplus y_{vel,r} \oplus y_{ee,r}$. The readout operation is written

$$y_{readout} = R\left(z, \theta_{readout}\right) \tag{3.27}$$

and its loss is

$$L_{readout} = \frac{1}{d_{target}}\sum_{\text{batch}}\left(y_{readout} - y_{target}\right)^2 \tag{3.28}$$

.

3.3.9 An Alternative to Step 2 for the Realistic Data: Isolating the Shared Information

Finally, similarly to section 3.3.4, we propose an alternative to step number 2 aiming at isolating only the information shared by both modalities. This is done by training two *cross-modality prediction* networks

$$\tilde{y}_{\setminus p} = M_v\left(y_p, \theta_{M_v}\right) \quad \text{and} \tag{3.29}$$

$$\tilde{y}_{\setminus v} = M_p\left(y_v, \theta_{M_p}\right) \tag{3.30}$$

with the losses

$$L_{M_v} = \sum_{\text{batch}} \frac{1}{d_v}\left(\tilde{y}_{\setminus p} - y_v\right)^2 \quad \text{and} \tag{3.31}$$

$$L_{M_p} = \sum_{\text{batch}} \frac{1}{d_p}\left(\tilde{y}_{\setminus v} - y_p\right)^2 \tag{3.32}$$

.

Again, like in section 3.3.7, the representations $\tilde{y}_{\setminus p}$ and $\tilde{y}_{\setminus v}$ are encoded using the autoencoder networks E_v, E_p, E_{pre}, D_{post}, D_v and D_p:

$$z_{pre} = E_v\left(\tilde{y}_{\setminus p}, \theta_{E_v}\right) \oplus E_p\left(\tilde{y}_{\setminus v}, \theta_{E_p}\right) \tag{3.33}$$

$$z = E_{pre}\left(z_{pre}, \theta_{E_{pre}}\right) \tag{3.34}$$

$$z_{post} = D_{post}\left(z, \theta_{D_{post}}\right) \tag{3.35}$$

$$\tilde{y}_v = D_v\left(z_{post,v}, \theta_{D_v}\right) \tag{3.36}$$

$$\tilde{y}_p = D_p\left(z_{post,p}, \theta_{D_p}\right) \tag{3.37}$$

.

For the option number 2, since the y_v is a one dimensional vector, we set $E_v = D_v = E_p = D_p = \text{id}$.

3.3.10 Description of the Networks

In the case of the option number 1, the network E_v is a convolutional neural network composed of 2 convolutional layers with kernel size 4 and stride 2 followed by a dense layer with output size 100. The network E_{pre} is a 3-layered MLP where all layer sizes but the last are 200. The last layer uses a linear activation function and has a size d_z. The network D_{post} is a 3-layered MLP where all layer sizes but the last are 200. The last layer uses a linear activation function and has a size $100 + d_p$. The network D_v is a deconvolutional neural network composed of a dense layer of size 8192 followed by two transposed convolutional layers with kernel sizes 4 and strides 2. Finally, the readout network is also a 3-layered MLP where all layer sizes but the last are 200. The last layer uses a linear activation function and has a size d_{target}.

For the cross-modality prediction, the network M_v is a deconvolutional neural network composed of a dense layer of size 8192 followed by two transposed convolutional layers with kernel size 4 and stride 2 and the network M_p is a convolutional neural network composed of two convolutional layers with kernel size 4 and stride 2 followed by a dense layer of size d_p.

Finally, as stated above, in the case of option number 2, y_v is a learned code of size 100 representing the visual information. In this case we set $E_v = D_v = E_p = D_p = \mathrm{id}$. The network learning the code is a convolutional autoencoder composed of 2 convolutions, one dense layer of size 100, one dense layer of size 8192 and 2 transposed convolutions.

3.4 Results

3.4.1 Lossy Compression of Multimodal Input Preferentially Encodes Information Shared Across Modalities

Figure 3.4 shows the reconstruction errors for exclusive vs. shared information as a function of d_z, the size of the autencoder's bottleneck, for the three different architectures. Each data point represents the mean of 3 repetitions of the experiments, and the shaded region around the curves indicate the standard deviation.

The grey dotted vertical line indicates the latent code dimension d_z matching the number of different univariate gaussian distributions used for generating the correlated vectors y_i, $d_{min} = d_m + nd_e$. Assuming that each dimension in the latent vector can encode the information in one normally distributed data source, when $d_z = d_{min}$ both the exclusive data and the shared data can theoretically be encoded with minimal information loss. Knowing that random guesses would score a reconstruction error of 1.0, we can augment the data with the theoretical values $r_m = 1$ and $r_e = 1$ for $d_z = 0$.

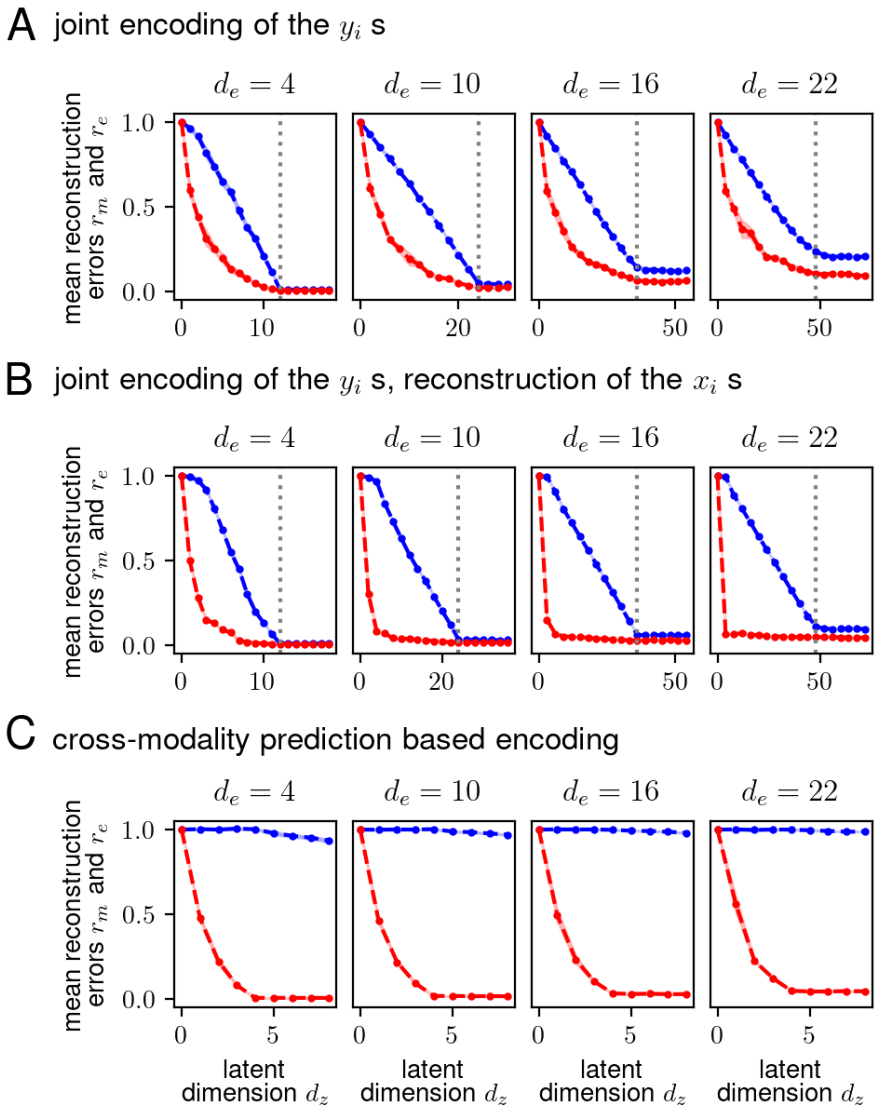


Fig. 3.4 Each plot represents the reconstruction error of the readout operation for the exclusive data r_e in blue, and for the shared data r_m in red, as a function of the auto-encoder latent dimension. The dotted vertical line indicates the latent dimension matching $nd_e + d_m$. The data point for a latent dimension of 0 is theoretically inferred to be equal to 1.0 (random guess). The four plots in one row correspond to different dimensions d_e of the exclusive data. The results are presented for the three architectures **A**, **B** and **C**.

The results for the baseline architecture (**A**), jointly encoding the correlated vectors y_i, show that the data shared by all modalities is consistently better reconstructed by the autoencoder for all latent code sizes d_z. In particular, this is also true for over-complete codes when $d_z > d_{min}$. Information loss in that regime is due to imperfections of the function approximator. When the code dimension is bellow d_{min}, the information loss is greater, as not all of the data can pass through the antoencoder's bottleneck. These results confirm our intuition from the Introduction (Sec. 3.1) that shared information should be preferentially encoded during lossy compression of multimodal inputs.

This information filtering is a consequence of neural networks' continuity, implying topological properties on the functions that they can learn. Indeed, while there exist non-continuous functions for which the dimensionality of the codomain is greater than that of the domain, the continuity property enforces that the dimension of the codomain is less or equal to that of the domain [19]. As a consequence, the dimensionality of the codomain of the decoder network of an autoencoder is less than or equal to the dimensionality of the latent code. If the dimension of the latent code is itself lower than that of the data, as can be enforced by a bottleneck, it follows that the data and its reconstruction sit on manifolds of different dimensionality, implying information loss.

In the *under-complete* regime, $d_z < d_{min}$, the autoencoder shows a stronger preference for retaining the shared data, partly filtering out the exclusive data. The chief reason for this is that the shared data is essentially counted n times in the network's reconstruction loss, while the exclusive data is counted only once. As the dimension of the exclusive data d_e increases, we still observe the two training regimes for d_z less or greater than d_{min}, even though the boundary between both tends to vanish as we reach the network's capacity.

The results for the second (control) architecture (**B**), jointly encoding the correlated vectors y_i by reconstructing the original data vectors x_i rather than y_i, are similar in nature to those of experiment **A**. The main differences occur at low values of d_z. The readout quality of the exclusive data is overall higher and that of the shared data lower.

Finally, results for the alternative architecture (**C**), encoding the cross-modality predictions, are significantly different and confirm that it is possible to isolate the mutual information between different data sources. Notice how for $d_e = 4$, the readout quality of the exclusive data r_e seems to improve slowly as the dimension d_z increases. We verified that the values of r_e remain high for high values of d_z, measuring reconstruction errors converging around 0.8. Thus, this architecture is more effective in stripping away any exclusive information. This is because, by definition, exclusive information cannot be encoded during the initial cross-modality prediction (Fig. 3.1C, orange part).

3.4.2 Increasing the Number of Modalities Promotes Retention of Shared Information

Figure 3.5 shows the results for varying the number of modalities. For architectures **A** and **B**, they show how increasing the number of modalities reinforces the retention of the shared data over the exclusive data. Note how the reconstruction errors for the shared information (red curves) decay more rapidly for higher numbers of modalities n. This is in contrast to architecture **C**, where results are very similar for different numbers of modalities. This is because the initial cross-modality prediction network (Fig. 3.1C, orange part) effectively removes all modality-specific information, leaving essentially the same encoding task for the subsequent autoencoder despite the different numbers of modalities n.

3.4.3 Results on the Realistic Data Dataset

Figure 3.6 shows the readout errors for the proprioceptive information from both arms, for the architecture jointly encoding the 2 modalities (**A**) and the one performing a cross-modality prediction (**C**), as a function of the size of the latent code. In both cases, the velocity information about the left arm, which is present in neither of the 2 modalities and thus serves as a control factor, is not recovered for any latent code size. This is indicated by a chance-level reconstruction quality of 1.0. For the joint-encoding architecture, the information which is present in both modalities (i.e. the right arm's joints positions and the right end-effector position) is well reconstructed. For the cross-modality architecture however, the right arm's joints positions are recovered with a MSE of around at best 0.2. The reason for this is that the position of some of the joints is not visible at all on the frames (like for example the last joint in the arm rotating the wrist) and in some positions, occlusion effects occur. The information about these joints is therefore not present in the mutual information and is thus filtered out by the cross-modality prediction. The joints velocities information has inherently a low entropy, making it easier to compress. As a result, the joint encoding approach shows a very good recovering of this information channel. The other approach however has properly filtered it out, even-though some of the information seems to have leaked out during the proprioception $\rightarrow$ vision cross-modality prediction. This is understandable given the big increase in dimensionality taking place in this operation. For the cross-modality prediction approach, the position of the left arm, which is present only in the visual modality, is properly filtered out with MSEs greater than 0.7 for all latent sizes. In the first approach however, only the left end-effector position is recovered for latent sizes greater than 14, which corresponds to the point where the proprioception of the right arm is fully represented in the code.

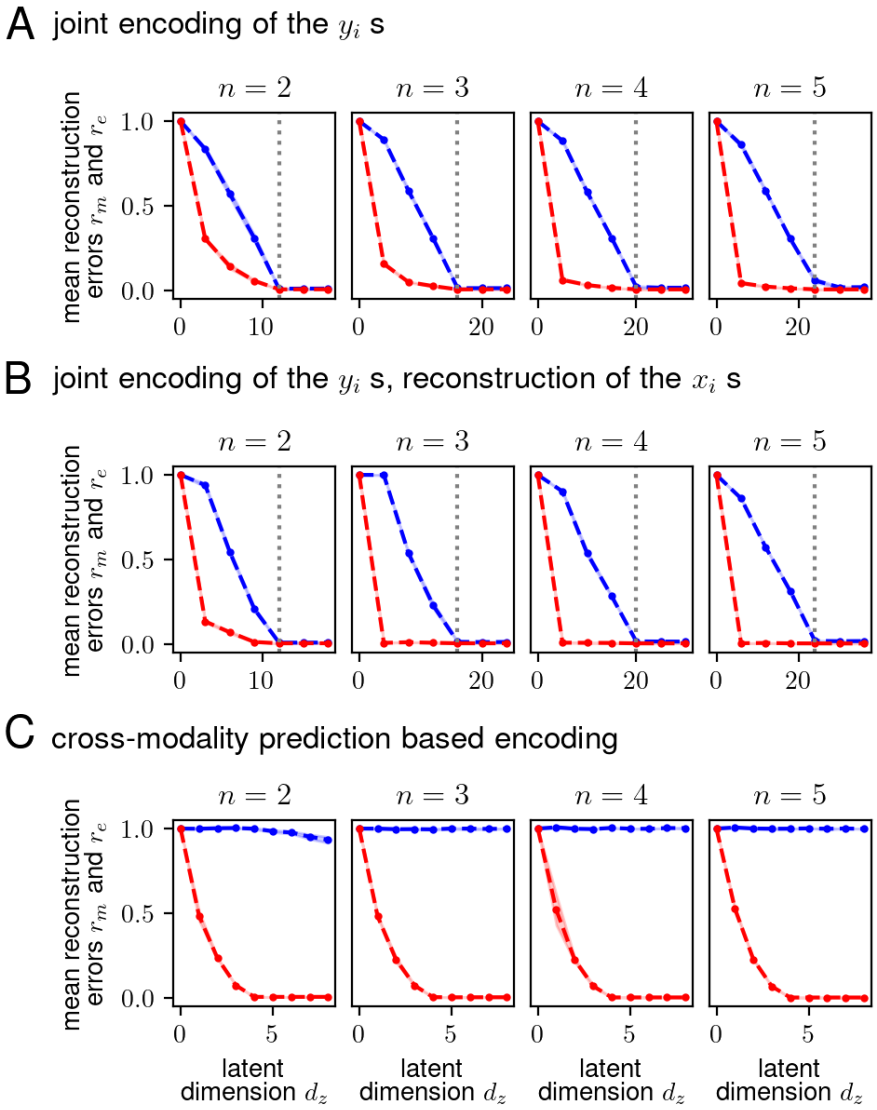


Fig. 3.5 Similarly to Fig. 3.4, each plot represents the reconstruction error of the readout operation for the exclusive data r_e in blue, and for the shared data r_m in red, as a function of the auto-encoder latent dimension. The four plots correspond to a different number n of modalities. The results are presented for the three architectures **A**, **B** and **C**.

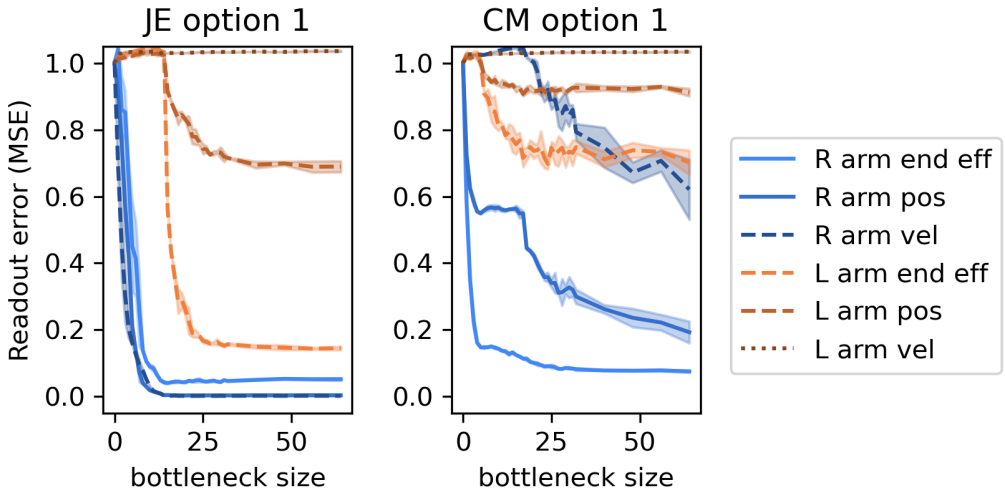


Fig. 3.6 Readout reconstruction errors for the joint-encoding and cross-modality approaches as a function of the size of the bottleneck of the encoding for the option number 1. Blue and red curves correspond to right and left arms respectively, solid lines correspond to information present in both modalities, dashed lines to information present in one modality only, and the dotted line to information present in none of the modalities.

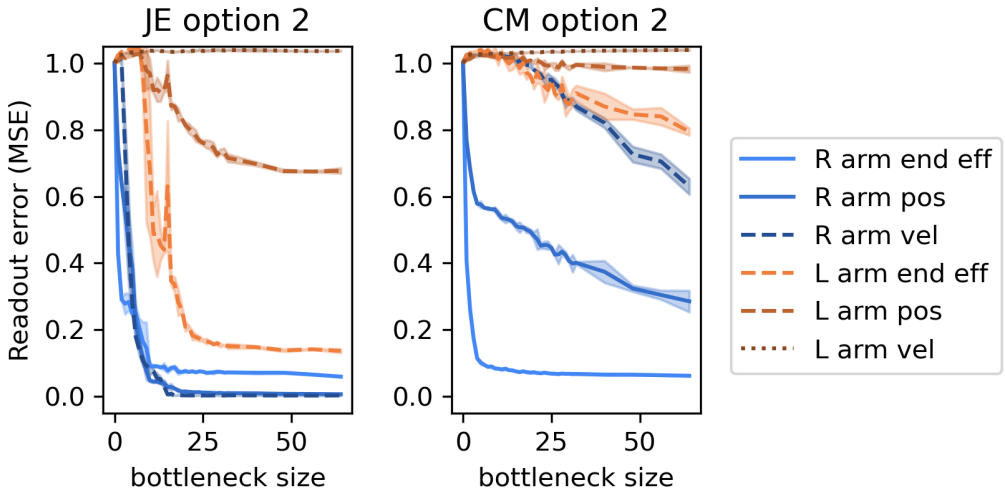


Fig. 3.7 Readout reconstruction errors for the joint-encoding and cross-modality approaches as a function of the size of the bottleneck of the encoding for the option number 2. Blue and red curves correspond to right and left arms respectively, solid lines correspond to information present in both modalities, dashed lines to information present in one modality only, and the dotted line to information present in none of the modalities.

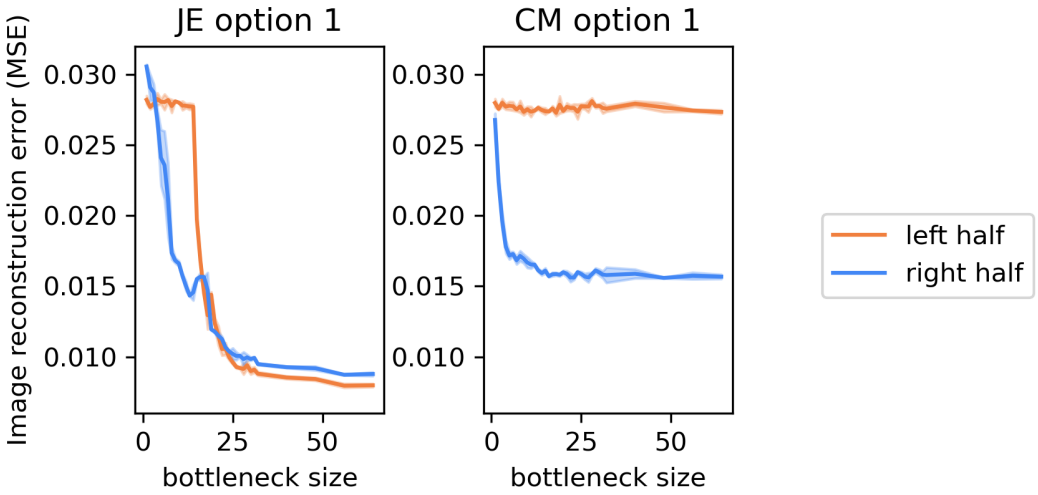


Fig. 3.8 Reconstruction error of the visual modality for the joint-encoding and cross-modality approaches as a function of the size of the bottleneck of the encoding for the option number 1. The error is split in two parts corresponding to the left and right halves of the frames. The results show that the pixels which share information with the proprioceptive modality are better reconstructed.

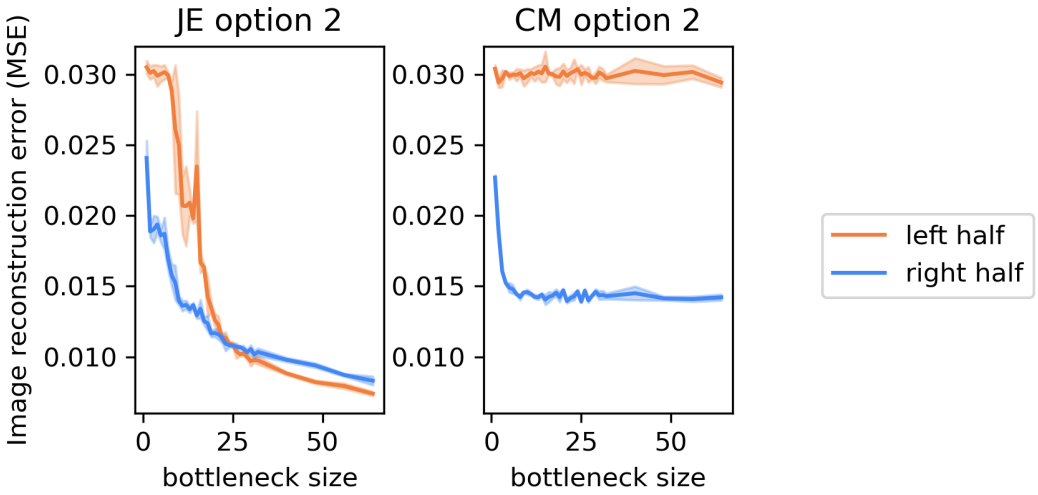


Fig. 3.9 Reconstruction error of the visual modality for the joint-encoding and cross-modality approaches as a function of the size of the bottleneck of the encoding for the option number 2. The error is split in two parts corresponding to the left and right halves of the frames. The results show that the pixels which share information with the proprioceptive modality are better reconstructed.

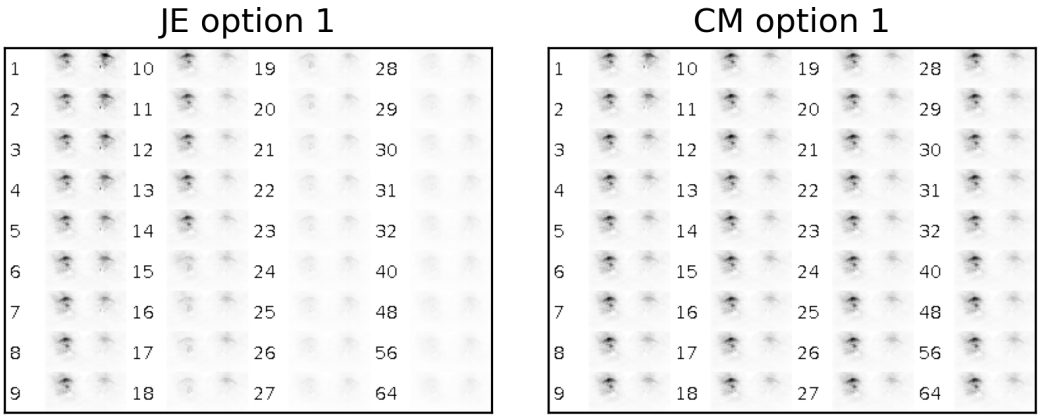


Fig. 3.10 Error map showing the mean reconstruction error of the visual modality for the joint-encoding and cross-modality approaches as a function of the size of the bottleneck of the encoding for the option number 1.

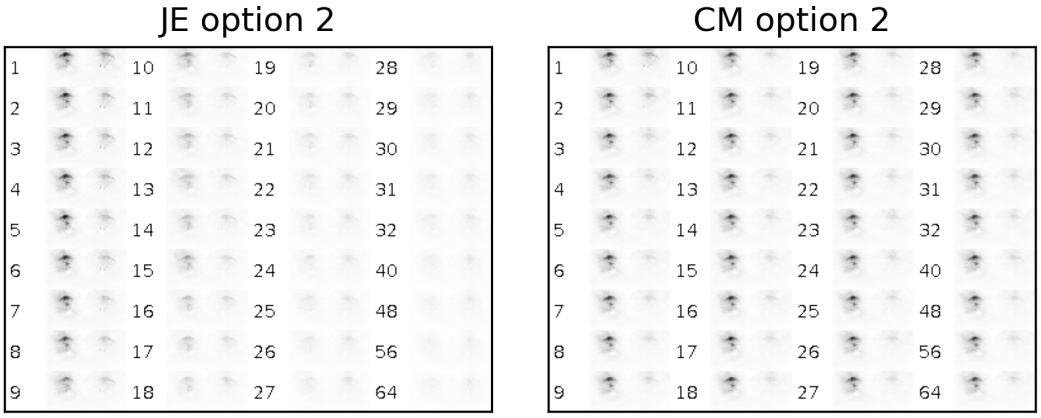


Fig. 3.11 Error map showing the mean reconstruction error of the visual modality for the joint-encoding and cross-modality approaches as a function of the size of the bottleneck of the encoding for the option number 2.

Figure 3.8 shows the reconstruction error of the visual information as a function of the latent code size for the joint-encoding (**A**) and cross-modality (**C**) approaches. The reconstruction error is split into two parts corresponding to the left and right half of the frames. Note that the chance reconstruction error is around 0.027 (MSE). The results show that in both approaches, the pixels corresponding to the right arm are better reconstructed. In the joint-encoding approach, when the latent code size allows it, the entirety of the frame is encoded while for the cross-modality approach, the left half of the frame is never encoded.

Finally, figure 3.10 shows a map of the visual reconstruction's MSE for each latent code size. The results clearly show that the joint-encoding approach (**A**) goes from a regime where neither the left nor the right part of the frame is reconstructed, to a regime where the right

part is, but not the left, to a regime where the whole frame is correctly reconstructed. For the cross-modality prediction (**C**), the system only goes through the 2 first regimes.

Figures 3.7, 3.9 and 3.11 show similar results for the option number 2. Note that for the figures 3.9 and 3.11, the frame reconstruction is $B\left(D_{post}\left(z, \theta_{D_{post}}\right), \theta_B\right)$ with B the decoder part of the autoencoder learning the latent representation y_v.

3.5 Conclusion

3.5.1 Learning Abstract Representations

In this Chapter, we have focused on an unsupervised approach to learning abstract representations of multimodal sensory data. Our analysis has been conducted on a synthetic dataset imitating multimodal sensory information, and validated on data coming from a simulated robotic setup. Our key result is that the lossy compression of multimodal sensory data naturally favors the retention of shared information.

We compared two architectures, yielding different results. The *joint encoding* approach shows the existence of two possible regimes in the training of autoencoders. In the non-saturated regime, almost all the information in the encoder's input is preserved in the representation. In this regime, autoencoders could be used to perform multimodal integration [88, 97, 127]. Next we showed that in the saturated regime, the information that is uniquely present in one modality tends to be discarded, thus forming a representation which is an abstraction of the real world. The latter is particularly relevant for defining abstract goals that an agent must pursue.

Although the joint-encoding approach favors the retention of the mutual information, some of the exclusive information is also present in the learned representation. Our second architecture, the *cross-modality* architecture, shows that it is possible to isolate precisely the mutual information from the exclusive information by training prediction networks. The mutual information is by definition statistically independent from the exclusive information, and we believe that it corresponds to an *independent factor of variation of the world*. Thus we think that this approach is interesting in order to construct a factored, abstract representation where concepts independent from each others are represented by independent variables.

3.5.2 Information Theoretic Perspective on Autoencoders

As discussed in sec. 2.2.4, in the last decades artificial neural networks have known a regain of interest within the scientific community. Today, Deep Neural Networks (DNN) are widely used in various applications and serve as the cement of many machine learning techniques.

Fundamentally, DNN are analogue to pipes conducting information which, like a gas, can be compressed or expanded. And like gases liquefy, when information is maximally compressed, it takes an incompressible form. This analogy is however not perfect: in the physical world, gases follow the law of conservation of mass, that is, no mass is neither created nor disappears. There is however no such thing as a *law of conservation of information*. In a closed system, information can be preserved like a gaz, but unlike gases it can also vanish into the void. Thus it is possible to squeeze an information flux beyond its incompressible form, but this implies that some information disappears. Our analysis reveals that the first information to disappear in traditional autoencoders is the information that is the least repeated in the data. Thus, all pieces of knowledge in an information flux are not equal: some are repeated more than others. Or in other words, the level of compression in an information flux is heterogeneous. In this work, we showed that autoencoders retain preferably the least compressed information.

3.5.3 Toward Intrinsically Motivated Reinforcement Learning

In the next Chapter, we will present Intrinsically Motivated Reinforcement Learning (IM-RL), as opposed to Extrinsically Motivated RL which we introduced in Chapter 2.

In short, in IM-RL rewards originate from the inner psyche of the agent. Jürgen Schmidhuber for example proposed to use a measure of the compression progress as a reward to drive an agent to explore parts of its environment where its mental code is improving [121].

As we will show in Chapters 4 and 5, our work about lossy compression is closely related to IM-RL. First, in Chapter 4 we will introduce the notion of Intrinsic Motivation (IM) and in Chapter 5 we will show how lossy compression can be used to generate an IM that can drive the acquisition of active vision.

Chapter 4

Intrinsically Motivated Reinforcement Learning

4.1 Introduction

In Chapter 2, we presented how a system - biological or artificial - can learn a behavior through the maximization of a measurement of its quality called reward. The rewards discussed so far are provided to the agent by the experimenter. For biological systems, external rewards correspond for example to resources found in the environment like food, or to external encouragements or punishments given by a teacher. Intuitively, rewards originating from the agent's environment are defined as extrinsic, as opposed to rewards that would be internal to the mind's working, called intrinsic.

It is important to note however that rewards themselves do not truly exist in the world. Our organism has only evolved to release neuromodulators (dopamine, serotonine, ocytocine, noradrenaline, endorphine...) as a function of its sensory experience. Ultimately, all rewards are internal. Therefore, what sense does it make to talk about intrinsic or extrinsic rewards? The concept of Intrinsic Motivation originates in the psychological literature in an attempt to explain why animals naturally engage in puzzles, or can be conditioned to have particular responses to neutral stimuli.

One way to categorize motivations is differentiating between those that are *close* to sensory experiences (i.e. a few variables only suffice to easily determine in what quantity the agent should be rewarded) and those that are more complex (i.e. the reward level is determined by a long history of sensory experiences and/or is function of abstract features present in the sensory experiences combining all sensory modalities). This schema (i.e. simple to complex rewards) sets a one-dimensional axis of reflection to try and classify

motivations. On one side of the spectrum, motivations are described as extrinsic (there is a simple link between experiences and reward), one the other side of the spectrum, motivations are described as intrinsic (significant internal computation is required to translate experiences in a reward signal).

In [6], Gianluca Baldassare proposes that they differ in that extrinsic motivations qualify the drive to maintain the homeostasis of needs detected within the visceral body (the integumentary, ingestive, excretory, circulatory, endocrine lymphatic and reproductive systems) such as for example the body temperature, energy level or thirst. Intrinsic Motivations on the other hand qualify the drive to improve skills and gain knowledge which is based on measurements from inside the brain. In a second publication [7], Baldassare proposes a more general definition: extrinsic motivations have the overall function of driving behavior and learning towards the acquisition of material resources, while Intrinsic Motivations are processes that drive the acquisition of knowledge and skills in the absence of extrinsic motivations.

Additionally to the distinction between intrinsic and extrinsic motivations, the literature offers more precise classifications of the Intrinsic Motivations.

In the following, we will present a typology of Intrinsic Motivations (IM) introduced by Jürgen Schmidhuber and a categorisation of IMs proposed by Gianluca Baldassarre. Then we will detail a few concrete implementations. In the next Chapter, we will introduce the Active Efficient Coding (AEC) IM and review existing work applying its principle. We will then verify that AEC's core idea is compatible with Deep Learning techniques, and finally we will present an application of AEC to the learning of active vision based on Deep Learning.

4.2 Classification of the Different Types of Intrinsic Motivations

Intrinsic Motivations have been studied both by the psychology branch of science, and the computational branch. In the latter, *Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990 - 2010)* [122] by Jürgen Schmidhuber proposes a typology of Intrinsic Motivations defined with respect to the description of 3 elementary components:

1. A predictor, compressor or model of the history of sensory inputs, internal states, Reinforcement Learning signals and actions.

2. A real-valued intrinsic reward indicative of the learning progress of (1).

3. A Reinforcement Learning based agent able to maximize the future expected reward.

Schmidhuber then proposes for each of these 3 components questions helping to further characterize the Intrinsic Motivation. (1) What can the predictor predict? The compressor compress? Is it deterministic or stochastic? Is the entire history of sensory experiences used to generate the real-valued reward, or only a portion? (2) Which measure is used to indicate the learning progress? Is the computation time of the predictor / compressor taken into account when measuring its performance? (3) How does the system deal with the problems of online learning and non-stationary reward?

A second classification of IM exists in the literature, distinguishing between 3 different classes of Intrinsic Motivations. In [94], the authors propose to differentiate between *knowledge-based* IMs and *competence-based* IMs. The first class covers all Intrinsic Motivations which encourage the acquisition of new information about the world. The second covers Intrinsic Motivations based on measurements of the quality of an already-learned behavior. In [10], the authors further refine the first class by distinguishing between *novelty-based* IMs and *prediction-based* IMs. In this section, we will try to define with more details the 3 classes of Intrinsic Motivations and give, when possible, examples of biological and computational implementations.

4.2.1 Novelty-Based IM

Novelty-based IMs (NB-IM) aim at improving an agent's knowledge of the world by encouraging it to discover new sensory experiences. This involves that the agent has a long term memory of its past experiences, such that it can estimate the level of novelty of the current observation. In [7], Baldassarre proposes an example of a hippocampal mechanism possibly implementing a NB-IM. The hippocampus is an important brain structure for memorization processes. It has been observed that simultaneous lesions to both left and right hippocampi cause profound difficulties in forming new memories, while also partly deteriorating memories already presents before the lesion. Measurements show that the hippocampus strongly responds to the perception of novel objects or novel arrangements of familiar items [64]. This activation in turn excites dopaminergic neurons of the ventral tegmental area, which sends dopamine to the hyppocampus itself, reinforcing the formation of memories. It is hypothesized that, through a dopamine delivery, the ventral tegmental area also drives the learning of a novelty seeking behavior.

There are many examples of novelty based IMs in the computational literature [12, 39, 21, 4]. [12] proposes to define the novelty reward based on a pseudo-count mechanism. Precisely counting how many times each state has been visited is often infeasible due to the dimensionality of the state space and to the fact it is often continuous rather than discrete. The authors propose to derive an approximation of the counter from a density model estimating

the probability of being in a state. They tested their approach on 5 Atari 2600 games acknowledged as hard, in the sense that an ε-greedy policy is inefficient at exploring them. One of these games in particular - *Montezuma's Revenge* - serves as a common IM benchmark. The reward used is $r = \frac{1}{\sqrt{N(x)+\varepsilon}}$ where $N(x)$ is the pseudo-count of times visiting the state x, thus rewarding the agent for visiting infrequent states.

[21] proposes a simple and innovative technique named *Random Network Distillation (RND)* to estimate the familiarity of an agent with a state. In this approach, a student neural network learns to approximate a random teacher network. The degree of proximity between their output informs us about the frequency at which the input state has been visited. Similarly to [12], the approach is tested on the Montezuma's Revenge Atari game. The authors show, that not only the agent learns to explore the entirety of the 24 rooms composing the game - which requires to unlock doors using keys that must be found in the environment - but it also develops interesting behaviors targeted towards enemies or items in the game.

In the 2 previous examples, the rewards are based on a life-long notion of novelty, meaning that a state is considered new if it has never been encountered since the beginning of training. [4] proposes an extension of [21] called Never Give Up (NGU) in which an additional term based on an episodic sense of novelty is added to the reward. This means that the agent is punished for visiting twice the same state within an episode - or rewarded for visiting a state that has not been visited yet within the episode. This new reward term is based on a memory buffer containing a representation of the last states observed by the agent during the current episode. The newly observed state is then compared with the k-nearest-neighbors already present in the buffer in order to derive its episodic novelty level. The publication investigates the benefit of the additional sense of episodic novelty over the life-long sense of novelty only.

Finally, inspiring from Maximum State-Visitation Entropy (MSVE), [39] proposes a new reward encouraging an agent to uniformly visit all states of an MDP. Moreover, building on the Geometry Aware Information Theory (GAIT) [32], this measurement of *uniformity* is defined with respect to a measure of similarity between the states (the *geometry* of the state space), discouraging the visitation of *common* states, ie states that are similar to many other states. The authors further propose a simple similarity metric based on a time adjacency principle: states that are close *in time* are considered similar. The approach is tested on simple discrete and continuous MDPs and the authors compare the results with the two previously presented IMs, Random Network Distillation (RND) [21] and Never Give Up (NGU) [4].

4.2.2 Prediction-Based IM

Prediction-based IMs describe the interest of an agent for surprises. It is defined with respect to the agent's ability to predict the future, given the past and the present. In [7], Baldassarre proposes an example of a tectal mechanism possibly implementing PB-IM in the brain [109]. In this publication, the authors show that in a few milliseconds, unexpected events like changes of the environment caused by an action activate the superior colliculus, which causes phasic dopamine bursts of the substantia nigra compacta which in turn reach the basal ganglia. They hypothesize that the dopamine signal reinforces the choice of the action that immediately precede an unpredicted event.

The first article presenting a prediction based IM dates back from 1991 [120]. In this publication, a controller network is trained to maximize the error of a predictior network at every timestep. As the author comments in [122], one drawback of this approach is that in highly stochastic environments, learning progress of the prediction model might be prevented instead of promoted, due to the fact that the agent is drawn to focus its search on parts of the environment where the prediction error is high. This flaw is commonly referred to as the *noisy-TV* problem. This name comes from the observation that a prediction-based curious agent like the one presented in [120] when placed in a maze explores its environment until it finds a source of stochasticity (the noisy-TV) whithin it. To address this issue, the author proposes in [118, 119] to focus on *the improvement* of the quality of the prediction model rather than on its quality itself. Indeed, if the prediction model can not be improved it means that either the prediction is already good or that it can not be learned. This describes well how infants are bored by things they already understand and master or by things that they can not apprehend.

More recently, [98] proposed an other model architecture that addresses the noisy-TV issue. The solution consists in learning a state representation z that preserves only the aspects of the environment that the agent can act on, leveraging the uncontrollable stochastic features of the state. This is done in a self supervised manner through the learning of an inverse model (a model learning which action brought the agent from state s to state s'). The prediction model then learns to predict the next state representation z', given the current state representation z and the selected action a. The authors show that with this technique, the agent is not attracted by stochasticity in the environment anymore. They demonstrate that a curious agent can learn to play the Nitendo game Super Mario Bross without any (extrinsic) reward coming from the game, and that it generalizes to previously unseen levels of the game.

During my PhD, I conducted a small project implementing a prediction-based IM like the one originally presented in [120] in a fully-deterministic, 2D, continuous world. In this

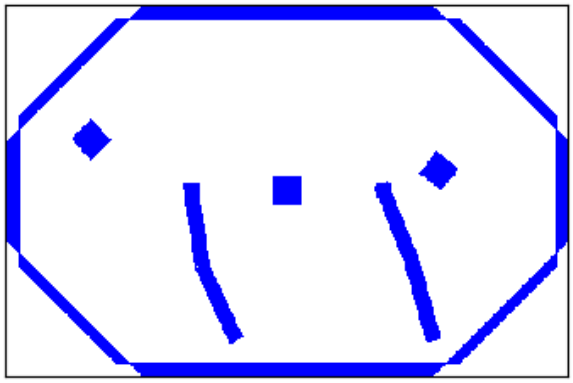
experiment, the agent is comprised of 2 arms with 2 joints each, placed in an arena along with 3 cubes that can be pushed (see figure 4.1). The particularity of this agent is that the 2 arms have a skin, enabling it to feel contacts with the cubes or with itself. It results that self-contacts are relatively easy to predict compared to contacts with the cubes. In line with our prediction, we observed that the agent learned to increase the frequency of unpredictable contacts with the cubes, however we found that it did not reduce the frequency of predictable contacts with itself. We also experimented with stochastic, as well as deterministic Reinforcement Learning algorithms, and found that the results were significantly affected. In the case of stochastic RL, the agent learned a high entropy policy consisting simply in fast stochastic arm movements that were harder to predict due to the movement inertia of the arms. The deterministic RL agent however suffered more from the non-stationary nature of the task which lead to unstable training. The deterministic policies mainly consisted in synchronous rotation of both arms in the same direction or quick repetitive motion of the arms. Figure 4.1 presents the results for the stochastic RL setup, and additionally details the effect of training the agent to maximize, minimize, or keep the prediction error around a target value.

The Active Efficient Coding (AEC) principle, subject of Chapter 5, can also be interpreted as a prediction-based IM. In AEC, instead of maximizing the prediction error, the agent tries to minimize the encoding error of its sensory experiences. The agent is thus incentivized to produce easily compressible sensory inputs.

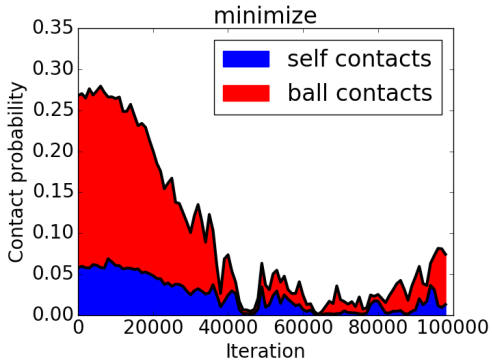
4.2.3 Competence-Based IM

Competence-based IMs are, as the name suggests, based on a measurement of the competence of the agent at solving a task or accomplishing a goal. It corresponds to an incentive to do things that we're good at or things that we can improve at, and serves the purpose of learning *what to learn and when*. Competence-based IMs are mostly relevant for solving tasks that can be decomposed in a hierarchy of sub-tasks. Indeed, often the acquisition of a sub-skill, like placing an object at a certain location, is conditioned by the acquisition of an other sub-skill, like grasping the object in the first place. Estimating the progress of the acquisition of such skills enables an agent to learn to select what competence to train when. As of this day, no biological examples of competence-based IMs are known and there are only few examples of implementations in the computational literature. This is partly due to the fact that they require the agent to be able to learn to solve multiple different tasks in parallel or in a hierarchy.

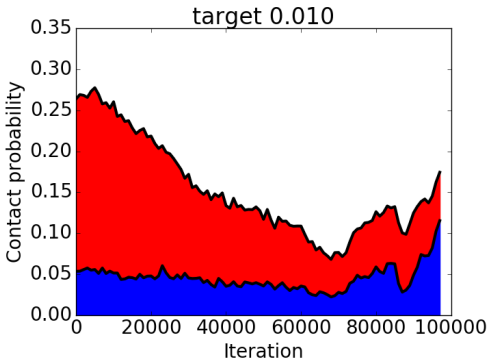
The very first occurrence of CB-IMs in the literature is proposed by Andrew Barto and Satinger Singh [11, 130] as a complement to the *option framework* [135]. The latter is a formalization of Hierarchical Reinforcement Learning (HRL), which is the subject of chapter



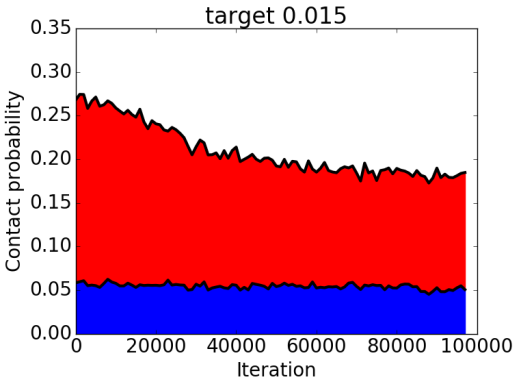
(a) Snapshot of the simple environment used to replicate a prediction-based Intrinsic Motivation similar to [120]. The two arms have a skin providing the agent with haptic feedback.



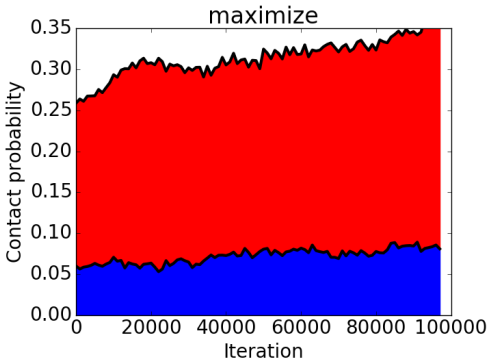
(b) Agent trying to minimize the prediction error. The frequency of contacts decreases.



(c) Agent trying to keep the prediction error around a value of 0.01. The frequency of contacts decreases but does not vanish.



(d) Agent trying to keep the prediction error around a value of 0.015. The frequency of contacts decreases but does not vanish.



(e) Agent trying to maximize the prediction error. The frequency of contacts increases.

Fig. 4.1 (a) Snapshot of the environment (b - e) Cumulative plot showing the per-iteration probability of a contact between the 2 arms and a cube (red) or between themselves (blue), as a function of training time. Four different training scenarios are presented: minimizing the prediction error, maximizing it, or keeping the prediction error around a target value (0.01 or 0.015).

6. In short, HRL is a branch of RL which aims at learning to achieve more complex tasks by decomposing them in multiple simpler tasks that are orchestrated by a controller. In [11], for each sub-task in the hierarchy, a model tries to predict the outcome of performing the associated *options* (sub-skill). The controller then chooses among the available *options* and his choice is rewarded for big prediction errors, similarly to a PB-IM. The reason why this IM is classified as a CB-IM rather than a PB-IM is because in the present case, the quality of the prediction also reflects the quality of the agent's competence at *doing*. Indeed, the prediction is much more accurate if the agent has learned to reliably solve the sub-task, in which case the terminal state after completion is consistently one of the states that trigger the option's termination. PB-IMs on the other hand, do not measure the competence of the agent at doing, but the competence at *predicting* or at *improving the prediction*.

Baldassarre proposes TD-CB-IM [8], a competence-based reward mechanism designed to train a selector network for a hierarchy of actors. In this research, 3 *experts* learn to accomplish elementary tasks and the selector decides at every iteration which of the experts to call. The experiment is then divided in 2 phases. In the *childhood* phase, the experts each receive an extrinsic reward and the selector gets the TD-CB Intrinsic Motivation. Then, during *adulthood*, the experts stop learning and are only exploited by the selector, which now receives an extrinsic reward. During childhood, TD-CB-IM rewards the selector for picking an expert which produced a high TD-error. The latter indicates, when positive, that the expert performed better than anticipated, and thus reflects the rate of improvement of the expert's competence. Interestingly, at the end of childhood, each expert is repeatedly chosen by the selector until the expert completes its task. This indicates that TD-CB-IM carries relevant information about the experts' skills.

Finally, in *Which is the best Intrinsic Motivation signal for learning multiple skills?* [114], Baldassarre presents a comparison of various prediction based and competence based IMs applied to a 2D robotic arm learning to accomplish multiple tasks. In this setup, a selector agent chooses between 8 experts, each learning to solve one task. 4 out of the 8 possible tasks are not solvable, and the selector must avoid to pick the associated experts. The latter is trained to maximize an intrinsic reward, and 15 possible IMs are compared. The results seem to indicate that the best IM consists in the prediction error of the achivement of the task, as opposed to the error in predicting the next state only.

4.2.4 Other IMs

More non-traditional Intrinsic Motivations that do not enter the NB / PB / CB categories can be found in the literature. The notion of reachability for example [115], introduced in 2018, shares attributes with all categories. In this approach, an agent tries to predict, given 2 states

s_A and s_B, the probability that s_B can be reached in under k steps when starting from state s_A. The agent then compares the state in which it finds itself with a short (episodic) history of previously encountered states. A high intrinsic reward is produced if the current state is not considered *reachable* from any states in the memory. This type of Intrinsic Motivation is relatable to NB-IM in the sense that a non-reachable state (according to the reachability prediction network) is either a completely new state, or a previously visited state for which a shortcut (a new, shorter path) has been discovered. It is also relatable to PB-IM in the sense that, given the history of visited states, the agent predicts if the current state was reachable. It is then rewarded, not for making an error in the prediction like PB-IM implies, but for reaching the current state faster than predicted.

An other interesting IM that do not entirely fit the categorization is called *empowerment* [58]. It is based on an information-theoretic formalism in which actions and future sensations are considered as an information transmission channel in the sense of Shanon. In information theory, the capacity of a transmission channel is defined as $C = \sup_{p(x)} I(X,Y)$ where $I(X,Y)$ is the mutual information between X and Y and $p(x)$ the probability distribution of X. In [58], the author proposes to define the empowerment of an agent as the capacity of the channel between a sequence of actions and the resulting state after performing the actions. It corresponds to the amount of information that the agent can *inject into* its sensors through its actions, or, in other words, to the potential influence that the agent has on its environment. The n-step empowerment is then defined $\mathfrak{E} = \max_{p(a_t,a_{t+1},\ldots,a_{t+n-1})} I(A_{t;t+n-1}, S_{t+n})$ where $A_{t;t+n-1}$ is a random variable representing a sequence of n actions and $p(a_t, a_{t+1}, \ldots, a_{t+n-1})$ its probability distribution. It results that the minimum n-step empowerment is reached if, independently of the n actions $a_t, \ldots, a_{t+n-1}$ the resulting state is always the same state s_{t+n} and the maximum when each different sequence of actions results in a different state s_{t+n}. From the principle of empowerment, arise multiple meaningful behaviors. If we consider that the agent can die, the state of being dead has by definition a single subsequent state and thus has a very low empowerment. It follows that an agent trying to maximize its empowerment will naturally learn to avoid death. If you now suppose that the agent has no goal or task to complete, but knows that one might arise in the future, by maximizing its empowerment, the agent put itself in a situation where it has a high number of possible future options. Empowerment can be understood as some measure of *preparedness*. In the special case of a 2 players game, by maximizing its empowerment and minimizing the other player's, an agent could learn to play games like go or chess. Indeed, at chess, maximising the number of squares that your pieces can move to while also minimizing your opponent's space is a good strategy. Moreover, keeping your pieces alive increases empowerment and the game stops when the other player does not have any moves left to play due to being checkmate. In

[52], the authors show how an agent controlling a double pendulum in order to maximize empowerment learns to reach the unstable equilibrium (i.e. the inverted vertical position). In [85], the authors demonstrate a few meaningful behavior that naturally arise from the empowerment principle. For example, an agent placed in a grid-world maze with spreading deadly lava learns to hide and to build walls to isolate itself. An other example shows how an empowered agent placed in an environment made of 2 rooms separated by a locked door learns to first pick-up the key that opens the door before placing itself at the location in the environment from which it has the most other states available, i.e. next to the door and in the biggest of the 2 rooms. Finally, they implemented a predator-prey scenario where the agent is being followed by the predator and show that, by maximizing its empowerment, it learns to dodge the predator.

4.3 Conclusion

We presented a distinction between extrinsic and intrinsic rewards. We then used a existing classification of Intrinsic Motivations to propose an overview of the research in that field. We also provided, when possible, analogies between the computational models and biology. As a summary, figure 4.2 shows a comparison of a few of the IMs that we detailed above. For each, the essential components are depicted and the reward equation is given.

As pointed out in [20], where multiple IMs are compared on different gaming environments,

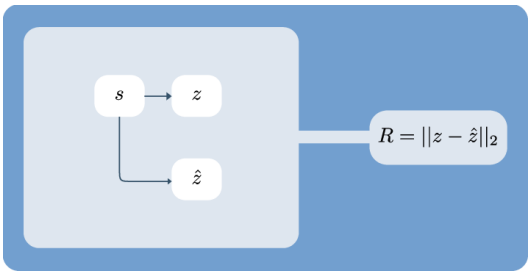
> *[The] results show surprisingly good performance, and a high degree of alignment between the intrinsic curiosity objective and the hand-designed extrinsic rewards of many game environments.*

This is evidence that there are general concepts, like curiosity, surprise, fun, empowerment, that are helpful for solving a multitude of tasks. Moreover, the neuro-scientific literature seems to corroborate this hypothesis as clues have been found, indicating that novelty-based and/or prediction-based IMs might be implemented in the brain, notably by the hippocampus, basal ganglia and superior colliculus.

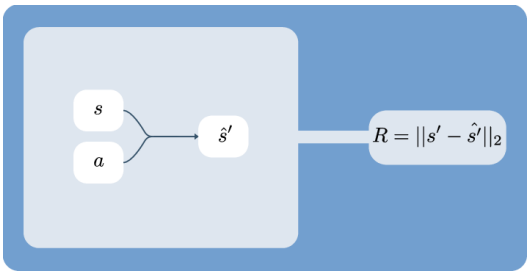
It seems clear that, in order to better understand how learning beings adapt to their environment, we must understand the reward mechanisms involved in that process. Although reward delivery is triggered by external sensory experiences, the rewards originate from the inside of the brain or the body. Thus, understanding how the brain and body convert sensory inputs into reward signals is key to comprehend the emergence of complex behaviors. The computational models that we presented in this section are inspired by biological systems,

however it is true that they remain biologically implausible. Nevertheless, they offer a way to study how biological systems might transform the sensory experiences into rewards.

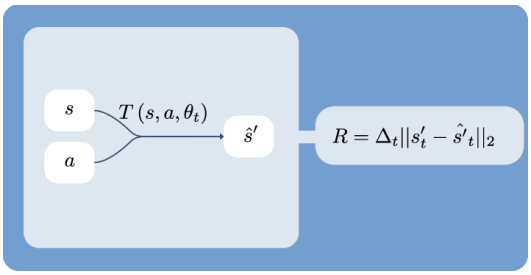
In the next Chapter we will focus on an IM called Active efficient Coding (AEC) and present its first successful implementation using deep learning methods.



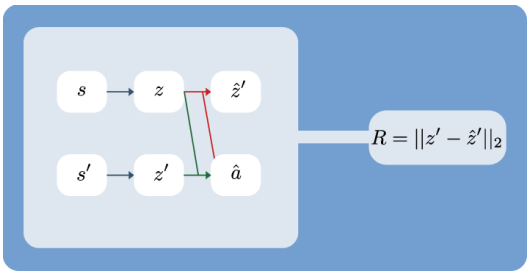
(a) In Random Network Distillation [21], a fixed, random teacher network transforms the sensory signal into a latent representation. A second network is trained to reproduce the teacher network and the error between both representations is used as a reward.



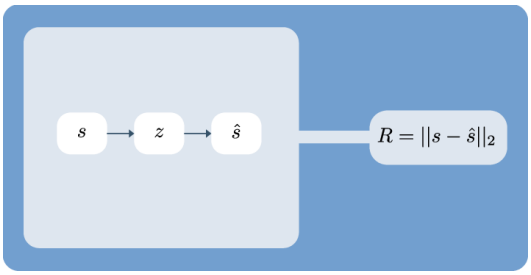
(b) In a typical prediction-based IM [120, 122], a model predicts the state resulting from taking action a while in state s and the error between the prediction and the true next state is used as a reward.



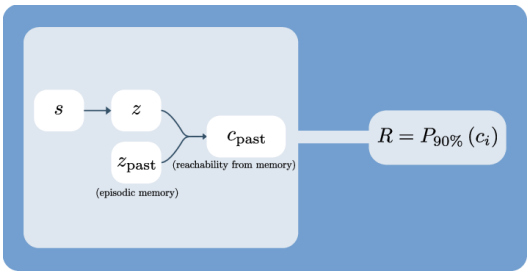
(c) Here $e = ||s'_t - \hat{s}'_t||_2$ is the error of the prediction model T. The agent is rewarded for visiting states where the prediction model can be improved, thus alleviating the *noisy-TV* problem. The improvement of the prediction model is measured as the difference between e before and after a weight update [118, 122].



(d) A more advanced prediction-based IM [98] proposes to learn a sensory representation z that contains only information about the controllable features of the environment. This is done via training an inverse model associating a pair of consecutive states to the action executed while in the first. The error of a prediction model operating in the learned action-dependent space serves as a reward for the agent.



(e) In Active Efficient Coding [147], the sensory input is encoded and decoded and the negative reconstruction error is used as a reward.



(f) A model learns to predict how many iterations c_i separate 2 different states [115]. The agent is then rewarded when it is evaluated as being 'far' from the recent history of states.

Fig. 4.2 Schematic overview of different Intrinsic Motivations. The states and actions are denoted s and a. The variable z is used to refer to latent representations. The x' notation indicates the next x (at time $t+1$) and the $\hat{x}$ notation that the variable is a learned approximation of the ground truth x.

Chapter 5

Active Efficient Coding

5.1 The Efficient Coding Hypothesis

Compared to the other organs, the brain has a relatively high energy demand. It represents about 20% of the total energy budget of the body [24]. However, this corresponds to an energy consumption of about 20 watts in average, making the brain very efficient energetically compared to computers. Thus, the processing and transmission of information in biological systems is both expensive compared to the energy available, and efficient compared to artificial systems. This suggests that, throughout evolution, the brain has been optimized to save energetic resources. To do so, it must have discovered a way of efficiently transmitting and processing information. Influenced by Claude Shannon's information theory, the neuroscientist Horas Barlow proposed in 1961 a hypothesis stating that the transmitted information in the brain is encoded prior to be sent in order to minimize the number of action potentials required for the transmission [9]. This hypothesis is called *Efficient Coding*.

Since 1961, further evidences have been found in favor of Efficient Coding. [92] for example shows that filters optimized for sparsely coding natural images resemble the receptive fields of simple-cells found in the visual cortex V1. The same observation has been made for the auditory domain, where networks optimized to encode natural sounds resemble the impulse response of cochlear filters in the inner-ear [70].

5.2 Efficiently Encodable Behavior

Multiple scientific evidences show that the brain uses an internal code to both extract knowledge from the sensory experiences and minimize the amount of energy needed to transmit information. This code thus depends on the subject's sensory experiences, which

themselves depend on the environment and on the behavior of the subject within it. This naturally raises the question, can the subject's behavior be optimized so as to produce easily encodable, low energy cost sensory experiences?

This idea forms the basis of an extension of the Efficient Coding hypothesis, called Active Efficient Coding (AEC). Starting in 2012, researches have been conducted to implement the principle of AEC as an Intrinsic Motivation that would shape the emergence of active vision. Active vision refers to the ability to move the eyes in order to better focus the attention on the surroundings. It is a cornerstone in the early stages of brain development, facilitating the learning of other abilities. It includes the learning of vergence and cyclovergence fixations, enabling stereopsis, and the learning of pursuit of objects in 3 dimensions.

The first paper about AEC [157] presents how this principle can drive the learning of vergence control only. Intuitively, the visual information reaching the eyes is the easiest to encode when the images on the two retinas are the same. In that case the encoding of one of the two images is sufficient to recover both. On the contrary, if the images are completely different, both must be encoded resulting in a bigger message to be transmitted. This information theoretic principle is the subject of the previous Chapter 3, *Learning Abstract Representations through Lossy Compression of Multi-Modal Signals*. The authors of [157] propose to use the Matching Pursuit (MP) algorithm [78] to encode two images jointly while sampling the disparity between the two images (i.e. the horizontal shift in pixels) from a known distribution with 0 mean. The results of this experiment show that for all disparity distributions, the binocular images with 0 pixels disparity are the best reconstructed and the pairs with the biggest absolute disparities are the less well reconstructed. Moreover, the results show that this effect is more pronounced if the disparity distribution has a low standard deviation. The authors then propose to use the reconstruction quality as the intrinsic reward of a Reinforcement Learning algorithm, associating the learnt visual code to a vergence command corresponding to a relative shift between the two images. This experiment is performed both given a fixed, already learnt, binocular image encoding and in an online fashion, where the code is learnt at the same time as the vergence controller. In both cases, the agent learns to cancel out the initial vergence disparity from which it is initialized at the beginning of each episode. Note how the AEC-IM is similar in nature to the prediction based IMs [118–120] quoted above. In the present case, instead of predicting the future sensory input given the current one, the agent only encodes and decodes the current visual information.

The encoding mechanism, implemented by the Matching Pursuit algorithm, jointly encodes left/right patches of size 10×10 pixels taken from the left/right input images. Technically, this imposes that the Reinforcement Learning agent can not *see* vergence

disparities bigger that 10 pixels. If we assume that the images are taken from standard low-resolution cameras, one pixel covers an angle of about $0.3°$ in the field of view and therefor the agent can not resolve disparities bigger than $0.3 \times 10 = 3.0°$. To overcome this limitation, and inspired by the human visual system, [73] and [74] propose to encode the visual information at 2 different spatial scales. This is done by training 2 matching pursuit algorithms in parallel, each encoding the input binocular image at a different resolution. The two scales are referred to as the fine and coarse scales. While the fine scale still encodes pixels which cover an angle of $0.3°$, the coarse scale now encodes pixels covering $2.4°$. This enables the agent to perceive vergence disparities of up to $2.4 \times 10 = 24.0°$. This second approach has been ported on a real robotic head and the authors report the resilience of this method to external perturbations in order to demonstrate its robustness.

An other research [56] applies the AEC principle to the learning of vergence on a bio-mechanical model of the eyes simulating the eye muscles. The vergence motion of the human eyes is controlled by two antagonist muscles for each eye. It results that a given eye position can be accomplished by many muscle contraction intensities. The authors show that while this redundancy does not alter the performance of the AEC principle, by adding a penalty term to the reward discouraging high metabolic demands from the muscles it is possible to learn to perform vergence control using the minimum energy. In a second paper [57], the authors use this model to study the development of active binocular vision under alternate rearing conditions. The approach models how impaired vision induces a change in the distribution of disparity tuned cells in the visual cortex and how it affects the learning of vergence control.

Next, [162] integrates cyclovergence, an additional type of eye movement, to the controller and shows that the AEC principle generalizes well. Cyclovergence consists in inwards and outwards rotations of both eyes around the line of view. Cyclovergence motions are particularly relevant for accurately fixating close objects situated above or bellow the horizontal plane.

Building on these researches, [156, 143, 137, 68] show how the AEC principle can also guide the learning of object tracking in time. This is done by encoding sequences of binocular pairs instead of a single pair, thus encouraging the agent to behave so as to produce temporally and binocularly constant images. In [155], the AEC model has been used to model a reflex called *optokinetic nystagmus*. It is the response of the visual system to a constant relative motion and is composed of two phases. The first phase consists in a simple tracking behavior in the direction of movement of the stimulus, and the second phase is a rapid saccadic movement in the opposite direction. Optokinetic nystagmus is often observed when looking sideways out of a moving vehicle.

Further extending the model, [160] proposes an attention mechanism called LBAIM based on information maximization interspersing object fixation phases with saccadic movements. The authors show that when augmented with this attention mechanism, the agent performs saccades to regions of the scene with a high relative binocular disparity so as to maximize the self-information of the perceived visual observation. Moreover, the system trained with attention shows a quicker learning compared to agents trained when performing random saccades or when performing saccades based on other attentional heuristics. In a follow-up paper [159], the authors propose a clever strategy to learn the attention model instead of relying on the LBAIM heuristic. It is based on an Intrinsic Motivation antagonist to the AEC, aiming at maximizing the reconstruction error after the saccade. Again, the learning speed of the agent using the LBAIM, the learned attention and random saccades are compared and the results show that both LBAIM and the learned attention yield faster learning. Finally, in [161] the authors combine this learned attention model with the learning of a cyclovergence controller similarly to [162].

In the AEC models presented so far, the agent's eye movements are rewarded for minimizing the loss of information taking place in the encoding process. This formulation lets the possibility to the agent to minimize the entropy of the visual sensory information, for example by closing its eyes or looking at a uniform color. This issue is referred to in the free energy and predictive processing literature as the "dark room problem". [29] proposes a new definition of AEC based on the maximization of the mutual information $MI(S,C) = H(S) - H(S|C)$ between the sensory information and its encoding. In this approach, a vergence control system is rewarded for minimizing the term $H(S|C)$ while an accommodation control system learns to maximize the term $H(S)$. This results in successful joint learning of both vergence and accommodation. Combined with a model of interocular suppression, the resulting model can be used to study the development of amblyopia, a disorder of the visual system that is characterized by an interocular difference in visual acuity. The results show that the recovery from an amblyopia-like state is possible if the receptive fields in the model remain plastic.

In this Chapter, we will present a modern implementation of the AEC principle that

1. replaces the Matching Pursuit algorithm by Deep Auto-Encoders for jointly learning a sensory representation and generating an intrinsic reward signal, and

2. combines the work in [156, 143, 137, 68] and in [162] to achieve the joint learning of vergence fixation, cyclovergence fixation and smooth pursuit (i.e. object tracking).

First, we will conduct a study similar to the one in [157] presented above, showing how the agent's input statistics affect the quality of the reconstruction of a Deep Auto-Encoder. Additionally, much like [73, 74], we will show the effect of using multiple spatial scales.

Finally we will exploit these results to generate an intrinsic reward signal that will drive the acquisition of the vergence, cyclovergence and tracking skills.

5.3 Efficient Auto-Encoding of Binocular Pairs with Controlled Error Statistics

5.3.1 Introduction

We presented the Efficient Coding hypothesis and its extension Active Efficient Coding and reviewed existing computational models of AEC. The latter are all based on the Matching Pursuit algorithm [78] or on the GASSOM algorithm [22] which have been used as models of the response of cortical neurons. In the field of developmental robotics however, architectures are more often based on deep networks.

Therefore, in the current section we will try to answer the following questions:

1. Can we use a Deep Auto-Encoder based algorithm instead of the Matching Pursuit algorithm in order to generate an Active Efficient Coding Intrinsic Motivation?

2. Can the Deep Auto-Encoder based IM drive simultaneously the acquisition of vergence, cyclovergence and smooth pursuit (i.e. tracking)?

3. What is the effect of using one vs. multiple spatial scales?

Before presenting our approach, we will briefly introduce the parvo- and magno-cellular pathways of the visual system.

5.3.2 Parvocellular vs. Magnocellular Pathways

The visual system is one of the oldest component of the human brain. It is believed that the emerging of the eyes precedes that of the brain in the phylogenetic tree of life [33]. Vision is deeply anchored in the human's neural system. Understanding the development of active vision in infants necessitates to study the inner working of the neural visual stream.

Initially, photons are transformed into electrical signals at the retina of each eyes. The information then travels along the optic nerves, which cross at the optic chiasma. There, the left eye / right eye separation of the information changes to a left visual scene part / right visual scene part separation, as half of the optic fibers from each eye do not cross the body mid-line (see Fig. 5.1). From there, the information is projected onto the Lateral Geniculate Nucleus (LGN). In the LGN, we will differentiate in particular two types of cells. The firsts

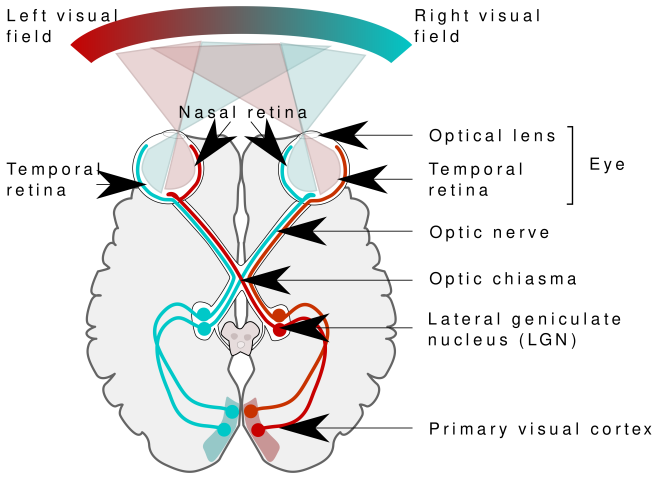


Fig. 5.1 Human visual system from the retina to the visual cortex V1. Information from similar regions of the visual scene is joint in the optic chiasma and tagged with velocity information in the LGN before it projects onto the visual cortex. Figure by Miquel Perello Nieto licensed under the Creative Commons Attribution-Share Alike 4.0 International license.

are the parvo-cellular cells. They carry the visual information like colors with a fine spatial resolution, but with a coarse temporal resolution. The seconds are the magno-cellular cells. As opposed to the parvo-cellular cells, they encode information about the speed of objects in sight with a finer resolution, but with a coarse spatial resolution. Finally, from the LGN, neurons project onto the visual cortex V1 where information from both eyes is combined. In V1, some cells have the property of encoding the apparent vergence disparity resulting from the incorrect vergence fixation of the presented stimulus.

In our model of AEC, we will also differentiate between 2 pathways of information which we call parvocellular and magnocellular. The parvocellular pathway conducts binocular frames coming from the two eyes, while the magnocellular pathway conducts pairs of consecutive binocular frames, thus capturing temporal information like the speed of the objects in sight.

5.3.3 Methods

In order to analyse the properties of Deep Auto-Encoders when applied on binocular data, we simulated 2 eyes separated by 6.8 cm within the robotic simulation software CoppeliaSim (previously named V-REP) and using the python API PyRep [49]. Following the work from [73, 74], each of the 2 eyes is composed of multiple cameras with different opening angles and operating at different resolutions in order to simulate different “spatial scales”. This

way, we can emulate in simulation how the human visual acuity decreases as we move away from the fovea. In front of the 2 eyes, we placed a square-shaped screen on which we can programmatically apply different textures. The screen can be placed at any distance from the 2 eyes and can move in any direction at a controlled uniform speed.

The 2 eyes have the following 4 degrees of freedom:

- *pan:* joint horizontal movement of the two eyes
- *tilt:* joint vertical movement of the two eyes
- *vergence:* inward and outward movement of the two eyes
- *cyclovergence:* inward and outward rotation of the two eyes around the line of gaze

In the reminder, we will refer to the pan, tilt, vergence, and cyclovergence degrees of freedom as *joints*.

Note that, in order to perform vergence and cyclovergence fixation, the agent must find the correct vergence and cyclovergence *angles* that will focus the attention of the agent on the screen, while in order to perform tracking, the agent must find the pan and tilt *angular speeds* that match the angular speeds of the screen. There is thus a difference in nature between the vergence / cyclovergence joints and the pan / tilt joints. The first operate on absolute angles while the second operate on angular speeds. In our model, this is reflected in the two pathways of information. The parvocellular pathway controlling the vergence and cyclovergence angles has no information about speeds (it encodes single left / right binocular pairs), while the magnocellular pathway does represent temporal information as it encodes jointly 2 consecutive left / right binocular pairs. In practice, the 2 frames (parvocellular) or 4 frames (magnocellular) are concatenated together along the color dimension prior to be encoded. This concatenation step is analogue to the cross-over taking place at the optic chiasm and the information split taking place at the LGNs (see Fig. 5.1). Indeed, the information from each retina is separated in 4 quadrants (NE, NW, SE, SW) in 2 consecutive steps. First at the optic chiasm the left eye / right eye information separation becomes a East quadrants / West quadrants separation. Next, at the LGNs the information stream is split in two pathways: one for the northern and one for the southern quadrants. This way, as the information reaches V1, the information from corresponding quadrants from the 2 retinas are superimposed.

In the current section, although we needed to explain the difference in nature between the 2 types of joints (parvo/magno-cellular), we will not concern ourselves with the generation of motor commands (this is the subject of section 5.4). The current section will deal with the

learning of a representation that we can use to derive an IM. More precisely, in this section we aim at highlighting the property that for each of the 4 joints, the acuity of the agent's mental representation is the highest when the joint errors equal zero. Or in other words, we aim at showing that the smallest possible reconstruction error of the Auto-Encoder is reached when the joint errors equal zero. Joint errors are defined as the difference between the angle or angular speed of the joints and the corresponding angle or angular speed of the screen

$$e_p = \dot{\alpha}_p - \dot{s}_p \quad \text{for pan,} \tag{5.1}$$

$$e_t = \dot{\alpha}_t - \dot{s}_t \quad \text{for tilt,} \tag{5.2}$$

$$e_v = \alpha_v - s_v \quad \text{for vergence and} \tag{5.3}$$

$$e_c = \alpha_c - s_c \quad \text{for cyclovergence} \tag{5.4}$$

with α and $\dot{\alpha}$ the joints angles and angular speeds, and s and $\dot{s}$ the position vector and speed vector of the screen expressed in the referential of the eyes. Note that for the cyclovergence, the value s_c denotes the cyclovergence angle of the eyes that would achieve the best fixation of the screen. This value is not trivial to obtain, so we limited our performance measurements to the case when the screen is at the same height as the eyes, in which case $s_c = 0$ and as a consequence, $e_c = \alpha_c$.

For each of the 4 joints, our approach is composed of 3 steps. First we will generate a dataset with controlled error statistics, then we will train Deep Auto-Encoders at different spatial scales to encode the samples from this dataset, and finally we will verify that the expected property is true using a validation dataset.

Step 1: Constructing the Training Datasets

A training dataset in our experiment is determined by a tuple (J, S, σ, N) with

- $J \in \{\text{pan}, \text{tilt}, \text{vergence}, \text{cyclovergence}\}$ characterizing for which joint the dataset is constructed
  - if $J \in \{\text{vergence}, \text{cyclovergence}\}$: the dataset consists in binocular pairs (I_L, I_R)
  - else if $J \in \{\text{pan}, \text{tilt}\}$: the dataset consists in temporal binocular pairs $(I_{L,t-1}, I_{R,t-1}, I_{L,t}, I_{R,t})$
- $S = \left(S_0, \ldots, S_{|S|}\right)$ is a list of tuples $S_i = (\alpha_i, r_i)$, each describing the properties of a "spatial scale" and with α_i the camera opening angle and r_i the height/width resolution

of the camera in pixels. In practice, we never used more than 2 spatial scales, and we will therefore replace the indexing S_i with the notation S_{fine} or S_{coarse}. We used $\alpha_{\text{fine}} = 9.0°$ and $\alpha_{\text{coarse}} = 27.0°$, $r_{\text{fine}} = r_{\text{coarse}} = 32\text{px}$.

- σ represents the standard deviation of a zero-centered gaussian from which the error of the joint J is sampled. All other joint errors are set to 0.
- N represents the size of the dataset.

The algorithm 5.1 describes how the training dataset samples are generated. In practice, we generated the combinations of training datasets $(J, (S_{\text{fine}}, S_{\text{coarse}}), \sigma, 10000)$ for all $J \in \{\text{pan}, \text{tilt}, \text{vergence}, \text{cyclovergence}\}$ and all $\sigma \in \{0, 1, 2, 4, 8, 16\}$, resulting in 24 distinct datasets.

Algorithm 5.1: Generating a new sample for a training dataset

place the screen at a random distance $d \sim \mathcal{U}(0.5, 5)$;
reset the vergence joint such that the eyes fixate the screen ;
reset the cyclovergence joint to an angle of $0.0°$;
reset the pan and tilt absolute angle and angular speed to $0.0°$ and $0.0°.s^{-1}$;
sample the joint error $e \sim \mathcal{N}(0, \sigma)$;
if J **is** *pan* **then**
    $(I_{L,t-1}, I_{R,t-1}) \leftarrow$ retrieve binocular pair ;
    move the screen by e pixels **horizontally** ;
    $(I_{L,t}, I_{R,t}) \leftarrow$ retrieve binocular pair ;
    $S \leftarrow (I_{L,t-1}, I_{R,t-1}, I_{L,t}, I_{R,t})$;
else if J **is** *tilt* **then**
    $(I_{L,t-1}, I_{R,t-1}) \leftarrow$ retrieve binocular pair ;
    move the screen by e pixels **vertically** ;
    $(I_{L,t}, I_{R,t}) \leftarrow$ retrieve binocular pair ;
    $S \leftarrow (I_{L,t-1}, I_{R,t-1}, I_{L,t}, I_{R,t})$;
else if J **is** *vergence* **then**
    move the vergence angle by e pixels ;
    $(I_L, I_R) \leftarrow$ retrieve binocular pair ;
    $S \leftarrow (I_L, I_R)$;
else if J **is** *cyclovergence* **then**
    move the cyclovergence angle by e degrees ;
    $(I_L, I_R) \leftarrow$ retrieve binocular pair ;
    $S \leftarrow (I_L, I_R)$;
add the sample S to the dataset ;

Step 2: Training the Deep Auto-Encoders on the Dataset

Let $v = \left(v_0, \ldots, v_{|S|}\right)$ denote one of the two visual sensory information streams (parvo- or magno-cellular) with v_i the information from the spatial scale indexed i. For each spatial scale, let E_i and D_i be, respectively, the encoder and decoder parts of an Auto-Encoder, such that

$$s_i = E_i\left(v_i, \theta_{E_i}\right) \text{ and} \tag{5.5}$$

$$\tilde{v}_i = D_i\left(s_i, \theta_{D_i}\right) \ , \tag{5.6}$$

with s_i representing the encoding of v_i and $\tilde{v}_i$ its reconstruction. The loss function for training the encoder and decoder weights θ_{E_i} and θ_{D_i} - also called the reconstruction error - is defined as the mean pixel-wise MSE

$$l_i = \frac{1}{3N_{\text{pixels}}} \sum (v_i - \tilde{v}_i)^2 \tag{5.7}$$

The mean total reconstruction error is defined

$$l = \frac{1}{|S|} \sum_i l_i \tag{5.8}$$

The auto-encoder for each scale corresponds to a 3-layered fully-connected network encoding patches of size 8×8 pixels. This patch-wise autoencoder is implemented as a convolutional neural network with filter size 8×8 in the first layer and 1×1 in the following layers. More details about the Auto-Encoder architecture can be found in the appendix in table B.1.

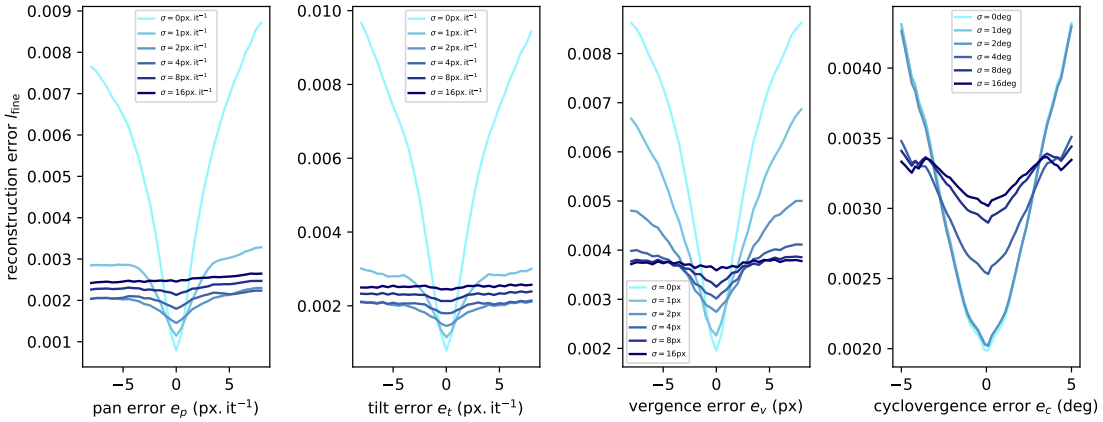
The network is trained on 50.000 batches of data of size 64 samples taken at random in the training dataset.

Step 3: Constructing a Testing Dataset for Measuring the Performance of the Deep Auto-Encoders

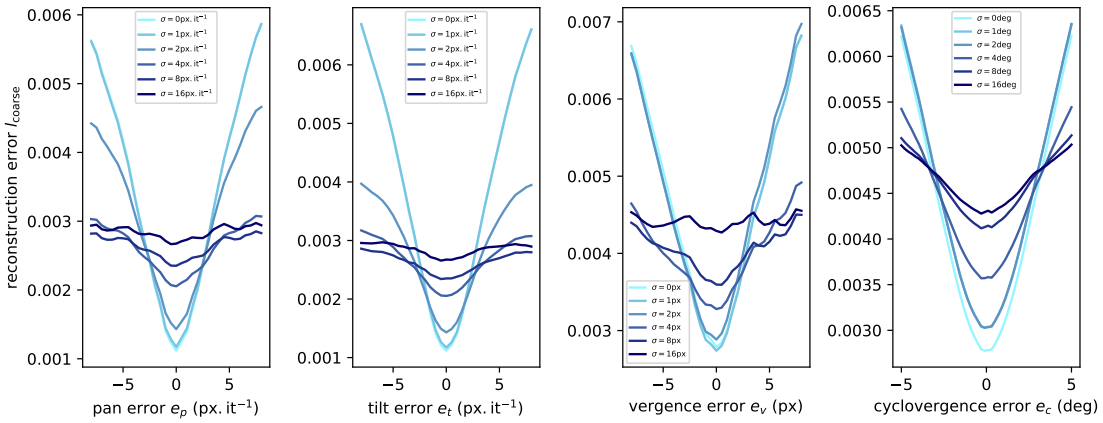
Once the Auto-Encoders have been trained on the dataset, we are interested in measuring the mean reconstruction errors l_i of the networks as a function of the joint errors. To this end, we constructed for each joint a testing dataset comprised of $(\text{images}, \text{joint error})$ pairs. Algorithm 5.2 summarizes how the testing datasets are obtained.

Algorithm 5.2: Constructing the testing dataset

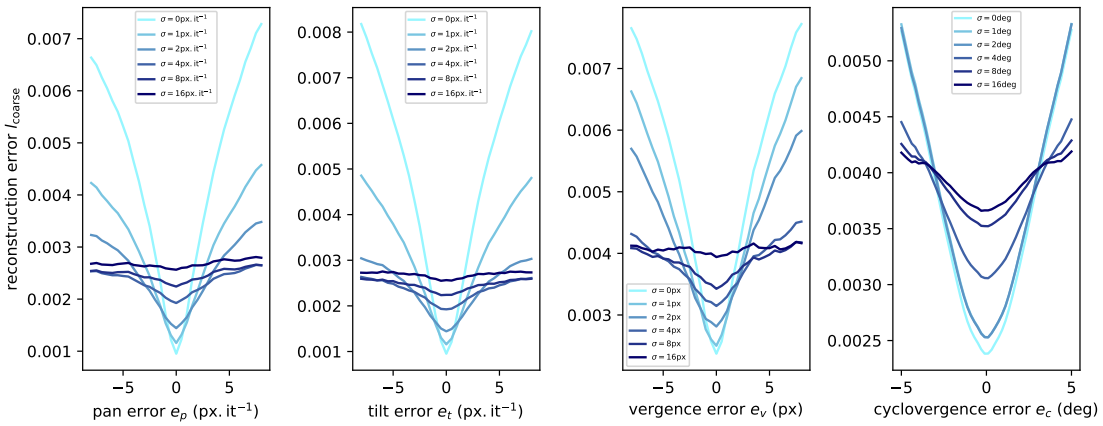
place the screen at a fixed distance $d = 2.5\text{m}$;
reset the vergence joint such that the eyes fixate the screen ;
reset the cyclovergence joint to an angle of $0.0°$;
reset the pan and tilt absolute angle and angular speed to $0.0°$ and $0.0°.s^{-1}$;
for $e \leftarrow e_{min}$ **to** e_{max} **by** e_{step} **do**
    else if J **is** *vergence* **then**
        move the vergence angle such that the vergence error equals e pixels ;
    else if J **is** *cyclovergence* **then**
        set the cyclovergence angle to e degrees ;
    for *each texture* **do**
        apply current texture on the screen ;
        if J **is** *pan* **then**
            $(I_{L,t-1}, I_{R,t-1}) \leftarrow$ retrieve binocular pair ;
            move the screen by e pixels **horizontally** ;
            $(I_{L,t}, I_{R,t}) \leftarrow$ retrieve binocular pair ;
            $S \leftarrow (I_{L,t-1}, I_{R,t-1}, I_{L,t}, I_{R,t}, e)$;
            move the screen by e pixels in the opposite direction ;
        else if J **is** *tilt* **then**
            $(I_{L,t-1}, I_{R,t-1}) \leftarrow$ retrieve binocular pair ;
            move the screen by e pixels **vertically** ;
            $(I_{L,t}, I_{R,t}) \leftarrow$ retrieve binocular pair ;
            $S \leftarrow (I_{L,t-1}, I_{R,t-1}, I_{L,t}, I_{R,t}, e)$;
            move the screen by e pixels in the opposite direction ;
        else if J **is** *vergence* **or** J **is** *cyclovergence* **then**
            $(I_L, I_R) \leftarrow$ retrieve binocular pair ;
            $S \leftarrow (I_L, I_R, e)$;
        add the sample S to the dataset ;
    end
end



(a) - fine scale - Reconstruction error l_{fine} of the fine scale Auto-Encoder as a function of the joint errors.



(b) - coarse scale - Reconstruction error l_{coarse} of the fine scale Auto-Encoder as a function of the joint errors.



(c) - both scales - Reconstruction error l of the fine and coarse scale Auto-Encoders as a function of the joint errors.

Fig. 5.2 Reconstruction errors (a) l_{fine} (b) l_{coarse} and (c) l as a function of the joint errors e_p, e_t, e_v and e_c for varying distributions of the joint's error in the training dataset.

5.3.4 Results

Figure 5.2 summarizes the results when training the fine and coarse Auto-Encoders on each of the datasets. Figure 5.2a shows the reconstruction error l_{fine} of the fine Auto-Encoder as a function of the joint errors e_p, e_t, e_v and e_c for each standard deviation σ. The joint errors are measured in pixels, rather than in degrees for the sake of clarity. For a camera angle of 9° and a resolution of 32 pixels, $1\text{px} = \frac{9}{32} = 0.28125°$. Figure 5.2b presents the same results but for the coarse Auto-Encoder. The joint errors for the coarse scale are expressed in the same unit as the fine scale, meaning that the errors are measured in fine-scale pixels (0.28125°), and not in coarse-scale pixels ($\frac{27}{32} = 0.84375°$). Finally, figure 5.2c shows the results for both spatial scales. Note that in all cases, the angles for the cyclovergence joint are always expressed in degrees. A rotation of 2° of the cyclovergence joint correspond to the rotation of both eyes by 1° in opposite directions.

In line with our predictions, the curves present a characteristic V-shape centered around 0, confirming that the minimum reconstruction error is reached when the joint errors are minimum. Moreover, the plots display clearly that this property is further reinforced when the proportion of samples in the training dataset with small joint errors increases (i.e. when σ decreases). This implies that in the online setting (i.e. when the eyes are controlled by a learning RL agent), as the agent learns to cancel out the joint errors, the V-shape profile of the reconstruction error gets reinforced.

Finally, by comparing the measurements obtained for the fine and coarse scales, we can confirm the observations made in [73, 74] about the effect of using different scales. The fine scale's preference for smaller joint errors is more pronounced than that of the coarse scale, while the coarse scale captures a wider range of apparent disparities.

5.3.5 Conclusion

In Chapter 3 we showed the general property of learned encoding algorithm to preferentially retain redundant information. In this Section we showed that this applies in particular to the encoding of binocular pairs with varying disparities. In the next section, we will define an intrinsic reward that will encourage the agent to produce movement commands that lower the resulting reconstruction error. Fig. 5.2 shows that the optimal policy wrt. this intrinsic reward should learn to minimize all joint errors, that is, to fixate the screen, to track it in time and to perform correct cyclovergence control.

Fig. 5.3 Left / right anaglyph of the two eyes before (left) and after (right) a fixation movement. The images from the left and right eyes are converted to grey scale and then superimposed as two color channels of a single image.

5.4 Active Efficient Coding with Deep Autoencoders

In this section, we will derive an intrinsic reward signal from the measurements obtained in the previous section, and quantify the performance of the resulting emerging behavior. This work is an extension of our publication for the *International Conference on Development and Learning* in 2020 called *Self-Calibrating Active Binocular Vision via Active Efficient Coding with Deep Autoencoders* [147].

5.4.1 The Model

Sensory Encoding Via Deep Autoencoders

When the two eyes verge on the same point, the foveal regions of the two retinal images become more and more similar. As a consequence, the mutual information between the left and right foveal image representations $\mathrm{MI}(I_\mathrm{L}, I_\mathrm{R})$ for a given disparity d increases as d goes to 0. It indicates how redundant the left and right images are and reflects the quality of the fixation. Similarly, tracking a moving object can be achieved by maximizing the information redundancy of the foveal image region across time by maximizing $\mathrm{MI}(I_t, I_{t-1})$ (for one or more eyes).

Accordingly to the experiment presented above, we propose to measure the redundancy in the visual data via training an Auto-Encoder, following the hypothesis that an information stream is better reconstructed when it is more redundant (see the offline measurements in Figs. 5.2, and the online measurements in Fig. 5.5). The first advantage of this technique is that it is agnostic to the underlying data representation (for example the RGB channels in left and right data streams could be expressed in different bases or be non-linearly transformed, as

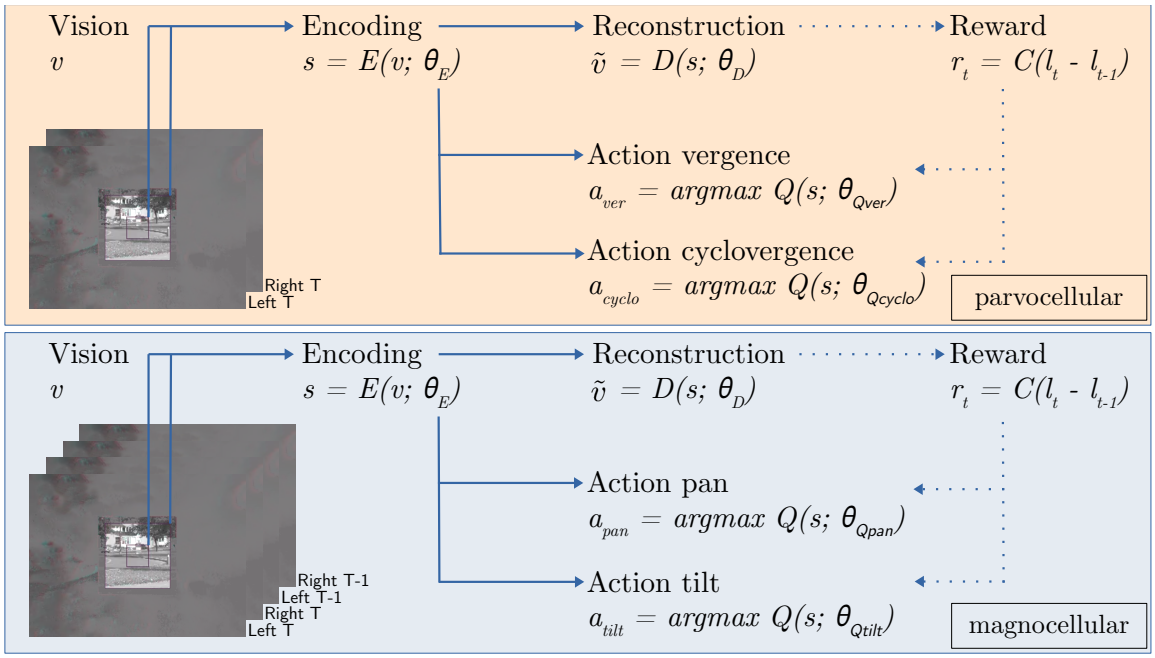


Fig. 5.4 Architecture of the model. Two regions are extracted at the center of the left / right camera images and encoded. The condensed representation is used to train the Q-function. The latter is trained to maximize the reward, which is proportional to the improvement of the reconstruction error of the encoder.

only the data redundancy truly matters). This makes the system robust to noise and hardware failures. The second advantage of this technique is that it learns a condensed representation of the sensory information which can be used by other learning components of the system. Such lossy compression of information may be essential for learning abstract representations at higher processing levels. Finally, as discussed in section 3.5.3, the algorithm could also exploit highly non-linear redundancies between different sensory modalities, given that the encoding network is sufficiently deep such that it captures these redundancies. This could enable to model behaviors related to multimodal contingencies: the resulting agents would learn behaviors directed toward the acquisition of information simultaneously through multiple sensory modalities, like for example by simultaneously looking at an object and touching it. This kind of IM could serve as a basis for modeling the emergence of visuo-motor contingencies.

Similarly to before, let $v = \left(v_0, \ldots, v_{|S|}\right)$ denote one of the two visual sensory information streams (parvo or magno-cellular) with v_i the information from the spatial scale indexed i. Again, for each spatial scale, let E_i and D_i be, respectively, the encoder and decoder parts of

an Auto-Encoder, such that

$$s_i = E_i\left(v_i, \theta_{E_i}\right) \text{ and} \tag{5.9}$$

$$\tilde{v}_i = D_i\left(s_i, \theta_{D_i}\right) , \tag{5.10}$$

with s_i representing the encoding and $\tilde{v}_i$ its reconstruction. The loss for training the Auto-Encoder for the scale i is the scale's mean reconstruction error

$$l_i = \frac{1}{3N_{\text{pixels}}} \sum (v_i - \tilde{v}_i)^2 \tag{5.11}$$

and the mean total reconstruction error for all scales is defined

$$l = \frac{1}{|S|} \sum_i l_i \tag{5.12}$$

.

Additionally, we will define the compressed visual information representation as the union of all the s_is

$$s = \left(s_0, s_1, \ldots, s_{|S|}\right) \tag{5.13}$$

$$= E\left(v, \theta_E\right) \tag{5.14}$$

The notation $E\left(., \theta_E\right)$ is used to denote the union of all encoders $E_i\left(., \theta_{E_i}\right)$.

Intrinsically Motivated Reinforcement Learning Formulation

We consider the classical Markov decision process framework, where at discrete time $t = 0, 1, 2, \ldots$ an agent observes sensory information $s_t = E\left(v_t, \theta_E\right)$ and chooses action a_t according to the distribution $\pi\left(a_t | s_t\right)$. After applying the action, the agent transitions to a new state according to a transition function $s_{t+1} = T\left(s_t, a_t\right)$, and receives a reward r_t. While Reinforcement Learning classically considers a reward provided by the agent's environment through a potentially stochastic reward model, we here define an intrinsic reward based on the agent's sensory encoding of its environment. Specifically, we define the reward

$$r_t = C\left(l_t - l_{t+1}\right) , \tag{5.15}$$

where C is a scaling factor. This reward signal measures the *improvement* of encoding quality, i.e., it favors movements that cause transitions from high to low reconstruction error of the Auto-Encoder representing the visual input.

The goal of Reinforcement Learning is to learn a policy function π that maximizes the (discounted) sum of future rewards R_t called return

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{5.16}$$

Where γ is a discount factor in $[0,1]$ ensuring the convergence of the reward sum. In this particular application of Reinforcement Learning, the agent does not need to plan ahead his behavior, consistent with our observation that the algorithm works best for $\gamma = 0$.

RL-Algorithm

We opted for a synchronously parallelized version of the DQN algorithm [83, 82]. It consists of a Q-value function approximation

$$q = \begin{pmatrix} q_0 \\ \vdots \\ q_n \end{pmatrix} = Q(s, \theta_Q) \ , \tag{5.17}$$

where q_j represents an estimate of the return after performing discrete action j in state s and n is the number of possible actions. The loss for training the Q-function is the Huber loss between the estimate and the return target [46]. Exploration during the training phase is performed via an ε-greedy sampling.

The critic network Q can be described in 2 parts. The first part operates individually on each scale. It is composed of a convolutional layer followed by a pooling layer. The results are then flattened and concatenated before being processed by 2 fully-connected layers in the second part. More details about the critic network architecture are given in the appendix in Tab. B.1.

5.4.2 Experimental Setup

Experiments were divided in episodes of 10 iterations. At the beginning of each new episode, a random screen distance is uniformly sampled in $[0.5,5]$(m), a random screen speed is sampled in $[0,4]$(px/iteration), a random movement direction is sampled in $[0,360]$ (deg), the vergence joint of the robot is reset such that the agent gaze focuses at a random distance sampled in $[0.5,5]$(m) and the cyclovergence angle in sampled in $[-5,5]$(deg). A new texture for the screen is chosen at random amongst natural stimuli taken from the *McGill Calibrated Color Image Database* [91].

All networks are trained using the Adam algorithm [55] with a learning rate of 5.10^{-4}. We use a value of $\varepsilon = 0.05$ for the ε-greedy sampling. Each time an episode is simulated, all its transitions are placed in a replay buffer of size 1000 and a batch of data is then sampled uniformly at random from the buffer for training the networks. We use a batch size of 200. In practice, we implemented hardware acceleration by parallelizing the data collection following the principle of *synchronous Reinforcement Learning*. We used a pool of 20 simulations in parallel for gathering training data, and we found that the size of the pool has almost no effect on the normalized training speed of the algorithm.

The pan, tilt and vergence joints used the same following action discretization: $\left[-4,-2,-1,-\frac{1}{2},0,\frac{1}{2},1,2,4\right]$ px/iteration^2 (pan and tilt), px/iteration (vergence). The action-set for the cyclovergence joint is $[-3.58,-1.79,0,1.79,3.58]$ degree/iteration. The vergence angle of the robot's eyes is constrained between 0° (parallel optical axes of the two eyes) and 20° (inward rotation of each eye by 10°) and the cyclovergence angle between $-10°$ and 10°. The pan and tilt joints are constrained to remain in $[-10°, 10°]$.

At regular intervals, the training is paused, and the agents' performances are measured. For evaluating the agents' performances, we gather two sets of data at each testing step. One, the *controlled-error set*, gauges the performance of the agents under defined apparent disparities. This is the same as the testing dataset that we used in the previous section. The other, the *behaviour set*, measures how the policies of the agents recover from initial disparities. All measurements are repeated for 20 stimuli displayed on the screen 2 m away from the eyes of the robot. To construct the controlled-error set, we simulated various pan, tilt and vergence errors by manually setting the speed of the screen and the vergence angle of the eyes. Only one joint was tested at a time, meaning that the errors for all other joints were artificially set to 0. We then recorded the reconstruction errors of the fine and coarse scales and the agents' preferred actions. The behaviour set is the recording of 20 iterations of the agents' behaviour, starting from controlled initial pan, tilt, cyclovergence or vergence errors.

5.4.3 Results

For successful learning, the pan, tilt, vergence and cyclovergence errors must become reflected in the reconstruction errors of the encoders, as explained in Section 5.3. We start by analyzing the reconstruction errors of the encoders for every pan, tilt, vergence and cyclovergence error in the online setup, similarly to the previous study conducted with controlled joint error statistics. Figure 5.5 shows that for each joint, the reconstruction error is minimal when the absolute joint error is minimal. In particular, Fig. 5.5 shows the reconstruction error for each individual stimulus displayed on the screen (in blue) and the average for all stimuli (in red). Repeating the same analysis using random weights for

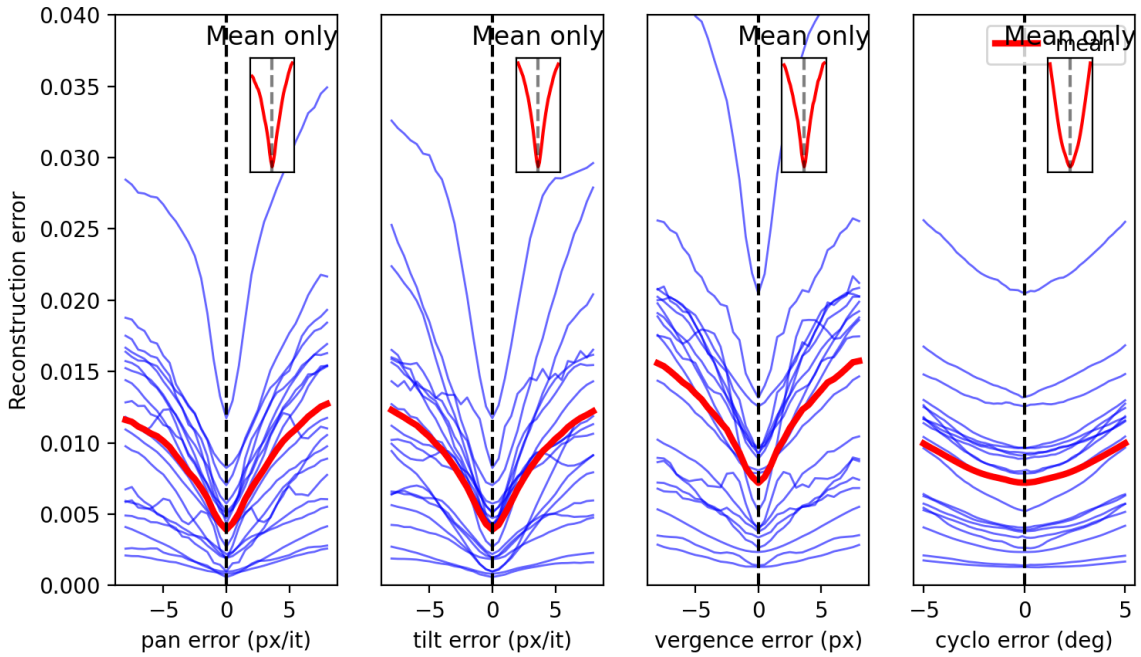


Fig. 5.5 The autoencoders' reconstruction errors averaged across the two scales as a function of the pan, tilt, vergence and cyclovergence error. Each blue curve corresponds to a different stimulus displayed on the screen. The red curve represents the mean. For each plot, the error for the two other joints is set to 0.

the encoder and decoder shows no difference in the reconstruction quality for low or high absolute errors (the mean error curve is flat instead of being V-shaped, with values around 0.27, not shown). The characteristic V-shaped curves are a consequence of both learning a compact code of the visual input and adapting the behavior to shape the statistics of the visual input [158], as shown in Fig. 5.2.

To show the precision of the learnt policies, we represent for each joint the probability of selecting each possible action in the action set as a function of that joint's error in Fig. 5.6. The diagonal shapes in the three policies indicate that the model has learned to accurately compensate for any pan, tilt, vergence and cyclovergence errors.

To show the speed at which the algorithm converges, Fig. 5.7 plots the average testing error at the end of episodes, that is, after 10 iterations, as a function of training time. The testing absolute error is measured at regular intervals as the mean absolute joint error, starting from initial errors taken in $[-4;4]$ px (vergence) or px/iteration (pan and tilt). The initial cyclovergence errors are taken in $[-5;5]$ degrees.

Finally, to show how quickly and accurately the algorithm fixates objects and tracks them, Fig. 5.8 shows the mean accuracy and its standard deviation, during 20 consecutive iterations,

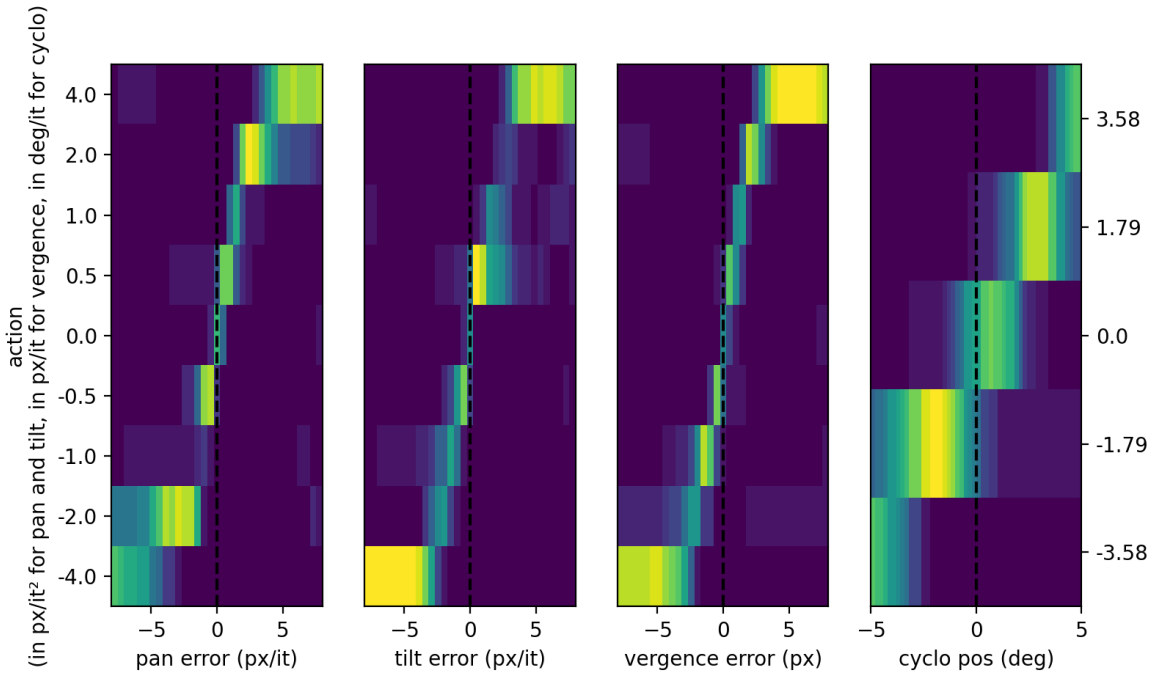


Fig. 5.6 Probability of choosing each action in the action sets as a function of the pan, tilt, vergence and cyclovergence errors. For each subplot, the error for the three other joints has been set to 0.

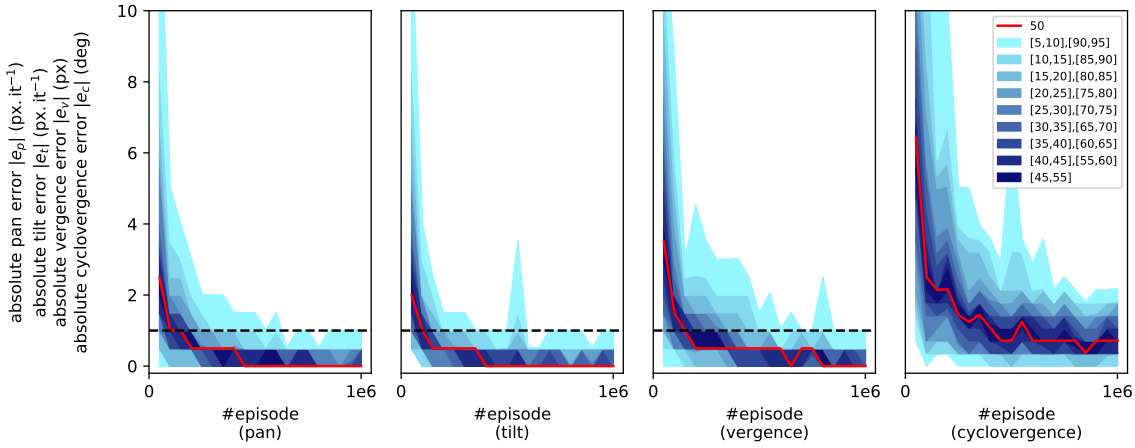


Fig. 5.7 Reduction of errors with training time. The red curves represent the median absolute error for each joint. The shaded regions show the distribution of the absolute error. The dotted horizontal line shows the theoretical precision limit of 1 px/iteration (pan and tilt) or px (vergence). All joints display sub-pixel fixation accuracy.

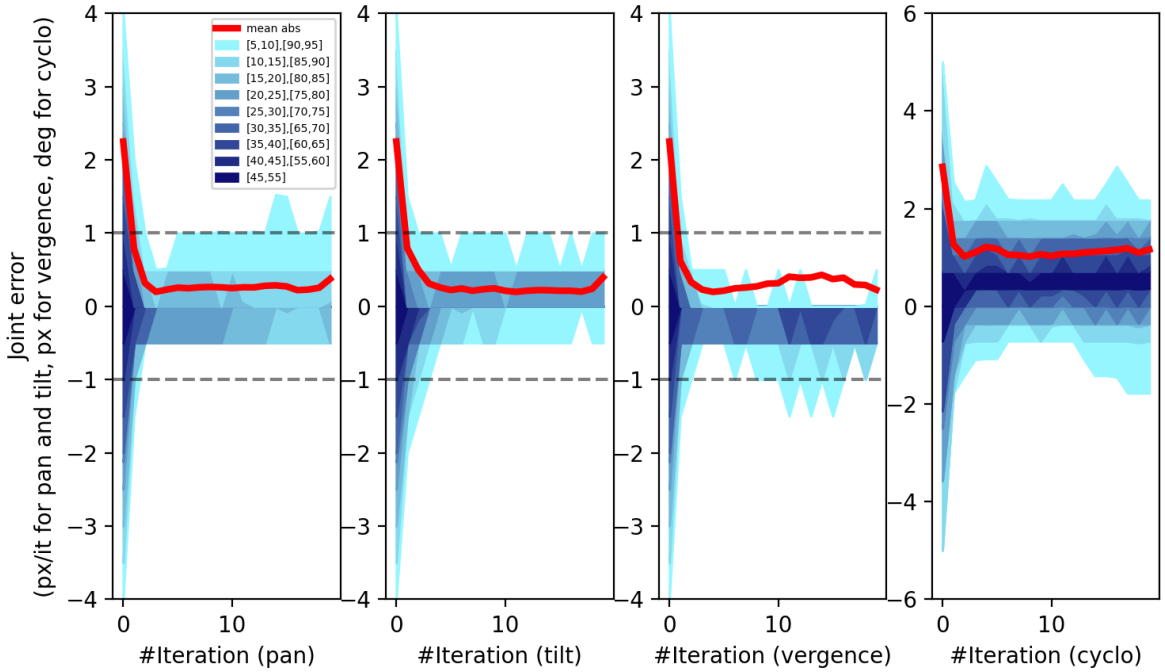


Fig. 5.8 Rapid object fixation and tracking. This figure represents the joint errors as a function of the iteration number within an episode. Similarly to Fig. 5.7, the coloured regions indicate the distribution of the joit errors. The red curves show the mean absolute error. The dashed lines represent the $[-1,1]$ px (vergence) or px/iteration (pan and tilt) interval. Pan, tilt, vergence and cyclovergence errors are decreasing quickly during one episode and typically reach sub-pixel levels in one or two steps. The shaded region indicates one standard deviation.

starting from evenly spaced errors in $[-4,4]$ px (vergence) or px/iteration (pan and tilt) and $[-5;5]$ degrees (cyclovergence). Subpixel error levels are typically reached in just one or two iterations despite the discrete action set. Note that an error of, e.g., 3 pixels cannot be reduced to zero in a single step because the two closest discrete actions of 2 pixels and 4 pixels would both lead to a remaining error of 1 pixel.

5.5 Conclusion

In this Chapter, we first presented Efficient Coding and its extension, Active Efficient Coding. We then reviewed existing work in the literature implementing the AEC principle. We proposed to replace the Matching Pursuit encoding algorithm by a more modern Deep Auto-Encoder. To this end, and in line with the research presented in Chapter 3, we first verified the property that an increase of the redundancy in the input sensory information

leads to an improvement of the encoding quality. We then exploited this property to define an Intrinsic Motivation for training a RL algorithm. The results show that the resulting agent autonomously learns to perform accurate vergence and cyclovergence fixation, as well as to track moving objects.

Follow-up work could investigate the benefit of *differentiating through* the critic loss, back to the encoder part of the Auto-Encoder. Since the apparent disparities are a predominant factor in order to predict the actions' values, we can anticipate that optimizing the weights of the encoder so as to also minimize the critic loss would have the effect of reinforcing the features that encode disparities in the learned representation.

Next, it would be interesting to compare the AEC-IM with the Prediction-Based IM proposed by Schmidhuber in [118–120, 122]. In these research, the agent learns to maximize its own error at predicting the near future. In theory, PB-IMs could be used for learning tracking of object, by rewarding the agent not for making big prediction errors, but rather for making accurate prediction of future observations. It is unclear however if the same kind of IM would also suit the learning of vergence and cyclovergence control. Indeed, predicting the next binocular pair is the easiest when the 2 eyes do not move. However, a such *temporal* PB-IM could also yield successful vergence and cyclovergence fixation if the loss in prediction quality engendered by the eye movement is compensated by a gain in reconstruction quality due to the increased redundancy in the left / right pair. For these two joints, an other kind of PB-IM similar to the *cross-modality prediction* presented in Chapter 3 could be used. The agent would then be rewarded for accurately predicting the left frame given the right frame, and vice-versa. As we have shown in the previous Chapter, the *cross-modality prediction* scheme tends to extract the mutual information between multiple information fluxes. Thus the magnocellular pathway would aim at maximizing the mutual information between 2 consecutive binocular pairs, while the parvocellular pathway would maximize the mutual information between the left and right frames within one binocular pair.

Finally, combining the work from Chapters 3 and 5, one could investigate the potential of Intrinsically Motivated visuo-motor coordination. By exploiting the fact that when an agent sees its own arm, some information is shared between the proprioceptive and visual modalities (see Fig. 3.6), one could try to develop an AEC-IM or a PB-IM that would encourage the agent to seek for this shared information. Foreseeably, such an agent would learn to look at the part of its arm that gives it the most information about its position, i.e. the wrist or the hand. Combined with AEC applied to binocular vision, the agent would potentially learn to fixate its hand, and pursue it as it is moving, enabling to learn to represent the world around itself in the referential frame of the body, rather that in the referential frame of the eyes or the hand. It is likely that a rich behavior would only emerge from the

combination of these different IMs. The coordination and the scheduling of the learned intrinsic skills is the subject of Hierarchical Reinforcement Learning, which we briefly introduced with the work of Marvin Minsky in section 2.1 of this thesis.

Chapter 6

Hierarchical Reinforcement Learning

6.1 Introduction

In previous chapters, we introduced Deep Reinforcement Learning and we showed how RL has been successfully applied to a wide range of robotic tasks, in simulation or on real robotic platforms. We then presented the sub-field of Intrinsically Motivated Reinforcement Learning and showed how some tasks are inherently centered around general information theoretic principles (see for example how the video game *Super Mario Bross* can be solved solely by an agent maximizing a prediction error). This chapter will aim at introducing Hierarchical Reinforcement Learning (HRL).

To better understand the motivations behind HRL, let us first discuss what are the main limitations of traditional RL. The first is its *sample inefficiency*, meaning that RL algorithms need a lot of data-points to converge. This is very problematic, as the data-generation is often a bottleneck. The second big limitation of RL, named the *curse of dimensionality*, resides in the combinatorial explosion as the dimensionality of the action space increases (the number of possible combinations of actions increases exponentially with the dimensionality). This renders the application of traditional RL to large-action-spaces problems impracticable. The third limitation of RL is its very poor generalization capability. Indeed, when trained multiple times, the same RL algorithm can solve a wide range of different tasks, but however, a trained agent can only solve one task and generally shows poor generalization to other even similar tasks. Finally, the fourth limitation of RL algorithms lies in their lack of abstraction of the states, of the actions and of temporal abstraction of the task in general, preventing abstract logical reasoning.

Hierarchical Reinforcement Learning holds the promise of alleviating all of these issues, while also resembling more the way humans and animals apprehend real world problems.

In section 2.1, we introduced the first learning robotic setup with Minsky's early work in the 70's and his book from 1988 called the *Society of Mind* [80]. Its core idea is that the mind is not composed of a single individual, but rather by a whole society of agents, each communicating with its neighbours, parents, children. In this view, a task such as *stacking a cube on a stack* is solved by a *stacking* agent part of the society, which will in turn call other agents for help in order to solve the task. In short, Hierarchical Reinforcement Learning is the name given to the branch of RL research trying to implement Minsky's idea.

The whole benefit of HRL over RL resides in a better abstraction of the tasks, which enables at the same time to reuse already acquired skills, thus alleviating data inefficiency and poor generalization capabilities, and to decompose big problems into smaller ones, thus alleviating the problem of the curse of dimensionality.

It is simple to convince oneself that in nature, humans (and some animals) decompose tasks in a hierarchy. See for example how each of us triggers a routine composed of steps in the morning - taking a shower, eating, listening to the radio, cleaning, brushing teeth, leaving - which each can be subdivided into smaller steps - turning on the water, waiting for it to be warm enough, shampoo, soap, rinsing, drying. Such mental structuring of tasks makes it possible to generalize our behavior instantly to new configurations of bedrooms, showers and kitchens. Indeed, the lower level agents are agnostic to the general configuration of the rooms, and are acting on very specific parts of the world, while the higher level agents are agnostic to the specific parts of the world and operate on a more abstract level. When learning really **is** required to adapt, low-level agents only need to be (re-)trained which enables very quick learning of new complex skills. Finally, this structuring enables reasoning at a much more abstracted level and opens the door to logical thinking and mathematical problem solving. In section 6.3 we will try to show how the learning of reaching in infants is decomposed in phases and how this hints at a hierarchical structure of human behavior.

In the HRL literature, a distinction has been drawn between two types of hierarchies. The first is called Feudal Reinforcement Learning [27] and has been proposed by Geoffrey Hinton in 1993, and the second is referred to as the Option Framework [135], proposed by Richard Sutton in 1998. In this Chapter, we will also try to relate Movement Primitives to HRL, and we will propose a formalism that describes Feudal HRL, Option HRL and Movement Primitives (MP). Next, we will present our approach to learning Movement Primitives as a starting point for implementing an HRL agent.

6.2 Related work

6.2.1 Feudal HRL

In its original publication [27], Feudal Reinforcement Learning is introduced as a way to speed up traditional RL by allowing learning to take place at multiple spatial and temporal scales simultaneously. The authors propose to organize different RL-agents in a strict hierarchy, such that each manager in the organization has a chief and subordinates. The manager then sends a command to one of its subordinates in order to achieve the task that it received from its own chief. The command is interpreted by the subordinate as part of its state space, such that its policy is conditioned on its chief's command.

The authors have identified two principles that a Feudal hierarchy must follow. The first is called *Reward Hiding*. It designates the fact that the subordinates must be rewarded for accomplishing their task, whether or not the task of the chief is accomplished.

The second is called *Information Hiding*. It designates the idea that the different managers in the hierarchy need different kinds of knowledge to take decisions. Agents higher in the hierarchy need abstract information, often temporally abstracted while agents lower in the hierarchy need higher resolution, less abstracted information.

The authors then propose an implementation of Feudal RL to path-finding in mazes. They compare the performances of the Hierarchy against a *flat* version.

While they state that Feudal Reinforcement Learning holds the promise of operating simultaneously at different resolutions of time, in practice, their implementation consists only in a spatial hierarchy.

More recently, in 2017, [142] proposed a modern redefinition of Feudal HRL based on Deep Reinforcement Learning. The resulting architecture consists in a fully differentiable 2-level hierarchy composed of one manager and one worker only. The manager sets goals for the worker in a latent space which is itself learnt by the manager. The worker is designed to interpret the goals as directions to be followed rather than absolute positions to be reached. The reformulation enables the worker and the manager to operate at different temporal resolutions. This has the benefit of facilitating long time-scale credit assignment. The new approach is compared with a standard RL architecture on the Atari game suite. Although the Feudal Architecture is designed to have only 2 levels, the authors claim that there are obvious generalizations of the architecture to more levels.

6.2.2 Options Based HRL

5 years after the original publication about Feudal RL, Richard Sutton published an article in which he introduced the concept of Options [135]. Options are build upon Semi-MDPs (SMDP) [18, 77, 96], which are analogues to MDPs but differ in that actions in SMDPs are allowed to *last* for an arbitrary amount of time. The discounting of the rewards must thus take into account the duration of the actions. An Option in the Option Framework consists in a tuple (I, π, β) where I designates a set of states s from which it is allowed to trigger the Option, where π is a traditional policy function mapping states to a probability distribution over the actions, and where β is the Option's termination probability function, conditioned on the state s. When the controlling agent selects an Option $O = (I, \pi, \beta)$ from state $s_0 \in I$, an action $a_0 \sim \pi(s_0)$ is applied, transitioning to a new state s_1, and the Option has a probability $\beta(s_0)$ to finish immediately. This process is iterated until the termination flag is sampled from β. Options can be composed hierarchically like the Feudal agents in Feudal HRL. For example, an *open-the-door* Option might use as actions 3 Options *reach-door-handle*, *grasp-door-handle* and *rotate-door-handle*.

6.2.3 Movement Primitives and Trajectory Learning

The primary goal of Movement Primitives (MP) is to allow to compose complex robot skills out of simple, canonical movements. As we will try to show, MPs are highly related to HRL. By definition, and as opposed to Options or Feudal agents, MPs consist in sequences of actions that do not depend on the intermediate states that are visited during the motion.

A significant portion of the movements that we perform daily are actually only very lightly conditioned on the state that we find ourselves in. Complex movements like walking for example only require a very limited amount of knowledge about the world around us. This is well shown in [65], where a quadrupedal robot learns to navigate through all sorts of very challenging terrain without having access to the visual modality nor any other ways of sensing the geometry of the ground other than through its legs. Furthermore, as the authors mention, it is possible to make a quadrupedal robot walk on a flat terrain only using a crude but well calibrated gaiting mechanism, instructing how to move the foot tips only as a function of time and completely independently of the state the quadruped finds itself in.

Movement Primitives relate to HRL insofar as they can be combined hierarchically in the same way Options or Feudal agents are. Moreover, most *higher level* tasks resemble scripts, in that they often only consist in a sequence of steps that can be determined ahead of execution. For example, opening a door will always be decomposed into first reaching for the handle, then grasping it, and finally rotating it, regardless of the specific position and shape

of the door and pose of the arm. Only the lower level skills (reaching the handle, grasping and turning) necessitate knowledge about the environment's configuration. *open-the-door* is thus similar to a movement primitive in the sense that the sub-actions that it takes do not depend on the intermediate states visited during the performance.

MPs have been extensively studied in the literature. In the simplest form, the experimenter pre-defines manually sequences of actions available to the agent, or he or she designs the control interface of the agent to match the specificities of the task at hands, like it is done for example in [65] and in most Quadrupedal locomotion research (see Sec. 2.3.5). In general however, MPs are fitted to a pre-recorded dataset of demonstrations given by the experimenter. Multiple implementations of such *generic* MPs have been developed over the years. Here we will introduce only 2, DMPs and ProMPs.

The most common implementation of MPs is based on Dynamic Movement Primitives (DMPs) [116]. In short, DMPs extend the dynamical system describing a PD (Proportional-Derivative) gain controller by adding an additional term (the forcing function) such that the profile of approach towards the fixed point of the system can be shaped through the tuning of additional parameters. DMPs expose interesting properties of scalability in space and time. Once a movement has been fitted, it is trivial to play it back faster, slower, or to extrapolate the motion to further or nearer target positions.

Next, [95] introduces the concept of PRObabilistic Movement Primitives (ProMPs) as a probabilistic framework for representing and learning MPs. Like DMPs, ProMPs are scalable in space and time. ProMPs additionally support parallel activation and smooth blending of multiple MPs, allowing to compose them sequentially and simultaneously. The parameters of ProMPs can be trained from demonstration or using RL algorithms. Technically, they correspond to probability distributions over trajectories from which it is possible to sample.

6.2.4 Unifying Feudal Reinforcement Learning, Options and Movement Primitives

As presented above, Feudal HRL, Options and MPs represent distinct manners of structuring a complex behavior. In some context, the skills that compose it must be parameterized by a task description vector. In that case one would prefer the Feudal HRL framework. In some other context, there is only one way a task can be accomplished and the task description vector becomes irrelevant. In that case one prefers the Option Framework. Finally, sometimes the skill only consists in accomplishing a succession of steps, parameterized or not by a task description vector. In that case, one would adopt the MP approach.

However, as we start to extrapolate the Options and the Feudal agents to allow for an uncountable space of children - i.e. an infinite amount of sub-options parameterized by a continuous vector - the difference between the two frameworks blends out. This suggests that it must be possible to establish a more general formalism that could encompass both Frameworks.

If we leverage the notions of initiation set I and termination function β, a manager agent, picking among children Options can be described by a function

$$k^{0\to1} = \Pi\left(s^0\right) \tag{6.1}$$

with $k^{0\to1} \in \mathscr{K}$ an integer number describing which Option has been picked at the level 1 by the level 0, and $\mathscr{K}$ the action space of the manager. Then, instead of defining each Option individually as in [135] we define them as a *brotherhood*

$$k^{1\to2} = \pi\left(s^1, k^{0\to1}\right) \tag{6.2}$$

. Similarly, Feudal agents can be described by the controller policy

$$\left(k^{0\to1}, p^{0\to1}\right) = \Pi\left(s^0\right) \tag{6.3}$$

with $k^{0\to1} \in \mathscr{K}$ an integer number describing which child Feudal agent has been picked at the level 1 by the level 0, $\mathscr{K}$ the manager's action space and $p^{0\to1} \in \mathscr{P}$ the task parameterization vector passed to the child. The Feudal *brotherhood* is then defined

$$\left(k^{1\to2}, p^{1\to2}\right) = \pi\left(s^1, k^{0\to1}, p^{0\to1}\right) \tag{6.4}$$

From this formulation of Feudal HRL and Options, it stands out that the only difference between them is the nature of the information going from the controller at level 0 to the worker at level 1. For the Option framework (Eq. 6.1) it consists in only $k^{0\to1}$, an integer value determining which Option is selected. For the Feudal framework (Eq. 6.3) it consists in $k^{0\to1}$ and $p^{0\to1}$, where $k^{0\to1}$ is again an integer value determining which child agent to select, and where $p^{0\to1}$ is the parameter passed to it, and which is often a multidimensional vector living in a continuous space.

Note that in Eq. 6.2 we assumed that agents at the level 2 were Options, while in Eq. 6.4 we assumed that the agents at level 2 were Feudal. It is possible however to mix Options and Feudal agents within the same hierarchy.

As stated above, if we try to generalize the Option framework such that agents in the hierarchy can chose among a continuous set of child Options, the boundary between the

Options and Feudal Frameworks starts to vanish. Formally, doing so corresponds to taking $k^{0\to1}$ in a continuous space rather than a discrete one in Eqs. 6.1 and 6.2. The resulting system is equivalent to a hierarchy composed of a controller sending commands to a single Feudal agent.

We propose that the exact nature of a brotherhood of agents can be described by the nature of the space in which the information transferred between two levels is living. We will reuse the notation $k \in \mathscr{K}$ to describe this information:

$$k^{0\to1} = \Pi\left(s^0\right) \tag{6.5}$$

The different possible topologies of the space $\mathscr{K}$ describe the different possible hierarchies. The elements $k \in \mathscr{K}$ can be interpreted as *task descriptors*.

- If $\mathscr{K} = \{0,1,\ldots,n\}$, the controller chooses among n Options.
- If $\mathscr{K} = \mathbb{R}^n$, the controller passes a n-dimensional task parameterization vector to its single Feudal child.
- If $\mathscr{K} = \{0,1,\ldots,n\} \times \mathbb{R}^m$, the controller passes a m-dimensional task parameterization vector to one of its n Feudal children.
- If $\mathscr{K} = \{0,1\} \times \mathbb{R}^2 \cup \{2,3\} \times \mathbb{R}^3$, the controller passes either a 2-dimensional task parameterization vector to either one of its Feudal children indexed 0 or 1, or a 3-dimensional one to its children 2 or 3.
- $\mathscr{K} = \{0,1\} \times \mathbb{R}^2 \cup \{2,3\}$ describes a brotherhood composed of a mixture of Options and Feudal agents: the controller passes either a 2-dimensional task parameterization vector to either one of its Feudal children indexed 0 or 1, or calls the Option 2 or 3.
- ...

To make our definition of HRL complete, agents in the hierarchy must be accompanied by a termination function $\beta(s)$ evaluated before and after every policy evaluation.

Then, in order to be able to incorporate Movement Primitives in the hierarchy as well, we only need to allow the agents' policy functions to produce sequences of ks instead of a single one.

Putting everything together, under this unified formalism an agent is defined by a tuple (Π, β) with

$$k_0^{l\to l+1}, \ldots, k_{n-1}^{l\to l+1} = \Pi\left(s^l, k_j^{l-1\to l}\right) \tag{6.6}$$

a sequence of n task descriptors passed to the level $l+1$ with the intent of solving the chief's task at position j in the sequence of the chief's tasks. The topology of the space $\mathscr{K}^{l\to l+1}$ determines the nature of the child agents in the brotherhood at level $l+1$.

6.3 The Hierarchization of Motor Skills in Infants

The adoption of the upright gait by primitive men reshaped the pelvis and narrowed the birth canal, in turn limiting the size of the babies head at birth. The evolutionary answer to this is to let the human brain finish development outside of the womb. As a consequence, contrary to most other species, newborns are unable to coordinate their body. Most quadrupeds start walking after only 30 minutes to one hour after birth while human babies are born with only few reflexes like sucking, rooting, a palmar grasp reflex, or a walking reflex triggered when the feet palm is in contact with a flat surface.

The dominant theory of the last century about motor development, the Neural Maturationist Theory, supposed motor skills as a continuation of existing reflexes. However, more recent scientific evidences disproved that theory, showing that the cortex is already involved in the modulation of motor behaviours at the fetal age. Today, 2 main theories try to describe the acquisition of motor skills: the Dynamic Systems Theory (DST) and the Neuronal Group Selection Theory (NGST). Both have in common that they assume phases of transition during the development and underline the importance of the experiences and the context.

Let us have a closer look in particular at the acquisition of the reaching skill by newborns. It starts already inside the mother's womb, at around 10-12 weeks post-menstrual age (PMA) with the apparition of hand-face contact. We observe that with increasing fetal age, the velocity of approach of the hand is tuned according to the sensitivity of the target location (eyes or mouth). From the first weeks after the birth, the precision of the movements is refined as the child is already able to make arm movements in response to an object, especially when fixating the object while in a sitting position. These movements called *prereaches* are not always directed towards the object and consist of oscillating or flapping movements, or movements that bring the hand to the mouth [140]. At around 3 months, these prereaches are more frequent and between 3 and 5 months their precision is further refined. The rate of successful reaches increases, finally resulting in successful grasping of objects. Reaches

are now improved through trial and error. The first successful reaching movements are still relatively imprecise and are diverse in trajectories. The reaching skill repertoire contains movements straightforwardly directed to the object, but also others that are composed of multiple sub-movements, called movement units [144]. Movement units can be isolated according to peaks in the velocity of the hand. They can be used to better track the evolution of the reaching competence. At 4 months, reaches can be decomposed into 3 to 7 movement units. This value decreases to 2.5 to 4 at an age of 6 months.

These observations about the progressive refinement of the reaching movement give a picture of the learning processes taking place in young infants. The Neuronal Group Selection Theory explains well this learning processes. It describes 2 phases in the learning of motor skills:

- The first is the so-called primary phase, during which the fetus or newborn shows various motor behaviors. They are caused by the spontaneous activity of the nervous system randomly triggering *motor options*, or *movement primitives* from a repertoire. At that stage, behaviors are already very diverse but have a limited capacity to adapt.

  In the frame of the learning of reaching, this explains the *prereaching* movements.

- During the secondary phase, infants learn to contextually adapt by selecting the appropriate *motor option* / *movement primitive* from the repertoire and learn to improve it. It is reported that from the first year on, learning is particularly effective in play with others, supposedly due to neural mirroring capacities.

  This second phase explains how children progressively achieve faster and more accurate reaches between 4 and 6 months.

There are multiple ways in which the observations from the NGS Theory can be interpreted in the frame of HRL.

First it is important to note that learning in infants must follow a precise curriculum. A new skill is learned only when the simpler skills it relies on are *sufficiently* developed. It is the role of the caregiver to present tasks to the child that match its competence level. We also reported in Chapter 4 Intrinsic Motivations based on competence improvement (CB-IMs) that may play the role of intrinsically forming a curriculum.

Next, when a task candidate for learning has been triggered, the child's first attempts resemble very crude and random movements. As the child progresses, the movements get more assured and the variability decreases in orders of magnitude. The simplest interpretation using the HRL framework is that the random exploration required for RL learning is following a precise scheduling. At first, the exploration in the set $\mathscr{K}$ is located in the parts of the

space that induces the most variability on the behavior. Following the rate of improvement of the skill, the location at which exploratory noise is applied changes towards a region which induces less variability on the behavior. Finally, noise is only applied on regions that control very detailed aspects of the behavior, allowing precise fine tuning of the skill. Typically, if we assume for example $\mathscr{K} = \{0,1\} \times \mathbb{R}^3$, exploration would first be made on the discrete component of the space (i.e. $\{0,1\}$). Then the entropy of the noise would progressively decrease, as the entropy of the noise on the other component of the space (i.e. $\mathbb{R}^3$) increases. Finally, as the skill is mastered, the total entropy decreases and no exploratory noise is applied anymore.

In the reminder of this chapter, we will present our approach to learning movement primitives. We will adopt an incremental approach, starting from a simple implementation of the TD3 algorithm [31] and will then incrementally incorporate changes to this implementation that will enable learning of movement primitives.

6.4 Methods

6.4.1 Traditional Reinforcement Learning with TD3

Let us consider a goal-based Markov Decision Problem determined by a state space S, an action space A, a goal space G, an initial state distribution $p_0(s)$, a state transition probability function $p(s'|s,a)$ and a reward probability function $r(s,s',g)$ with $g \in G$ and a discount factor γ.

The goal of Reinforcement Learning is to find a policy function $\pi(s,g)$ that maximizes the objective

$$R_\gamma^\pi = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, s_{t+1}, g)\right] \tag{6.7}$$

for a sequence of states $\{s_t\}_{t\in\mathbb{N}}$ with $s_0 \sim p_0$ and the following states are obtained by following the policy π.

Let us now introduce the notations that we will use for the TD3 algorithm [31]. TD3, like other RL algorithms, relies on target networks which we will denote with a *prime*. The policy network will be written $\pi_\theta(s,g)$ and the 2 critics $Q_{w_0}(s,g,a)$ and $Q_{w_1}(s,g,a)$.

The update rules for the target networks are given by

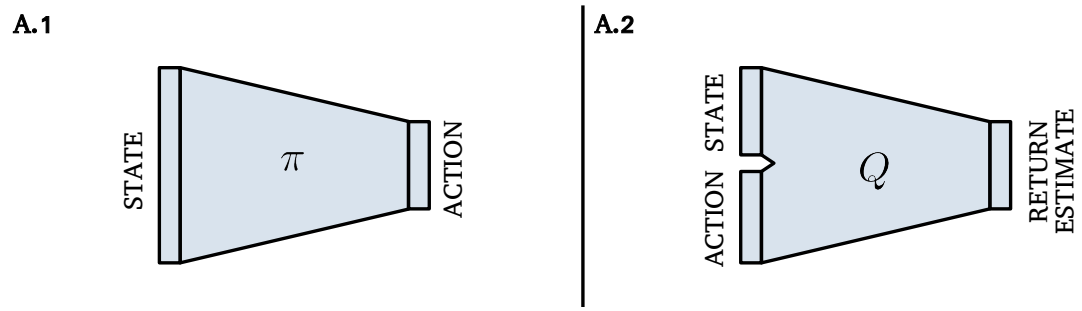


Fig. 6.1 Illustration of the *actor* and *critic* networks in the case of traditional actor-critic RL (i.e. the `default` approach). Note that in TD3, there are 2 separate critics, plus copies of each networks, called *target networks* whose weights geometrically follow that of the firsts, which are not depicted here.

$$w'_0 = \tau w_0 + (1-\tau)\, w'_0 \text{ and} \tag{6.8}$$
$$w'_1 = \tau w_1 + (1-\tau)\, w'_1 \text{ for the critics and} \tag{6.9}$$
$$\theta' = \tau\theta + (1-\tau)\, \theta' \text{ for the policy.} \tag{6.10}$$

The policy weights θ are then simply trained using the Adam algorithm in order to maximize the objective

$$O_\pi = \frac{1}{2} \sum_{i \in \{0,1\}} \mathbb{E}\left[Q_{w_i}\left(s, g, \pi_\theta\left(s, g\right)\right)\right] \tag{6.11}$$

.

In the original TD3 publication [31], the critic networks are trained to minimize the squared, minimum, target 1-step TD-error

$$\delta_i^2 = \left(\tilde{R} - Q_{w_i}\left(s, g, a\right)\right)^2 \text{ where} \tag{6.12}$$
$$\tilde{R} = r + \gamma \min_j Q_{w'_j}\left(s', g, \pi_{\theta'}\left(s', g\right)\right) \text{ is the 1-step target return.} \tag{6.13}$$

For more details about the TD3 algorithm, please refer to the section 2.2.5. Figure 6.1 shows the Actor and Critic networks as used by the TD3 algorithm.

In the remainder of this Chapter, we will refer to the experiments using the plain TD3 algorithm with no other modifications as the `default` experiment, as opposed to the `AS`, `BN` and `H-TD3` experiments detailed below.

6.4.2 Learning Simple MPs with TD3

Next, in order to learn MPs, we propose that the policy directly outputs trajectories instead of single actions. This is done by changing the dimensionality of the action space, and interpreting the actions returned by the policy as a sequence of velocities

$$a = v_0, v_1, \ldots, v_{n-1} = \pi_\theta(s, g) \tag{6.14}$$

.

We call this approach the Action Sequence (`AS`) approach (see Fig. 6.2). Of course, such a dramatic increase of the dimensionality of the action space exposes the algorithm to the problem of the *curse of dimensionality*. In order to partly counteract this issue, we reformulated the critic such that it evaluates the value of the intermediate states s_i visited during the execution of the MP and of the corresponding intermediate velocities v_i rather than evaluating only (s_0, a). In this redefinition of the critic we set it to operate at the same frequency as the simulation. Thus the loss for the critic becomes

$$\delta_j^2 = \sum_{i<n} \left(\tilde{R}_i - Q_{w_j}(s_i, g, v_i)\right)^2 \text{ where} \tag{6.15}$$

$$\tilde{R}_i = r_i + \gamma \min_j Q_{w'_j}\left(s_{i+1}, g, v'_{i+1}\right) \text{ is the 1-step target return with} \tag{6.16}$$

$$a' = v'_0, v'_1, \ldots, v'_{n-1} = \pi_{\theta'}(s, g) \tag{6.17}$$

and the loss for the policy becomes

$$O_\pi = \frac{1}{2} \sum_{j \in \{0,1\}} \sum_{i<n} \mathbb{E}\left[Q_{w_j}(s_i, g, v_i)\right] \tag{6.18}$$

.

6.4.3 Improving on the 1-step TD Update Rule

TD3 [31] is itself based on the DDPG algorithm [71]. In the original DDPG publication, the authors proposed to generate exploratory noise from an Ornstein Uhlenbeck random process. In the subsequent TD3 publication, the authors went back to the more traditional normally distributed exploratory noise.

Here, we propose to use a slightly different exploration scheme by composing the exploratory noise's gaussian distribution with a Bernouilli distribution of parameter p_{explore},

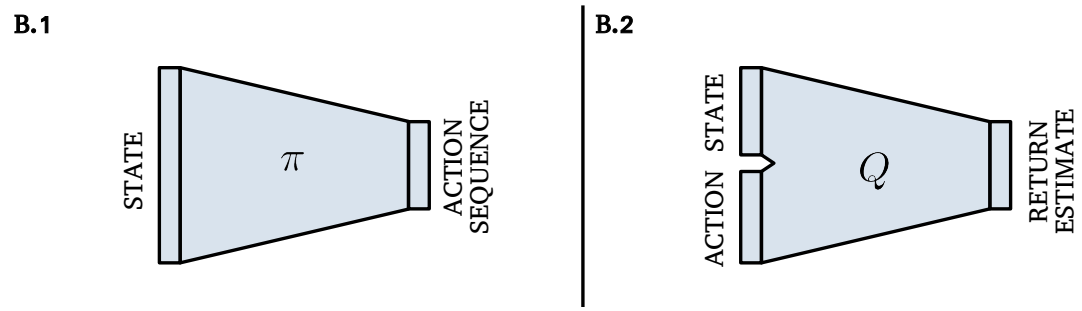


Fig. 6.2 Illustration of trajectory based RL (`AS` approach). The actor directly outputs sequences of actions instead of single actions. The critic however still operates on single actions, implying that it *sees* the intermediate states visited during the execution of the action sequence.

meaning that the agent does not explore at every iteration. In our implementation, the noise is thus defined

$$n = \begin{cases} 0 & \text{with probability } 1 - p_{\text{explore}} \\ n \sim \mathcal{N}\left(0, \sigma_{\text{explore}}\right) & \text{with probability } p_{\text{explore}} \end{cases} \tag{6.19}$$

.

Using this exploration scheme leverages the use of the 1-step TD update rule, which, as shown in both the DDPG and TD3 publications, is the main cause of instability in the training of the critic, and is the reason why target networks have been introduced. Indeed, the critic network's purpose is to approximate the discounted sum of future rewards, starting by taking action a in state s, and following the policy π thereafter. The most accurate and quickest way of estimating this quantity is to empirically measure it, by sampling non-exploratory experiences from the MDP, or in other words, to perform Monte-Carlo estimates (see section 2.2.3). Exploratory noise is however required in order to efficiently discover improvements of the behavior, preventing Monte-Carlo estimates and making the approximation of the true return more complex. The solution found is Temporal Difference learning (see section 2.2.3). In particular, 1-step TD learning exploits the Bellman equation (eq. 2.7) in order to bootstrap the return estimate from the estimate of the return in the next iteration. As shown in section 2.2.3, it is possible to obtain more accurate estimates when bootstraping the return estimate with an estimate of the return n iterations after the current iteration, given that no exploratory noise is applied during these n steps. This approach is intermediate between 1-step TD and Monte-Carlo estimates.

Our new exploration scheme implies that the agent performs a fixed portion of non-exploring steps, enabling to obtain more precise estimates of the true return. In our proposed improved version, which we call "ada-step TD", the critic target is given by

$$\tilde{R}_i = r_i + \alpha_i \gamma \min_j Q_{w'_j}\left(s', g, \pi_{\theta'}\left(s', g\right)\right) + \bar{\alpha}_i \gamma \tilde{R}_{i+1} \tag{6.20}$$

with $\alpha_i \sim \mathscr{B}\left(p_{\text{explore}}\right)$ the (binary) variable sampled from the Bernoulli distribution describing if the agent explored at iteration i and $\bar{\alpha}_i = 1 - \alpha_i$, meaning that for exploratory steps, the update rule is the 1-step TD update rule (eq. 6.13 and 6.16) else a Monte-Carlo-like accumulation of the rewards is made.

In the Results Section 6.6, we will present the advantage of using the ada-step TD update rule over the 1-step TD for the `default` and `AS` architecture. For the `BN` and `H-TD3` approaches presented below, we used the ada-step update rule only.

6.4.4 Expressing the MPs in a Lower Dimensional Space

Next, as depicted in Fig. 6.3, we propose to add a bottleneck on the policy network with the intent of performing exploration in the resulting latent space, rather than on the very high-dimensional trajectory space. The hope is that the resulting low dimensional manifold of trajectories contains meaningful movement primitives and that exploring in the space of these movement primitives results in faster discovery of better behaviors.

We will denote with the subscript H the part of the policy network before the bottleneck and with L the part of the policy network after the bottleneck.

$$\pi_\theta\left(s, g\right) = \pi_{\theta_L}\left(z\right) \text{ with} \tag{6.21}$$

$$z = \pi_{\theta_H}\left(s, g\right) \tag{6.22}$$

We will denote d_z the dimensionality of the bottleneck.

The bottlenecked experiments are referred to as `BN`.

In order to perform exploration at the level of the bottleneck, we propose to compose the exploratory noise distribution with another Bernoulli distribution of parameter $p_{\text{explore},H}$ describing whether the exploration is performed at the bottleneck or on the trajectories.

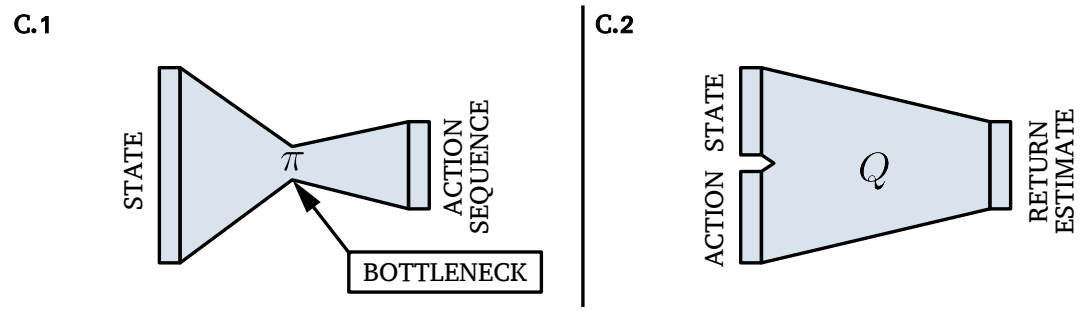


Fig. 6.3 Illustration of the bottleneck applied on the policy network (`BN` approach). Generating action sequences rather than plain actions results in a big increase of the dimensionality of the policy's output, which is problematic due to the curse of dimensionality. Applying a bottleneck on the policy network guaranties that the action sequences are living on a manifold of lower dimensionality. The noise applied on the action sequence however is still living in a high dimensional space.

As a results, with probability $p_{\text{explore},H}$, the noise

$$n_H = \begin{cases} 0 & \text{with probability } 1 - p_{\text{explore}} \\ n \sim \mathcal{N}\left(0, \sigma_{\text{explore},H}\right) & \text{with probability } p_{\text{explore}} \end{cases} \tag{6.23}$$

is applied on z otherwise the noise

$$n_L = \begin{cases} 0 & \text{with probability } 1 - p_{\text{explore}} \\ n \sim \mathcal{N}\left(0, \sigma_{\text{explore},L}\right) & \text{with probability } p_{\text{explore}} \end{cases} \tag{6.24}$$

is applied on a.

6.4.5 Forming a Hierarchy by Slicing the Policy Function in 2 Parts

In order to make the approach fully hierarchical, we propose to *slice* the `BN` approach in 2 and to interpret each half as an individual TD3 algorithm. To this end, we use a second set of critics operating on the latent code z (see Fig. 6.4). Again, we'll use the subscripts H and L to differentiate between the higher level and lower level critics. In the same way as before, the cost function for the H critics is

$$\delta_{j,H}^2 = \left(\tilde{R}_H - Q_{w_{H,j}}(s,g,z)\right)^2 \text{ with} \tag{6.25}$$

$$\tilde{R}_H = r + \gamma_H \alpha \min_j Q_{w'_{H,j}}\left(s',g,z'\right) + \bar{\alpha}\tilde{R}'_H \text{ and} \tag{6.26}$$

$$z' = \pi_{\theta'_H}\left(s',g\right) \tag{6.27}$$

. The cost function for the L critics is

$$\delta_{j,L}^2 = \sum_{i<n} \left(\tilde{R}_{L,i} - Q_{w_{L,j}}(s_i, g, v_i)\right)^2 \text{ where} \tag{6.28}$$

$$\tilde{R}_{L,i} = r_i + \gamma_L \alpha_i \min_j Q_{w'_{H,j}}\left(s_{i+1}, g, v'_{i+1}\right) + \bar{\alpha}_i \tilde{R}_{L,i+1} \text{ is the target return with} \tag{6.29}$$

$$a' = v'_0, v'_1, \ldots, v'_{n-1} = \pi_{\theta'_L}\left(z'\right) \text{ the target action for the L agent and} \tag{6.30}$$

$$z' = \pi_{\theta'_H}\left(s', g\right) \text{ the target action for the H agent} \tag{6.31}$$

. Note that the two critics use different discount factors $\gamma_H = \gamma^n$ and $\gamma_L = \gamma$ to account for the fact that the higher and lower level agents operate at different timescales.

The two objectives for the policy networks become

$$O_{\pi,H} = \frac{1}{2} \sum_{j \in \{0,1\}} \mathbb{E}\left[Q_{w_{j,H}}(s, g, z)\right] \text{ with} \tag{6.32}$$

$$z = \pi_{\theta_H}(s, g) \text{ and} \tag{6.33}$$

$$O_{\pi,L} = \frac{1}{2} \sum_{j \in \{0,1\}} \sum_{i<n} \mathbb{E}\left[Q_{w_{j,L}}(s, g, v_i)\right] \text{ with} \tag{6.34}$$

$$a = v_0, v_1, \ldots, v_{n-1} = \pi_{\theta_L}(z) \tag{6.35}$$

and the objectives $O_{\pi,H}$ and $O_{\pi,L}$ are maximized with respect to the weights θ_H and θ_L respectively.

The differences with the `BN` approach are the following:

- Use of critics operating at the bottleneck
- Use two actors objectives $O_{\pi,H}$ wrt. θ_H and $O_{\pi,H}$ wrt. θ_L instead on only $O_{\pi,L}$ wrt. θ_H and θ_L

The resulting architecture, which we call `H-TD3`, is composed of 2 TD3 algorithms *chained* together (see Fig. 6.4). The first algorithm learns to use the low dimensional movement primitives that are learned by the second. As explained above, performing exploration at the abstract level is highly desirable, as it is simpler to explore in a low dimensional manifold, however performing exploration both on the higher and lower level requires extra care. Indeed, from the perspective of the higher level TD3, applying noise both on the bottleneck z and on the velocities sequence a at the same time has the apparent effect of making the world more stochastic. From the perspective of the lower level TD3 on the other hand, applying noise twice has the apparent effect of making the agent's sensors noisy.

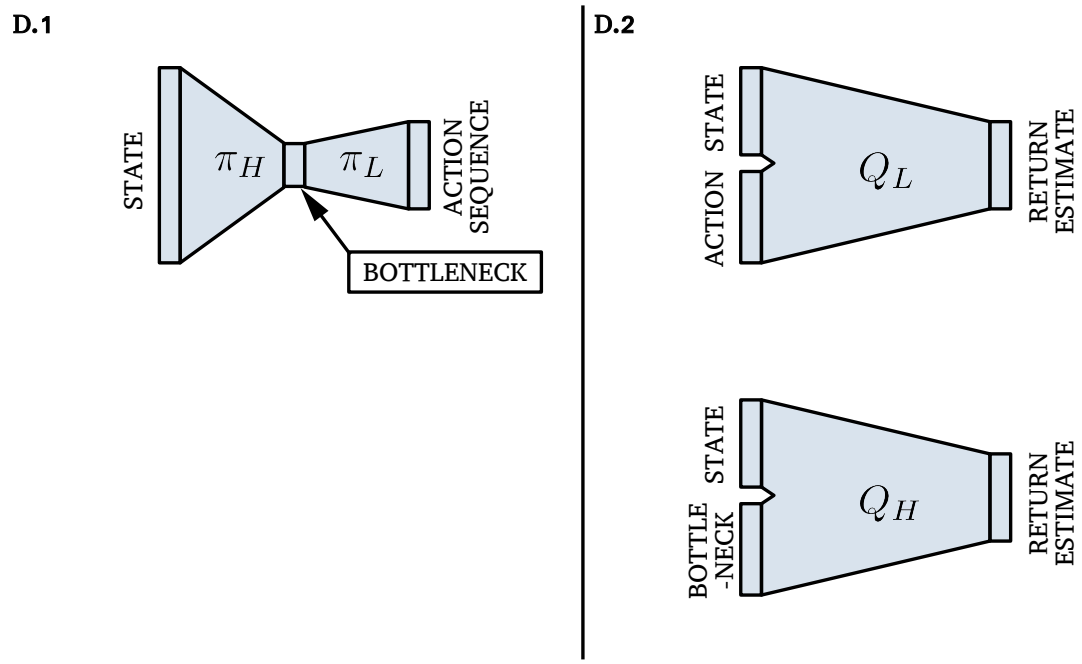


Fig. 6.4 `H-TD3` approach. In order to apply noise directly *inside* the low dimensional manifold, that is, to apply noise on the bottleneck, we need to introduce a second critic for training the policy parameters prior to the bottleneck.

Both of these disadvantages can be avoided if the exploratory noise is applied only on one of the two action spaces at a time.

Moreover, applying noise sparsely benefits the ada-step update rule introduced in section 6.4.3.

6.5 Experiment

6.5.1 The Robotic Environment

For the purpose of this experiment, we simulated a 7 DoF robotic arm within CoppeliaSim. The robot's joints are controlled in velocity mode. We used a simulation time-step $dt = 0.2\,\text{s}$. Next to the arm, we placed various actuators that it can interact with. Each actuator has an internal state (ON/OFF) which can be switched through interaction. We implemented 3 kinds of actuators that the robot can interact with: a push button, a lever that can swing left/right and a tap that can be rotated 90°. In the simplest experiment, we used only 4 push-buttons, placed in a circle around the base of the robotic arm (see Fig. 6.5).

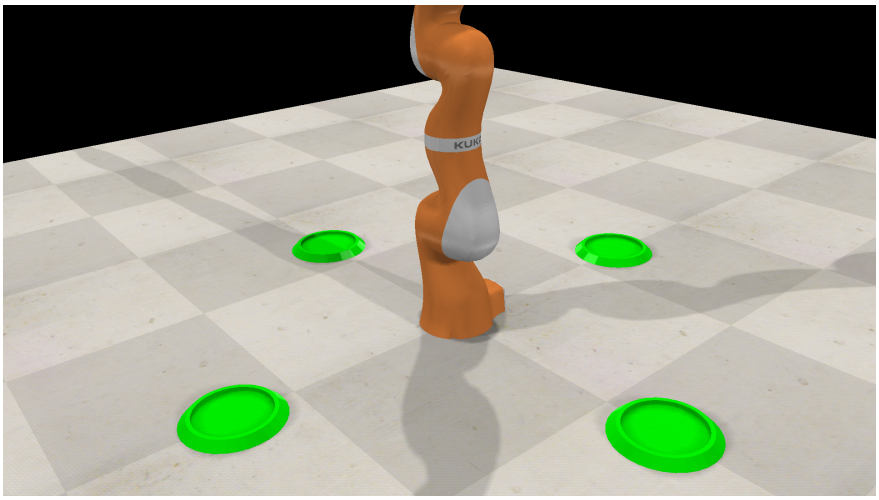


Fig. 6.5 Sample picture showing the robotic arm and 4 actuators placed around it.

In order to speed up the computations, our implementation is parallelized following a synchronous parallelization scheme. In practice, we simulate 40 robotic environments in parallel.

The internal state of each of the objects forms a binary vector which we use to define goals. In each new episode, the agent will try to actuate each of the objects such that their internal state matches with the goal internal state.

6.5.2 Training Procedure and Evaluation Metric

The training is decomposed into episodes of length 30 sec, or 150 iterations. At the beginning of each episode, the robot is reset to a random pose, the internal states of each of the actuators is randomized and a new random goal is sampled.

In the end of the episode, the data collected is placed into a replay buffer for training. If the internal state of the actuators matches with the goal that has been sampled at the beginning of the episode, the episode is considered successful.

Every 20 exploratory episodes, a non-exploratory episode is performed in order to track the evolution of the rate of successful episodes. This metric which we call the *success rate* serves as our main measure of the quality of the behavior learnt by the agent. The values reported below are the averages of 10 runs.

If we denote $\hat{g}_t$ the vector describing the state of the objects at time t and g the goal configuration, the agent's reward at time t is defined as

$$r_t = ||\hat{g}_{t-1} - g|| - ||\hat{g}_t - g|| \tag{6.36}$$

.

6.5.3 Parameters of the Model

Comparing different Reinforcement Learning models is not trivial, and there are multiple ways in which this can be done. A logical way of doing this, well adapted when comparing vastly different RL models, is to compare their peak performances, that is, to compare them on the best set of hyperparameters. This allows to use completely different network architectures, or to use very different training schemes.

Another approach is to randomize the hyperparameters, and to compare the mean and max performances. This necessitates that the distribution from which the hyperparameters are sampled match for the different architectures to be compared. This method however requires a great computation power.

Our approach in this experiment is to compare the different models on the same set of parameters. This comparison strategy is fair, in that it offers a simple way to measure the benefit brought by a single change between two models. However, it brings no guaranty that for each model the peak performance of the model is reached, and thus it does not allow to draw conclusions about the absolute performances of the models.

For all neural networks and all losses, we used a learning rate of 10^{-3}. To make the comparison fair, the total depth of the policy network when training the hierarchical approach (`H-TD3`, depth of H plus depth of L) equals the depth of the single policy network in the non-hierarchical cases (`default`; `AS`; `BN`). We used 3 layers for the high level (H) and 3 layers for the low level (L) policy networks, for a total of 6 layers. Each layer but the bottleneck (when applicable) contains 200 units. The critic networks (whether we are in the `H-TD3` case or not) are composed of 5 layers of size 500 followed by the output layer of size 1.

The TD3 algorithm depends on 2 hyperparameters: the update rate of the target networks τ which we set to 0.01 and the policy target smoothing parameter which we set to the very low value of 0.01.

For the non-hierarchical cases (`default`; `AS`; `BN`), the standard deviation of the gaussian distribution from which the exploratory noise is sampled σ_{explore} was set to 0.5. For the hierarchical case (`H-TD3`), we set $\sigma_{\text{explore},H}$ to 0.2 and $\sigma_{\text{explore},L}$ to 0.5. In all cases, the probability of exploring p_{explore} was set to 0.1.

The discount factor was set to 0.836 sec^{-1} (for the H agent in the case `H-TD3`), or 0.965 it^{-1}.

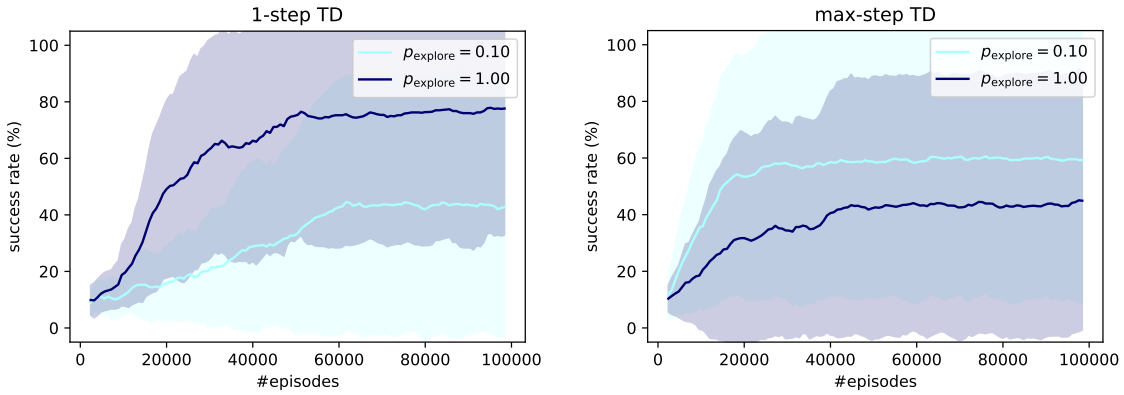


Fig. 6.6 Evaluation of the success rate of the `default` agent throughout training, for 2 values of the p_{explore} parameter, and for 2 update rules (1-step TD and ada-step TD).

For training, batches of size 256 were sampled from the replay buffer which has a size of 100000. The training is designed such that on average, each data sample collected from the environment is used 8 times to train the critic, and 4 times to train the actor.

6.6 Results

Fig. 6.6 shows the result of the `default` experiment in 2 conditions and for varying values of the p_{explore} parameter. The left plot shows the performances when using the 1-step TD update rule, and the right plot shows the performances of the ada-step update rule. The shaded areas in Fig. 6.6 (and all other figures in this section) indicate the standard deviation computed over 10 repetitions.

With the default set of parameters, which enables us to compare the different architectures, we found that around 45 to 50 % of the `default` experiments do not result in the improvement of the agent's success rate over the course of training. We observe that during these failed experiments, the policy network's output tends to diverge to extreme velocity commands early on, which results in the robotic arm hitting the limit position on many of its joints. The exploratory noise applied on the policy is not sufficient to counteract the extreme values outputted by the policy and as a consequence, the agent never experiences any reward, resulting in no learning.

We found that this issue can be resolved simply by decreasing the depth of the policy network from 6 layers down to 3 layers. By reducing the number of parameters in the network, we also reduce the magnitude of the effect of the Adam weight update [55] on the network, notably in the beginning of training, which helps maintaining the policy's output in a

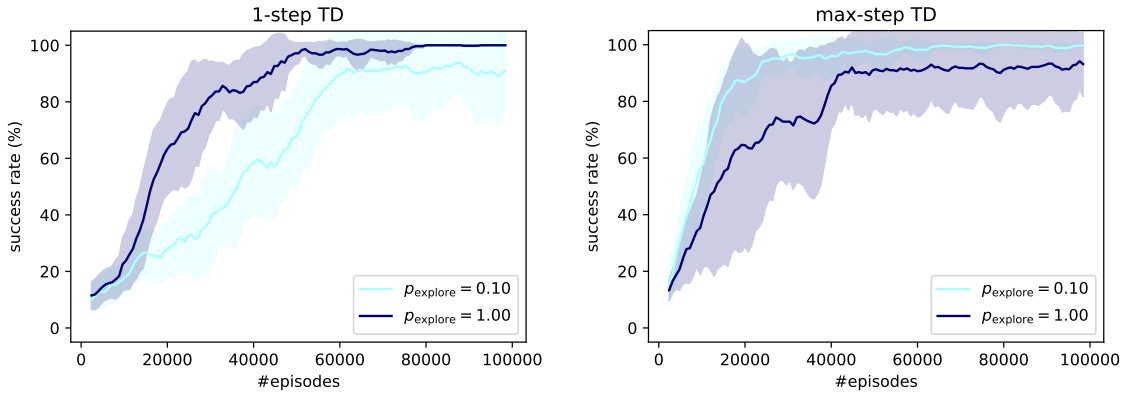


Fig. 6.7 Evaluation of the success rate of the `default` agent throughout training, for 2 values of the p_{explore} parameter, and for 2 update rules (1-step TD and ada-step TD). Out of the 10 repetitions, only the experiments where the average success rate over 100000 episodes is greater than 10 are taken into account.

reasonable range. A similar outcome is obtained by lowering the actor's learning rate, which results in the successful training of an agent with a policy network of depth 6. Our choice of the value 6 for the depth of the network is such that the `default` and `AS` architectures are comparable with the `BN` and `H-TD3` architectures which require more depth. The results of the experiments with a shallower network or with a lower policy learning rate are available in the Appendix (Figs. C.1 and C.2) and are reported in Tab. 6.1.

Fig. 6.7 shows the same result as Fig. 6.6 after filtering out the experiments which failed. An experiment was considered failed if the average success rate over 100000 episodes did not exceed 10%. The results show the advantage of the ada-step update rule over the 1-step TD when in conjunction with a sparse exploratory noise. Interestingly, in the 1-step TD case, a ten-fold decrease of the amount of noise applied on the policy for exploration results only in a light decrease of the overall performances.

Next, Fig. 6.8 shows the result of learning actions sequences `AS` rather than single actions. The results are presented for multiple lengths of sequences n. When $n = 1$, the architecture is equivalent to the `default` architecture. We chose to keep the p_{explore} parameter at a value of 0.1 for these experiments and the next.

The results show that learning action sequences is possible. Moreover, it seems to really benefit from the ada-step update rule. The problem described earlier for the `default` approach, where actions produced by the policy were biased toward extreme values, is not a problem anymore in the case where the actor outputs action sequences `AS`. In the `default` case, unlucky runs would learn to output extreme velocities in the first few episodes of

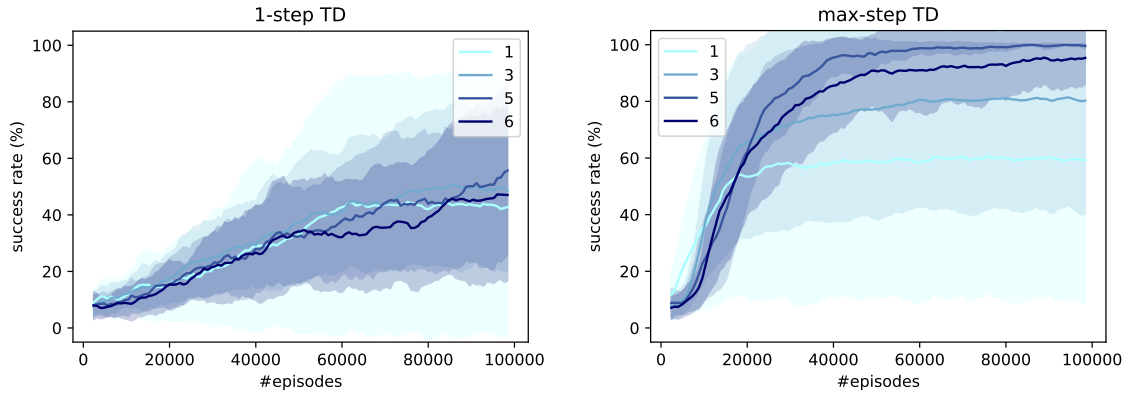


Fig. 6.8 Evaluation of the success rate of the AS agent throughout training for different sequence lengths and for 2 update rules (1-step TD and ada-step TD).

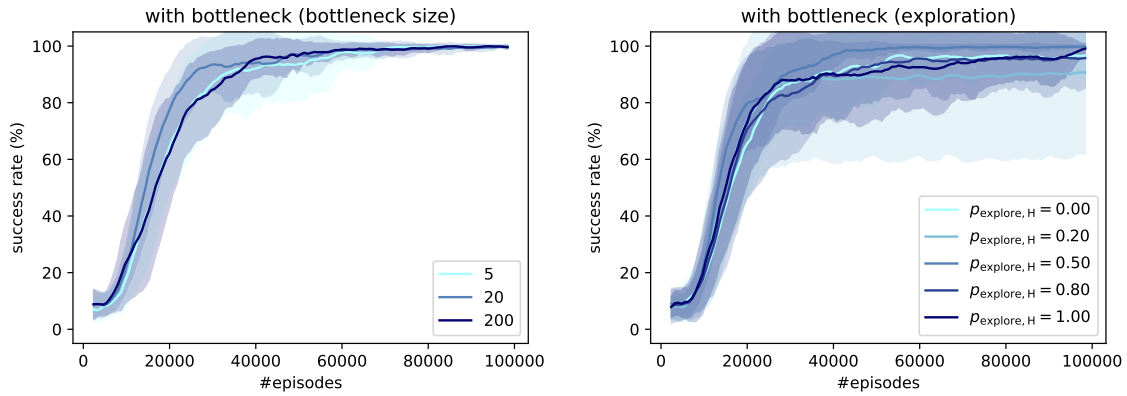


Fig. 6.9 Evaluation of the success rate of the agent throughout training, for the BN approach when varying the bottleneck size (left) or the probability to explore on the bottleneck (right).

training independently of the input presented to the policy network. When repeated for a few consecutive iterations, these actions result in the arm curling up on itself. In the AS case however, since each action is composed of multiple velocities, the application of an action sequence composed of extreme values results in a vibrating movement and the arm does not curl up on itself, thus not preventing the discovery of rewards.

Next, Fig. 6.9 shows the results for the BN experiments. For these experiments, we used an action sequence size $n = 5$ (1 second of movement per sequence) and we leveraged the 1-step TD update rule.

The left subfigure shows the effect of introducing a bottleneck, while still exploring only at the level of the velocities sequences. The case where $|z| = 200$ corresponds to

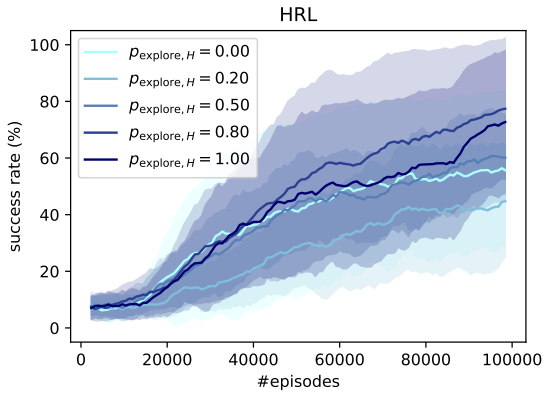


Fig. 6.10 Evaluation of the success rate of the agent throughout training, for the `H-TD3` approach.

using no bottleneck. The graph shows that the performances are essentially not affected by the presence of the bottleneck. The action sequence of dimentionality $7 \times 5 = 35$ can be compressed by a factor 7, down to a dimensionality of 5.

The right subfigure shows the effect of varying the proportion of noise applied on the bottleneck vs. on the velocity sequences, using a bottleneck size $|z| = 5$. The results show that there is a small but significant advantage in performing exploration on the abstract z space. Interestingly, the algorithm even still performs well when exploring only on the z space (i.e. when $p_{\text{explore},H} = 1.00$).

Finally, Fig. 6.10 shows the result of the `H-TD3` experiment for different values of the $p_{\text{explore},H}$ parameter, describing the probability that the noise is applied on the bottleneck. We used a bottleneck of size $|z| = 5$. The results for this experiment and for all others are also reported in the table 6.1. Although worst than for the `BN` approach, here too the results show that there is an advantage in performing exploration on the abstract z space.

It is important to note that, as pointed out in section 6.5.3, comparing different RL architectures is not trivial. In the present case in particular, the `H-TD3` architecture is significantly different from the `default`, `AS` and `BN` approaches. Many parameters, such as the critic and actor learning rates, the discount factor or the standard deviation of the exploratory noise distribution are doubled in the `H-TD3` approach since it consists in 2 TD3 algorithms chained together. Our strategy was to try to keep the comparison fair by keeping the parameters of the `H-TD3` approach as close as possible to that of the other approaches. However, we cannot rule out that additional tuning of the parameters result in better performances, potentially matching the `BN` approach.

ARCHITECTURE	PARAMETER	VALUES				
	p_{explore}	0.1	1			
`default` (1-step TD)		38.92	33.21			
`default` (ada-step TD)		**48.49**	38.57			
`default` (1-step TD, low LR)*		49.99	58.59			
`default` (ada-step TD, low LR)*		78.80	72.20			
`default` (1-step TD, shallow)*		57.42	57.57			
`default` (ada-step TD, shallow)*		82.61	73.19			
	n	1	3	5	6	
`AS` (1-step TD)		38.92	33.92	30.61	28.73	
`AS` (ada-step TD)		48.49	69.21	**83.01**	76.07	
	$\lvert z \rvert$	5	20	200		
`BN` (bottleneck size)		80.92	**83.75**	83.01		
	$p_{\text{explore},H}$	0	0.2	0.5	0.8	1
`BN` (exploration)		79.72	77.11	**84.66**	79.19	79.94
	$p_{\text{explore},H}$	0	0.2	0.5	0.8	1
`H-TD3`		37.32	26.27	36.87	**44.98**	40.09

Table 6.1 Mean success rate over 100000 episodes for different architectures and for different parameterizations averaged over 10 runs. The experiments marked with a * use a different network architecture or different parameters and are thus excluded from the comparison.

6.7 Conclusion

The first conclusion that we can draw from the results presented above, is that the problem of instabilities inherent to Q-learning which have been identified and analysed in the original DDPG [71] and TD3 [31] publications can be efficiently circumvented, at least for robotic manipulation tasks, by performing a sparser exploration, in conjunction with the use of the ada-step update rule presented in section 6.4.3. In scenarios where sparse exploration can be employed, the target networks introduced in the DDPG algorithm become irrelevant if replaced by the more clever ada-step update rule. However the latter does not affect the overestimation bias depicted in the TD3 publication. More generally, our observations regarding the use of sparse exploration raise the question of the exploration-exploitation problem. The decision whether the agent should explore new options, or instead should follow its current best policy is a fundamental limitation of RL algorithms. Designing RL algorithms that would autonomously adapt their exploration-exploitation balance to the task remains scarcely addressed in the RL literature. [117, 3] proposed to implement 5 exploration strategies and to treat the problem of choosing the exploration strategy to follow during training as a non-stationary multi-armed bandit problem. [102] investigated the benefit of what they called *intra-episodic exploration*, a type of exploration strategy where the agent switches from an exploratory mode to a exploitative mode within episodes.

Next, our experiments about the action sequences (`AS`) reveal that it is possible to train agents with action-spaces of high dimensions (> 42) using the TD3 algorithm. The `AS` approach performs comparatively better than the `default` approach when used in conjunction with the ada-step TD update rule, a simple improvement over the 1-step TD update rule. The problem of the curse of dimensionality seem to be related to the dimensionality of the *exploration space*, rather than the dimension of the action-space. Indeed, in the task that we presented here, the most relevant property of the system that determines above anything else the acquisition of rewards is the 3 dimensional position of the end effector of the arm. Although the exploratory noise is applied on a 7 dimensional (`default`) or 35 dimensional (`AS`, `BN`, `H-TD3`) vector, the consequences of the noise on the arm can essentially be summarized as the delta position of the end effector of the arm, which lives in 3 dimensions. Increasing the dimensionality of the action space by interpreting the action as velocity sequences did not result in an *effective* increase of the dimensionality of the *exploration space*. This observation suggests that it must be possible to learn in action-spaces of even higher dimensions, given that the dimension of the *exploration space* remains low. Let's assume for example that a complete humanoid robot consists in 100 controllable joints. Exploring simultaneously on the 100 joints won't produce any meaningful, good exploratory attempts. A better strategy would be to explore one limb at a time ($7 - 10$ dimensions), or

even only on a couple of well chosen joints at a time ($2-3$ dimensions). This effectively reduces the dimensionality of the *exploration space*. This is also very related to the Neural Group Selection Theory (NGST) briefly introduced in section 6.3. A similar exploration scheme is observed in infants: when presented a stimulus, newborns and 1 month old toddlers react by moving simultaneously both arms and legs. At around 1.5 month old, they learn to differentiate between the upper and lower body and at the age of 3 months, they differentiate between the left and right arm, moving preferentially the arm the closest to the stimulus.

In our third and fourth sets of experiments, we found that the velocity sequences can be highly compressed using a bottleneck, and that performing exploration in the resulting abstract space speeds up training. However, our specific `H-TD3` implementation of abstract exploration does not outperform the simpler `BN` approach.

Chapter 7

Conclusion

In this thesis, we tried to answer the question of the acquisition of elementary motor skills - like for example the development of active vision, or the acquisition of the reaching skill - by taking inspiration from developmental psychology and by using artificial neural networks, a technique rooted in computational neurosciences.

In Chapter 2, we first introduced the Reinforcement Learning task, and the key concepts associated to it. Next, in order to illustrate the relation between learning in silico and learning in vivo, we pointed out evidences that these key concepts are analogue to mechanisms taking place in the brain. More specifically, we reported that the Temporal Difference Error - a value involved in TD-RL algorithms - has been measured in the prefrontal cortex of mammals. Then, we introduced Deep Q-learning and TD3, the RL algorithms that we used is the remainder of the thesis, by fitting them into the broader families of algorithms they belong to. Finally, we reviewed existing works about Reinforcement Learning methods applied to robotics. The analysis of the literature shows that the promise of versatile robotic arms has not yet been implemented. As of 2021, it seems that RL methods applied to robotic manipulation do not compete well against the long established engineered approaches used in the industry. The main obstacle is probably that by essence, behaviors learned through RL-based methods are prone to colliding with the environment (especially during training, but also during execution) while today's robotic arms are not designed for collisions. Next, combining together and orchestrating together multiple skills learned through RL also remains an open question. Finally, robots often don't have access to any haptic feedback, which might play a crucial role in the learning of dexterous manipulation. Although RL did not meet a franc success when applied to robotic manipulation, recent advances have enabled researchers for the first time to successfully achieve robotic locomotion on 2 or 4 legs, without using LiDAR cameras. The locomotion task is much more adapted to today's RL-methods, in that walking constitutes a single skill to be learned (as opposed to manipulation which is

a repertoire composed of many skills). Moreover, walking requires online adaptation to unexpected perturbations, and a short to middle time-scale anticipation ability, for which RL has displayed good performances. The locomotion task is further facilitated by the use of modern high efficiency brushless motors in conjunction with backdrivable low gear-ratio gearboxes giving a natural *springness* to robotic legs and making them very compliant. It is likely that the next scientific breakthroughs in the field of RL applied to robotic manipulation are conditioned by the engineering of hardware both mechanically compliant and resilient to collisions, that would ideally also integrate haptic sensors.

Besides RL methods and their applications, we addressed the topic of representation learning in Chapter 3. More specifically, we compared qualitatively and quantitatively two different ways of learning abstract representations, which are different in nature and serve different purposes. Our study is conducted with the intent of integrating together sensory information from different modalities. Our multimodal integration approach naturally brings the question of measuring the information that is shared between the modalities, and thus gives an information theoretic perspective on Autoencoders. Speaking informally, we showed that in an information flux, not all *bits of information* are equally compressed, and that Autoencoders have a clear tendency to retain in the learned representation the part of the information that is the least compressed. Finally, we argued that meaningful Intrinsic Motivations might be obtained from the learning of a joint representation of multimodal sensory information. More precisely, we argued that these IMs hold the potential of driving the learning of sensory-motor contingencies.

Intrinsic Motivations are in the computational literature what approaches best the modeling of primitive emotions. In Chapter 4, we proposed a short review of Intrinsic Motivations. Multiple taxonomies have been proposed to try and classify the different IMs. In particular, Gianlucca Baldassare's classification identifies 3 main classes: NB-IM, PB-IM and CB-IM. NB-IMs and PB-IMs find possible neuroscientific groundings, notably in the hippocampus, basal ganglia and superior colliculus. A large scale study on IMs showed a *high degree of alignment between the intrinsic curiosity objective and the hand-designed extrinsic rewards of many game environments* [20]. This supports the hypothesis that some emotions (like surprise, amusement, fear, etc) serve as general purpose rules that drive or facilitate the acquisition of new skills.

Next, in Chapter 5, we studied in particular the AEC-IM. First, following up on our work about multimodal integration, we showed that the reconstruction error of a deep autoencoder can serve as an indicator of the redundancy of the input signal. We then used this property to derive an IM that drives the acquisition of active vision, comprising the joint control of the eyes' vergence and cyclovergence angles, and the joint control of the pan and tilt angles

in order to achieve smooth pursuit of objects. The resulting agent is able to autonomously calibrate itself, learning to fixate objects in depth, and in time. Moreover, it resembles closely the human visual system in its design.

Finally, Chapter 6 addresses the problem of robotic manipulation through the angle of Hierarchical Reinforcement Learning. Like Marvin Minsky hypothesized in the 80's, the NGST neuroscientific and psychological theory about motor-development tends to indicate that the behavior is hierarchically structured. Inspiring from Minsky's ideas and from the NGST theory, we proposed a simple hierarchical RL model based on movement primitives. Our proposed model is compared with a simpler, non-hierarchical TD3 model. We found that there is an advantage in learning movement primitives, and that moreover, these primitives can be expressed in a low dimensional manifold. We also found that exploring in the resulting low dimensional manifold results in faster learning.

Overall, Reinforcement Learning has known many successes in the last decade, but it has also shown its limitations. As we have shown, HRL holds the promise of alleviating many of these obstacles however, so far, researchers failed to successfully apply HRL techniques to real world scenarios. Most researches in HRL are aiming at building hierarchies of generic agents which, like stem cells, would then specialize into a specific skill. A less ambitious approach would consists of manually defining the tasks that compose the hierarchy. This approach would enable researchers to study other aspects of artificial cognition, like for example Pavlovian mechanisms.

Consider the simple task of reaching for an object, which we would structure as the sequence of 2 separate skills *pre-reach* and *post-reach*. The reward for the *post-reach* task is defined as a sparse reward for reaching the target location within 3 cm. Now the reward for the *pre-reach* task can be defined as a sparse reward indicating whether or not the *post-reach* skill succeeded. As a result, the *pre-reach* agent is responsible for bringing the agent to a pose, from which the *post-reach* agent will manage to successfully reach the target location. In this simple example, the nature of the tasks composing the hierarchy is fixed by the experimenter, thus it is not the role of the (sub)agents to figure out which skill to learn. The interest of the experiment resides in comparing the different possible ways in which the task can be structured, in order to compare the end performance as well as the learning speed. From a neuroscientific point of view, this experiment can serve as a simple way of modeling second order rewards.

Next, let's consider the very complex problem of controlling 2 robotic arms simultaneously. It is clear that a plain RL approach is not adapted: often we would want only one arm to move at a time, and when exploring, we also want only one arm to explore at a time. A clever hierarchy that would control 2 arms should be able to execute agents *in parallel* (i.e.

to control both arms at the same time, like for example to pass an object from one hand to the other), but the controller should also learn to idle one of the 2 arms such that learning can take place on the other. Moreover, a clever hierarchization should enable transfer learning from one arm to the other. Designing a hierarchy composed only of *generic agents* for this scenario is clearly out of our reach in 2022. However comparing hierarchies composed of pre-defined tasks / skills would give researchers precious insights about how learning should be structured.

Potentially, one objective for researchers conducting such experiments could be to define the elementary tasks that define the hierarchy in a manner that allows for generalization, which would then open the way for the most distant milestone of creating hierarchies of *generic agents*.

Another objective in this line of research would be to refine the hierarchy in order to rely less and less on *oracle knowledge*, like for example the raw (x, y, z) position and speed of the objects in the environment. Ideally the hierarchy should be made such that the first skills to be learned by the agent are those that will latter enable it to locate the objects around it using nothing else than its sensors. Foreseeably, in such a refined hierarchy, most agents are Instrinsically Motivated. Some interesting work by our collaborators François De La Bourdonnaye and Céline Teulière [28] give good insights about how a such refinement can be done.

In conclusion, instead of focusing on the genericness of the agents composing a hierarchy, I advocate for concentrating efforts on the modularity of specialized agents, designed to solve a specific (sub)task. Moreover, I argue that researchers should start with a *coarse* hierarchy, at first exploiting extensively the *oracle knowledge*. Next, from an optimized, *coarse* hierarchy, it is possible to later refine it so as to learn to rely less on the ground truth knowledge. Designing Intrinsically Motivated sub-agents is a very challenging process and often imposes axes of research orthogonal to the primary goal of learning hierarchies. Finally, I believe that scientists should agree on one benchmark environment for developmental robotics, and more importantly, on a common software architectural pattern in order to make performance comparisons efficient and repeatable.

References

[1] An, J. and Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18.

[2] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR.

[3] Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. (2020a). Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR.

[4] Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. (2020b). Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*.

[5] Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

[6] Baldassarre, G. (2011). What are intrinsic motivations? a biological perspective. In *2011 IEEE international conference on development and learning (ICDL)*, volume 2, pages 1–8. IEEE.

[7] Baldassarre, G. (2019). Intrinsic motivations and open-ended learning. *arXiv preprint arXiv:1912.13263*.

[8] Baldassarre, G. and Mirolli, M. (2013). Deciding which skill to learn when: temporal-difference competence-based intrinsic motivation (td-cb-im). In *Intrinsically Motivated learning in natural and artificial systems*, pages 257–278. Springer.

[9] Barlow, H. B. et al. (1961). Possible principles underlying the transformation of sensory messages. *Sensory communication*, 1(01).

[10] Barto, A., Mirolli, M., and Baldassarre, G. (2013). Novelty or surprise? *Frontiers in psychology*, 4:907.

[11] Barto, A. G., Singh, S., and Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the 3rd International Conference on Development and Learning*, pages 112–19. Piscataway, NJ.

[12] Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. *arXiv preprint arXiv:1606.01868*.

[13] Bellman, R. (1957). Dynamic programming. *Princeton university press, Princeton*.

[14] Bengio, Y. (2017). The consciousness prior. *arXiv preprint arXiv:1709.08568*.

[15] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

[16] Bengio, Y., LeCun, Y., and Henderson, D. (1994). Globally trained handwritten word recognizer using spatial representation, convolutional neural networks, and hidden markov models. In *Advances in neural information processing systems*, pages 937–944.

[17] Bhattacharya, S., Singla, A., Dholakiya, D., Bhatnagar, S., Amrutur, B., Ghosal, A., Kolathaya, S., et al. (2019). Learning active spine behaviors for dynamic and efficient locomotion in quadruped robots. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–6. IEEE.

[18] Bradtke, S. J. and Duff, M. O. (1995). Reinforcement learning methods for continuous-time markov decision. *Advances in Neural Information Processing Systems 7*, 7:393.

[19] Brouwer, L. E. (1911). Beweis der invarianz der dimensionenzahl. *Mathematische Annalen*, 70(2):161–165.

[20] Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2018a). Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*.

[21] Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018b). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

[22] Chandrapala, T. N. and Shi, B. E. (2015). Learning slowness in a sparse model of invariant feature detection. *Neural computation*, 27(7):1496–1529.

[23] Chen, D., Zhou, B., Koltun, V., and Krähenbühl, P. (2020). Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR.

[24] Clarke, D. D. (1999). Circulation and energy metabolism of the brain. *Basic neurochemistry: Molecular, cellular, and medical aspects*.

[25] Dabney, W., Kurth-Nelson, Z., Uchida, N., Starkweather, C. K., Hassabis, D., Munos, R., and Botvinick, M. (2020). A distributional code for value in dopamine-based reinforcement learning. *Nature*, 577(7792):671–675.

[26] Daniel, C., Neumann, G., and Peters, J. (2012). Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pages 273–281.

[27] Dayan, P. and Hinton, G. (1993). Feudal reinforcement learning. nips'93 (pp. 271–278).

[28] de La Bourdonnaye, F., Teulière, C., Triesch, J., and Chateau, T. (2018). Stage-wise learning of reaching using little prior knowledge. *Frontiers in Robotics and AI*, 5:110.

[29] Eckmann, S., Klimmasch, L., Shi, B. E., and Triesch, J. (2020). Active efficient coding explains the development of binocular vision and its failure in amblyopia. *Proceedings of the National Academy of Sciences*, 117(11):6156–6162.

[30] François-Lavet, V., Bengio, Y., Precup, D., and Pineau, J. (2019). Combined reinforcement learning via abstract representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3582–3589.

[31] Fujimoto, S., Van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

[32] Gallego, J., Vani, A., Schwarzer, M., and Lacoste-Julien, S. (2019). Gait: A geometric approach to information theory. *arXiv preprint arXiv:1906.08325*.

[33] Gehring, W. J. (2005). New perspectives on eye development and the evolution of eyes and photoreceptors. *Journal of Heredity*, 96(3):171–184.

[34] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.

[35] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

[36] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE.

[37] Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838.

[38] Guenter, F., Hersch, M., Calinon, S., and Billard, A. (2007). Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics*, 21(13):1521–1544.

[39] Guo, Z. D., Azar, M. G., Saade, A., Thakoor, S., Piot, B., Pires, B. A., Valko, M., Mesnard, T., Lattimore, T., and Munos, R. (2021). Geometric entropic exploration. *arXiv preprint arXiv:2101.02055*.

[40] Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., and Levine, S. (2018a). Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*.

[41] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018b). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

[42] Hasselt, H. V. (2010). Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621.

[43] Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., et al. (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

[44] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2016). beta-vae: Learning basic visual concepts with a constrained variational framework.

[45] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[46] Huber, P. J. (1964). Robust estimation of a location parameter. *Ann. Math. Statist.*, 35(1):73–101.

[47] Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., and Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26).

[48] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[49] James, S., Freese, M., and Davison, A. J. (2019). Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*.

[50] Jauffret, A., Cuperlier, N., Gaussier, P., and Tarroux, P. (2012). Multimodal integration of visual place cells and grid cells for navigation tasks of a real robot. In *International Conference on Simulation of Adaptive Behavior*, pages 136–145. Springer.

[51] Joshi, S., Kumra, S., and Sahin, F. (2020). Robotic grasping using deep reinforcement learning. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 1461–1466. IEEE.

[52] Jung, T., Polani, D., and Stone, P. (2011). Empowerment for continuous agent—environment systems. *Adaptive Behavior*, 19(1):16–39.

[53] Kalakrishnan, M., Righetti, L., Pastor, P., and Schaal, S. (2011). Learning force control policies for compliant manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4639–4644. IEEE.

[54] Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks.

[55] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[56] Klimmasch, L., Lelais, A., Lichtenstein, A., Shi, B. E., and Triesch, J. (2017). Learning of active binocular vision in a biomechanical model of the oculomotor system. In *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 21–26.

[57] Klimmasch, L., Schneider, J., Lelais, A., Shi, B. E., and Triesch, J. (2020). The development of active binocular vision under normal and alternate rearing conditions. *bioRxiv*.

[58] Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2005). All else being equal be empowered. In *European Conference on Artificial Life*, pages 744–753. Springer.

[59] Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

[60] Kolathaya, S., Joglekar, A., Shetty, S., Dholakiya, D., Sagi, A., Bhattacharya, S., Singla, A., Bhatnagar, S., Ghosal, A., Amrutur, B., et al. (2019). Trajectory based deep policy search for quadrupedal walking. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–6. IEEE.

[61] Kroemer, O. B., Detry, R., Piater, J., and Peters, J. (2010). Combining active learning and reactive control for robot grasping. *Robotics and Autonomous systems*, 58(9):1105–1116.

[62] Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29:3675–3683.

[63] Kumar, V., Todorov, E., and Levine, S. (2016). Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383. IEEE.

[64] Kumaran, D. and Maguire, E. A. (2007). Which computational mechanisms operate in the hippocampus during novelty detection? *Hippocampus*, 17(9):735–748.

[65] Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. (2020). Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47).

[66] Lee, S., Park, M., Lee, K., and Lee, J. (2019). Scalable muscle-actuated human simulation and control. *ACM Transactions on Graphics (TOG)*, 38(4):1–13.

[67] Legendre, A. M. (1805). *Nouvelles méthodes pour la détermination des orbites des comètes*. F. Didot.

[68] Lelais, A., Mahn, J., Narayan, V., Zhang, C., Shi, B. E., and Triesch, J. (2019). Autonomous development of active binocular and motion vision through active efficient coding. *Frontiers in neurorobotics*, 13:49.

[69] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

[70] Lewicki, M. S. (2002). Efficient coding of natural sounds. *Nature neuroscience*, 5(4):356–363.

[71] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[72] Lin, H.-I. and Lai, C.-C. (2012). Learning collision-free reaching skill from primitives. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2383–2388. IEEE.

[73] Lonini, L., Forestier, S., Teulière, C., Zhao, Y., Shi, B. E., and Triesch, J. (2013a). Robust active binocular vision through intrinsically motivated learning. *Frontiers in neurorobotics*, 7:20.

[74] Lonini, L., Zhao, Y., Chandrashekhariah, P., Shi, B. E., and Triesch, J. (2013b). Autonomous learning of active multi-scale binocular vision. In *2013 IEEE third joint international conference on development and learning and epigenetic robotics (ICDL)*, pages 1–6. IEEE.

[75] Loquercio, A., Kaufmann, E., Ranftl, R., Dosovitskiy, A., Koltun, V., and Scaramuzza, D. (2019). Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14.

[76] Luo, S., Kasaei, H., and Schomaker, L. (2020). Accelerating reinforcement learning for reaching using continuous curriculum learning. *arXiv preprint arXiv:2002.02697*.

[77] Mahadevan, S., Marchalleck, N., Das, T. K., and Gosavi, A. (1997). Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 202–210. MORGAN KAUFMANN PUBLISHERS, INC.

[78] Mallat, S. G. and Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12):3397–3415.

[79] McNiel, D. B., Choi, J. S., Hessburg, J. P., and Francis, J. T. (2016). Reward value is encoded in primary somatosensory cortex and can be decoded from neural activity during performance of a psychophysical task. In *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 3064–3067. IEEE.

[80] Minsky, M. (1988). *Society of mind*. Simon and Schuster.

[81] Minsky, M. and Papert, S. A. (1969). *Perceptrons: An introduction to computational geometry*. MIT press.

[82] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

[83] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[84] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

[85] Mohamed, S. and Rezende, D. J. (2015). Variational information maximisation for intrinsically motivated reinforcement learning. *arXiv preprint arXiv:1509.08731*.

[86] Mudigonda, M., Agrawal, P., Deweese, M., and Malik, J. (2018). Investigating deep reinforcement learning for grasping objects with an anthropomorphic hand.

[87] Niv, Y. (2019). Learning task-state representations. *Nature neuroscience*, 22(10):1544–1553.

[88] Noda, K., Arie, H., Suga, Y., and Ogata, T. (2014). Multimodal integration learning of robot behavior using deep neural networks. *Robotics and Autonomous Systems*, 62(6):721–736.

[89] Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*. https://distill.pub/2017/feature-visualization.

[90] Olazaran, M. (1996). A sociological study of the official history of the perceptrons controversy. *Social Studies of Science*, 26(3):611–659.

[91] Olmos, A. and Kingdom, F. A. A. (2004). A biologically inspired algorithm for the recovery of shading and reflectance images. *Perception*, 33(12):1463–1473. PMID: 15729913.

[92] Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325.

[93] OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. (2018). Learning dexterous in-hand manipulation. *CoRR*.

[94] Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286.

[95] Paraschos, A., Daniel, C., Peters, J., Neumann, G., et al. (2013). Probabilistic movement primitives. *Advances in neural information processing systems*.

[96] Parr, R. E. (1998). *Hierarchical control and learning for Markov decision processes*. University of California, Berkeley.

[97] Patel, N., Choromanska, A., Krishnamurthy, P., and Khorrami, F. (2017). Sensor modality fusion with cnns for ugv autonomous driving in indoor environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1531–1536. IEEE.

[98] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR.

[99] Peters, J., Mülling, K., and Altun, Y. (2010). Relative entropy policy search. In *AAAI*, volume 10, pages 1607–1612. Atlanta.

[100] Peters, J. and Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

[101] Pham, T.-H., De Magistris, G., and Tachibana, R. (2018). Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6236–6243. IEEE.

[102] Pîslar, M., Szepesvari, D., Ostrovski, G., Borsa, D., and Schaul, T. (2021). When should agents explore? *arXiv preprint arXiv:2108.11811*.

[103] Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., and Ganguli, S. (2016). Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pages 3360–3368.

[104] Quillen, D., Jang, E., Nachum, O., Finn, C., Ibarz, J., and Levine, S. (2018). Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6284–6291. IEEE.

[105] Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2018). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*.

[106] Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

[107] Ramakrishnan, A., Byun, Y. W., Rand, K., Pedersen, C. E., Lebedev, M. A., and Nicolelis, M. A. (2017). Cortical neurons multiplex reward-related signals along with sensory and motor information. *Proceedings of the National Academy of Sciences*, 114(24):E4841–E4850.

[108] Ramkumar, P., Dekleva, B., Cooler, S., Miller, L., and Kording, K. (2016). Premotor and motor cortices encode reward. *PloS one*, 11(8):e0160851.

[109] Redgrave, P. and Gurney, K. (2006). The short-latency dopamine signal: a role in discovering novel actions? *Nature reviews neuroscience*, 7(12):967–975.

[110] Rezende, D. J. and Viola, F. (2018). Taming vaes.

[111] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[112] Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. cornell aeronautical laboratory. *Inc.: Buffalo, NY, USA*.

[113] Rossi, R. J. (2018). *Mathematical statistics: an introduction to likelihood based inference*. John Wiley & Sons.

[114] Santucci, V. G., Baldassarre, G., and Mirolli, M. (2013). Which is the best intrinsic motivation signal for learning multiple skills? *Frontiers in neurorobotics*, 7:22.

[115] Savinov, N., Raichuk, A., Marinier, R., Vincent, D., Pollefeys, M., Lillicrap, T., and Gelly, S. (2018). Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*.

[116] Schaal, S. (2006). Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer.

[117] Schaul, T., Borsa, D., Ding, D., Szepesvari, D., Ostrovski, G., Dabney, W., and Osindero, S. (2019). Adapting behaviour for learning progress. *arXiv preprint arXiv:1912.06910*.

[118] Schmidhuber, J. (1991a). Adaptive confidence and adaptive curiosity. In *Institut fur Informatik, Technische Universitat Munchen, Arcisstr. 21, 800 Munchen 2*. Citeseer.

[119] Schmidhuber, J. (1991b). Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463.

[120] Schmidhuber, J. (1991c). A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227.

[121] Schmidhuber, J. (2008). Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In *Workshop on anticipatory behavior in adaptive learning systems*, pages 48–76. Springer.

[122] Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247.

[123] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2019). Mastering atari, go, chess and shogi by planning with a learned model. corr abs/1911.08265 (2019). *arXiv preprint arXiv:1911.08265*.

[124] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

[125] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[126] Schultz, W., Dayan, P., and Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599.

[127] Shim, M. S., Zhao, C., Li, Y., Zhang, X., Zhang, W., and Li, P. (2019). Robust deep multi-modal sensor fusion using fusion weight regularization and target learning. *arXiv preprint arXiv:1901.10610*.

[128] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

[129] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms.

[130] Singh, S., Barto, A. G., and Chentanez, N. (2005). Intrinsically motivated reinforcement learning. Technical report, MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER SCIENCE.

[131] Singla, A., Bhattacharya, S., Dholakiya, D., Bhatnagar, S., Ghosal, A., Amrutur, B., and Kolathaya, S. (2019). Realizing learned quadruped locomotion behaviors through kinematic motion primitives. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7434–7440. IEEE.

[132] Srinivas, A., Laskin, M., and Abbeel, P. (2020). Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*.

[133] Stulp, F., Theodorou, E., Buchli, J., and Schaal, S. (2011). Learning to grasp under uncertainty. In *2011 IEEE International Conference on Robotics and Automation*, pages 5703–5708. IEEE.

[134] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An introduction*. MIT press.

[135] Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

[136] Tarigoppula, V. S. A., Choi, J. S., Hessburg, J. P., McNiel, D. B., Marsh, B. T., and Francis, J. T. (2018). Motor cortex encodes a temporal difference reinforcement learning process. *bioRxiv*, page 257337.

[137] Teulière, C., Forestier, S., Lonini, L., Zhang, C., Zhao, Y., Shi, B., and Triesch, J. (2015). Self-calibrating smooth pursuit through active efficient coding. *Robotics and Autonomous Systems*, 71:3–12.

[138] Theodorou, E., Buchli, J., and Schaal, S. (2010a). Learning policy improvements with path integrals. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 828–835.

[139] Theodorou, E., Buchli, J., and Schaal, S. (2010b). Reinforcement learning of motor skills in high dimensions: A path integral approach. In *2010 IEEE International Conference on Robotics and Automation*, pages 2397–2403. IEEE.

[140] Van der Fits, I., Klip, A., Van Eykern, L., and Hadders-Algra, M. (1999). Postural adjustments during spontaneous and goal-directed arm movements in the first half year of life. *Behavioural Brain Research*, 106(1-2):75–90.

[141] Van Hasselt, H. and Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279. IEEE.

[142] Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR.

[143] Vikram, T. N., Teulière, C., Zhang, C., Shi, B. E., and Triesch, J. (2014). Autonomous learning of smooth pursuit and vergence through active efficient coding. In *4th International Conference on Development and Learning and on Epigenetic Robotics*, pages 448–453. IEEE.

[144] von Hofsten, C. (1991). Structuring of early reaching movements: a longitudinal study. *Journal of motor behavior*, 23(4):280–292.

[145] Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., Hassabis, D., and Botvinick, M. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nature neuroscience*, 21(6):860–868.

[146] Wang, K., Kang, B., Shao, J., and Feng, J. (2020). Improving generalization in reinforcement learning with mixture regularization. *arXiv preprint arXiv:2010.10814*.

[147] Wilmot, C., Shi, B. E., and Triesch, J. (2020). Self-calibrating active binocular vision via active efficient coding with deep autoencoders. In *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 1–6. IEEE.

[148] Wilmot, C. and Triesch, J. (2021). Learning abstract representations through lossy compression of multi-modal signals. *arXiv preprint arXiv:2101.11376*.

[149] Won, J. and Lee, J. (2019). Learning body shape variation in physics-based characters. *ACM Transactions on Graphics (TOG)*, 38(6):1–12.

[150] Xie, Z., Berseth, G., Clary, P., Hurst, J., and van de Panne, M. (2018). Feedback control for cassie with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1241–1246. IEEE.

[151] Xie, Z., Clary, P., Dao, J., Morais, P., Hurst, J., and Panne, M. (2020). Learning locomotion skills for cassie: Iterative design and sim-to-real. In *Conference on Robot Learning*, pages 317–329.

[152] Xie, Z., Clary, P., Dao, J., Morais, P., Hurst, J., and van de Panne, M. (2019). Iterative reinforcement learning based design of dynamic locomotion skills for cassie. *arXiv preprint arXiv:1903.09537*.

[153] Yu, W., Kumar, V. C., Turk, G., and Liu, C. K. (2019). Sim-to-real transfer for biped locomotion. *arXiv preprint arXiv:1903.01390*.

[154] Zeng, A., Song, S., Welker, S., Lee, J., Rodriguez, A., and Funkhouser, T. (2018). 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. IEEE.

[155] Zhang, C., Triesch, J., and Shi, B. E. (2016). An active-efficient-coding model of optokinetic nystagmus. *Journal of vision*, 16(14):10–10.

[156] Zhang, C., Zhao, Y., Triesch, J., and Shi, B. E. (2014). Intrinsically motivated learning of visual motion perception and smooth pursuit. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1902–1908. IEEE.

[157] Zhao, Y., Rothkopf, C. A., Triesch, J., and Shi, B. E. (2012a). A unified model of the joint development of disparity selectivity and vergence control. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–6. IEEE.

[158] Zhao, Y., Rothkopf, C. A., Triesch, J., and Shi, B. E. (2012b). A unified model of the joint development of disparity selectivity and vergence control. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–6. IEEE.

[159] Zhu, Q., Triesch, J., and Shi, B. E. (2017). Autonomous, self-calibrating binocular vision based on learned attention and active efficient coding. In *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 27–32.

[160] Zhu, Q., Triesch, J., and Shi, B. E. (2017). Joint learning of binocularly driven saccades and vergence by active efficient coding. *Frontiers in neurorobotics*, 11:58.

[161] Zhu, Q., Triesch, J., and Shi, B. E. (2020). Integration of vergence, cyclovergence, and saccades through active efficient coding. In *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 1–6.

[162] Zhu, Q., Zhang, C., Triesch, J., and Shi, B. E. (2018). Autonomous learning of cyclovergence control based on active efficient coding. In *2018 Joint IEEE 8th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 251–256.

Appendix A

Overview of Learning Algorithm for Robotic Control

Publication	year	Real Robot	Algorithm	Task	Control mode	Reward
[38]	2007	yes	NAC	Reaching	Gaussian Mixture Model	Dense reward based on proximity to target object
[76]	2020	yes	DDPG / PCCL-DDPG	Reaching	delta joint angle	Comparison of dense and sparse rawards
[101]	2018	yes	OptLayer (based on TRPO)	Reaching	delta joint angle	Multiple reward shaping strategies : dense reward based on the distance to the target, penalization for collision, bonus when end-effector very close to target

[28]	2018	yes	DDPG	Reaching	delta joint angle	Comparison between a dense reward based on the proximity to the target object, a sparse reward and a shaped reward
[36]	2016	yes	NAF	Reaching	Velocity control	Dense reward based on the proximity to the target + metabolic cost
[36]	2016	yes	NAF	Pick-place	Velocity control	Dense reward based on the proximity of the end effector to the door handle + door opening angle
[53]	2011	yes	PI²	Pick-place	DMP	Sparse reward upon success
[69]	2016	yes	BADMM	Pick-place	Torque control	Dense reward based on the distance to the target + metabolic cost
[133]	2011	yes	PI²	Grasp	DMP	Sparce reward upon successful grasp
[104]	2018	no	multiple	Grasp	End-effector velocities	Sparce reward upon successful grasp
[51]	2020	yes	DQN, DDQN	Grasp	Delta end-effector position	Reward of 10 upon grasp + reward of 1 uppon contact with the object
[86]	2018	no	TRPO	Grasp	Position control	Dense reward based on the distance between the palm and the object + sparse term based on object distance from the table
[61]	2010	yes	CGB	Grasp	DMP	NA

[154]	2018	yes	DQN	Grasp	IK based movement primitives	Reward of 1 for successful grasps, and 0.5 for pushes that make detectable changes to the environment
[105]	2018	no	NPG, DAPG	In-hand	Position control	Comparison of sparse and shaped reward
[63]	2016	yes	Custom	In-hand	Custom	Distance between the hand position and the desired hand position + distance between the object pose and the desired object pose
[65]	2020	yes	TRPO	Quad	Position PD in the reference frame of a predefined motion	Linear velocity reward + angular velocity reward + base motion reward + foot clearance reward + body collision reward + foot motion smoothness reward + metabolic cost (see supplementary material s4)
[139]	2010	yes	PI²	Quad	DMP	Jumped distance + keeping roll and yaw angles
[40]	2018	yes	SAC + entropy maximization	Quad	Position PD	Walking distance + metabolic cost
[17]	2019	yes	PPO	Quad	Position PD	Desired velocity + metabolic cost
[131]	2019	yes	PPO	Quad	Position PD	Desired velocity + metabolic cost
[60]	2019	yes	PPO	Quad	Position PD	Desired velocity + metabolic cost

[151]	2019	yes	PPO	Biped	Position PD	Reward defined with respect to a reference motion
[152]	2019	yes	PPO	Biped	Position PD	Reward defined with respect to a reference motion
[149]	2019	no	PPO	Biped	Position PD	Reward defined with respect to a reference motion
[66]	2019	no	PPO	Biped	Muscle activation	Reward defined with respect to a reference motion
[43]	2017	no	PPO	Biped	Torque control	Reward for reaching a target velocity + torque based metabolic cost + encourage the agent to stay near the center of the track
[150]	2018	yes	PPO	Biped	Position PD	Reward defined with respect to a reference motion

Table A.1 Summary of publications cited in the overview of Deep Reinforcement Learning algorithms applied to robotics.

Appendix B

Active Efficient Coding

Network	Architecture
Encoder (vergence, cyclovergence)	conv 96 filters, size 8×8 stride 4
	conv 24 filters, size 1×1 stride 1
Decoder (vergence, cyclovergence)	conv 384 filters, size 1×1 stride 1
Encoder (pan, tilt)	conv 192 filters, size 8×8 stride 4
	conv 48 filters, size 1×1 stride 1
Decoder (pan, tilt)	conv 768 filters, size 1×1 stride 1
Critic	conv 2×2 stride 1
	max-pooling 2×2 stride 2
	flatten
	concatenate all scales
	fully-connected 200
	fully-connected 9

Table B.1 Network architectures

Parameter	Value
Pan range	$[-10°, 10°]$
Tilt range	$[-10°, 10°]$
Vergence range	$[0°, 20°]$
Discount factor γ	0
Encoder learning rate	5.10^{-4}
Critic learning rate	5.10^{-4}
Episode length	10 iterations
Buffer size	1000 transitions
Batch size	200 transitions
Epsilon ε	0.05
Reward scaling factor C	600

Table B.2 Parameter values

Appendix C

Hierarchical Reinforcement Learning

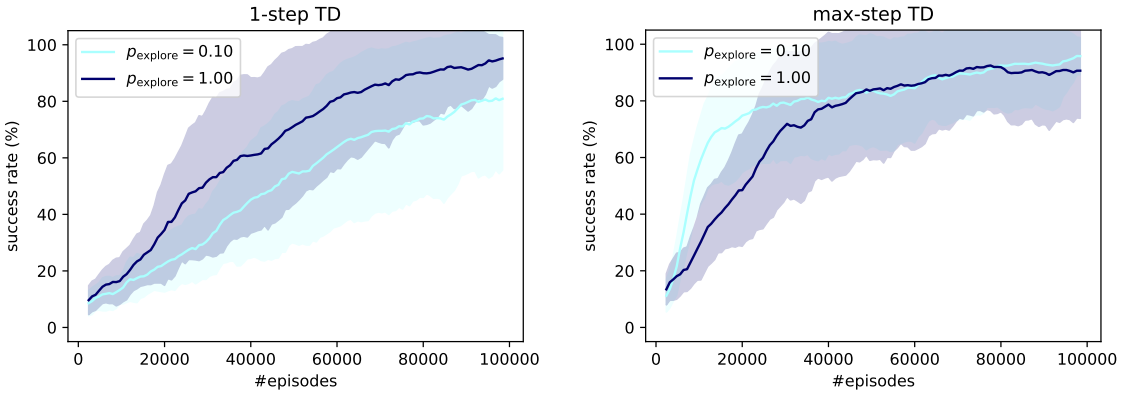


Fig. C.1 Evaluation of the success rate throughout training, for 2 values of the p_{explore} parameter, and for 2 update rules (1-step TD and max-step TD). `default` architecture, with a policy learning rate decreased from 10^{-3} down to 10^{-4}.

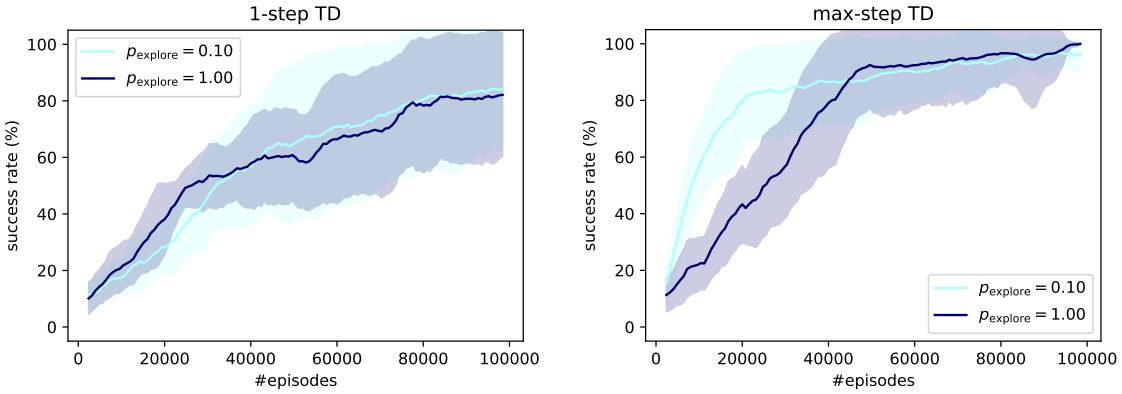


Fig. C.2 Evaluation of the success rate throughout training, for 2 values of the p_{explore} parameter, and for 2 update rules (1-step TD and max-step TD). `default` architecture, but the number of layers has been decreased from 6 down to 3.

Charles Wilmot, PhD

Research Scientist / Research Engineer

Personal Data

Date of birth:	15 November 1993
Address:	31 rue Joseph Hentgès, 59420 Mouvaux
Phone:	+33 6 71 93 59 19
email:	wilmot@fias.uni-frankfurt.de
Nationality:	French

Work experience

2016 - 2021 — PhD in Artificial Intelligence at FIAS in Frankfurt.

I worked on a European project called GOAL-Robot under the supervision of Jochen Triesch. GOAL stands for Goal Oriented Autonomous Learning. Our main objective is to build a robot that self defines goals in a bottom-up, open-ended manner.

2015 — Internship in a French start-up in Frankfurt

NeoSpheres is an ITHR. Management of data transfer between multiple API.

Studies

2013 - 2016 — National superior school of computer science in Bordeaux.

ENSEIRB is a state French *Grande Ecole*, from which I graduated as IT engineer.

2011 - 2013 — Preparatory classes for *Grandes Ecoles* at the lycee Henry Wallon (Valenciennes), section MPSI (Maths and Physics)

I studied Math and Physics for 2 years in order to prepare the national entrance competitions to *Grandes Ecoles*. Equivalent to at least a good Bachelor's degree in Maths.

Scientific Records

2022 — WILMOT, Charles, Deep Reinforcement Learning Applied to Robotics, PhD thesis.

2021 — WILMOT, Charles, BALDASSARRE, Gianluca, et TRIESCH, Jochen. Learning Abstract Representations through Lossy Compression of Multi-Modal Signals. In : *IEEE Transactions on Cognitive and Developmental Systems*.

2020 — WILMOT, Charles, SHI, Bertram E., et TRIESCH, Jochen. Self-Calibrating Active Binocular Vision via Active Efficient Coding with Deep Autoencoders. In : *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. *IEEE*, p. 1-6.

Languages

French:	native
English:	fluent
German:	fluent

Skills

Languages:	python, C++, GO, MySQL
Prefered ML Libs:	Jax, TensorFlow
Cloud Framworks:	openstack, slurm workload manager
Robotics:	CoppeliaSim from the creators of V-REP, GAZEBO, ROS Robots