HAL
open science

Regroupement de compétences robotiques en compétences plus générales

Adrien Matricon

To cite this version:

Adrien Matricon. Regroupement de compétences robotiques en compétences plus générales. Robotique [cs.RO]. Université de Bordeaux, 2018. Français. NNT : 2018BORD0081 . tel-01895789

HAL Id: tel-01895789

https://theses.hal.science/tel-01895789

Submitted on 15 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

par **Adrien Matricon**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Regroupement de compétences robotiques en compétences plus générales

---

Date de soutenance : 11 juin 2018

Devant la commission d'examen composée de :

David Filliat	Professeur, ENSTA Paristech	Encadrant
Pierre-Yves Oudeyer	Directeur de recherche, Inria Bordeaux	Directeur
Alexandre Pitti	Maître de conférences, Université de Cergy-Pontoise	Rapporteur
Peter Ford Dominey	Directeur de recherche, CNRS	Rapporteur
Sao Mai Nguyen	Maître de conférences, IMT Atlantique	Examinatrice
Pierrick Legrand	Maître de conférences, Université de Bordeaux	Examinateur

Président du jury : Peter Ford Dominey

Résumé La découverte de contingences sensorimotrices et leur structuration en compétences sont des enjeux importants en robotique. En particulier, on souhaite pouvoir apprendre ainsi des compétences qui soient à la fois riches sémantiquement et aussi générales que possible.

Cette thèse s'est intéressée à la question du regroupement de compétences robotiques en compétences plus générales. Après avoir posé formellement les notions que l'on peut trouver dans la littérature de compétence et de compétence paramétrée, nous avons relié les compétences paramétrées aux modèles inverses et nous nous sommes inspirés du dualisme entre modèles directs et inverses pour proposer un formalisme dual, les compétences paramétrées directes.

Nous avons ensuite entrepris de déterminer quand il était ou non pertinent de réunir des compétences au sein d'une même compétence paramétrée directe. Le problème est alors ramené à un problème de régression, et des algorithmes s'inspirant du principe du rasoir d'Ockham sont proposés pour y chercher une solution sous la forme d'un mélange d'experts de complexité minimale. Ces algorithmes sont ensuite appliqués à des données de manipulation d'objets en simulation.

Title Merging Robotic Skills into More General Skills

Abstract The discovery of sensorimotor contingencies and their structuration into skills are both important topics in the field of robotics. In particular, robots need the ability to learn skills which are both semantically rich and as general as possible.

During this thesis, we studied the question of merging robotic skills into more general skills. After formally defining the notions that can be found in the litterature of skills and parameterized skills, we established a link between paramaterized skills and inverse models, then mirrored the dualism between forward and inverse models to propose a dual type of parameterized skills: forward-parameterized skills.

We went on to determine when merging skills into a forward-parameterized skill is relevant and when it's not. The problem is then formulated as a regression problem, and algorithms inspired by the Occam Razor principle are proposed to find a mixture of experts that solves it with minimal complexity. Those algorithms are then applied to simulated object-manipulation data.

Keywords robotics, skills, tasks, transfer, generalization

Mots-clés robotique, compétences, tâches, transfert, généralisation

Laboratoire d'accueil U2IS, ENSTA ParisTech, 828 boulevard des Maréchaux 91762 Palaiseau cedex, au sein de l'équipe Flowers commune avec : Inria Bordeaux - Sud-Ouest, 200 avenue de la Vieille Tour 33405 Talence cedex

Remerciements

Je souhaite tout d'abord remercier mes directeurs de thèse. D'une part Pierre-Yves, pour la grande liberté qu'il m'a laissée, mais tout particulièrement David, qui a été très compréhensif malgré les difficultés que j'ai pu rencontrer dans le cadre de mes travaux de recherche.

Je suis également reconnaissant envers le projet européen DREAM, auquel j'ai pu contribuer durant ma thèse et qui m'a financé lors des derniers mois de celle-ci.

Je souhaite par ailleurs remercier Catherine et Nicole, qui font toutes les démarches pour nous afin que nous puissions nous (presque) ne pas nous en soucier et nous focaliser sur nos recherches, ainsi que Thierry qui a dû lutter avec l'administration pour me permettre de donner des TDs.

Merci ensuite à Thibault, Timothée, François, Olivier, Julien, Alexandre, Antonin et Vyshakh pour leurs discussions ainsi que pour tous ces thés et cafés pris ensemble suivis d'une "réunion" en Acte II.

Je souhaite également remercier Panagiotis, Clément et Natalia, avec qui j'ai apprécié partager successivement mon bureau.

Merci aussi à Arturo, David Octavian, Sébastien, Moad, Harmish, Tianming et Yvon, les membres de la "team Magnan", pour tous ces repas que nous avons partagés le midi.

Un grand merci également à Antoine, Clément x2, Louise, Pierre-Henri, Céline, Arturo, Pauline et Gennaro pour toutes les soirées jeux entre doctorants ou entre doctorants et ex-doctorants.

Je souhaite enfin remercier Daniela, Adina, Roxana, François, Emmanuel, Adriana, Omar, Saad, Yuxin, Thomas, Amir, Mathieu, Bruno, Antoine, Vladimir, ainsi que les stagiaires successifs et que tous ceux que j'ai oubliés, bref tous ces gens que j'ai fréquentés à l'U2IS et qui ont contribué à faire de mon passage en ces lieux un séjour mémorable.

Je garderai en particulier un souvenir indélébile des poignées de main de Pierre-Henri, du docteur Maboul dans les expériences d'Arturo, des questionnaires de Roxana, de l'odeur de l'encens dans le bureau d'Omar, de Thibault qui répare les jouets du bébé de Daniela (ainsi que mon grille-pain), du "droit de réponse" de Timothée, des goodies de Natalia, des minions de François, et du son des pas d'Antoine qui se balade dans le couloir.

Merci aussi à tous ceux qui m'ont soutenu pendant ces trois ans et demi,

qu'il s'agisse de ma famille ou de mes amis, ainsi qu'aux membres de l'association FaeriX que je fréquente depuis maintenant 5 ans.

Bref, merci à tous !

Table des matières

Table des matières **vii**

Introduction **1**
Contexte . . . 1
De la fiction à la réalité . . . 1
Du robot expert au robot domestique ? . . . 3
La robotique développementale . . . 5
Objectifs et contributions . . . 6
Problème traité . . . 6
Contributions . . . 7
Organisation des chapitres . . . 7

1 Les compétences robotiques **9**
1.1 Au-delà des actions : les options . . . 9
1.2 Qu'est-ce qu'une tâche ? . . . 12
1.3 Vous avez dit compétence ? . . . 13
1.4 Tâches et compétences paramétrées . . . 14
1.5 Transfert entre plusieurs apprentissages . . . 15
1.5.1 Les problématiques de transfert . . . 17
1.5.2 Les modalités du transfert . . . 21
1.5.3 Transfert et compétences . . . 25
1.6 Conclusion . . . 27

2 Une nouvelle approche **29**
2.1 Tâches, buts, effets . . . 29
2.2 Modèles directs, modèles inverses . . . 31
2.3 Expérience de pensée . . . 35
2.3.1 Situation . . . 35
2.3.2 Formalisme, modèle direct . . . 35
2.3.3 Modèles inverses et généralisation . . . 36
2.3.4 Une nouvelle forme de généralisation . . . 37
2.3.5 Pertinence . . . 38
2.4 Formalisation de l'approche . . . 41

2.4.1 Quelques définitions 42
2.4.2 Hypothèses de travail 42
2.4.3 Discussion des hypothèses 42
2.4.4 Cœur de l'approche 43
2.5 Conclusion 44

3 Les algorithmes de régression 45
3.1 Enjeu des méthodes de régression 45
3.2 Régression linéaire 46
3.2.1 Moindres carrés 46
3.2.2 Maximum de vraisemblance 48
3.2.3 Dilemme biais-variance, régression d'arête 49
3.2.4 Maximum a posteriori 53
3.2.5 Régression d'angle minimal 54
3.2.6 Régression à vecteurs de support 54
3.2.7 Les autres approches 55
3.2.8 Limites de ces approches 57
3.3 Régression non paramétrique 57
3.3.1 Algorithme des k plus proches voisins 57
3.3.2 Régression par la méthode du noyau 59
3.3.3 LOESS et régression localement pondérée 59
3.3.4 Processus gaussiens 61
3.3.5 Limite de ces approches 63
3.3.6 Cas des réseaux de neurones 63
3.4 Mélange d'experts 64
3.4.1 Régressions locales pondérées et régression pondérée par des champs récepteurs 65
3.4.2 Réduction de la dimensionnalité et régression projective localement pondérée 65
3.4.3 XCSF 66
3.4.4 Utilisation de la distribution des données 67
3.4.5 Limites des méthodes existantes 67
3.5 Conclusion 68

4 L'algorithme COCOTTE 71
4.1 Le compromis entre le nombre et la versatilité des experts . . . 71
4.1.1 Rappel du problème 71
4.1.2 Et si on utilisait la validation croisée ? 73
4.1.3 Utilisation de la complexité : le rasoir d'Ockham 74
4.2 Hypothèses et définitions 78
4.2.1 Retour sur les hypothèses 78
4.2.2 Complexité 79
4.2.3 Rendre compte de données 81

4.3 COCOTTE : algorithme et implémentation 81
4.3.1 Recherche au sein d'une famille de fonctions 82
4.3.2 Recherche d'un expert de complexité minimale 83
4.3.3 Fusion d'experts 86
4.3.4 COCOTTE 87
4.3.5 Extension à plusieurs dimensions 90
4.3.6 Implémentation 90
4.4 Limites et améliorations possibles 91
4.5 Un début d'amélioration : COCOTTE+ 93
4.5.1 Structure de données 93
4.5.2 Fusion d'experts et vol de points 93
4.5.3 Mise à jour de la collection d'arbres 96
4.6 Conclusion 96

5 Expériences **99**
5.1 Expérience : application à des cas simples 99
5.1.1 Comportement lié à l'imprécision 100
5.1.2 Comportement lié aux ruptures de régularité 101
5.1.3 Et COCOTTE+ ? 103
5.2 Expérience : simulation d'interaction d'un bras robotique avec un cube 107
5.2.1 Environnement de simulation 107
5.2.2 Protocole expérimental 108
5.2.3 Résultats 108
5.2.4 Et COCOTTE+ ? 113
5.3 Conclusion 115

Conclusion et perspectives **117**
Résumé 117
Pistes pour de futures recherches 119
Extensions directes des travaux présentés 119
Extensions à plus long terme 120
Perspectives 121

A Pseudo-codes **123**

Bibliographie **131**

Introduction

Contexte

De la fiction à la réalité

Qu'est-ce qu'un robot ? Si tous ne s'accordent pas sur une définition précise, on peut toutefois avancer qu'un robot est avant tout un travailleur artificiel ; le mot "robot" vient lui-même du tchèque "robota", qui signifie "travail, besogne, corvée". Du temps où les robots n'existaient encore que dans la fiction, qu'une œuvre mette en avant les risques que peut présenter une telle technologie ou les moyens d'y faire face, le robot est toujours pensé au départ comme un machine dont le rôle est de remplacer l'homme dans toutes les tâches ingrates ou dangereuses, voire de supprimer le besoin même pour l'homme de travailler (*cf.* Figure 1).

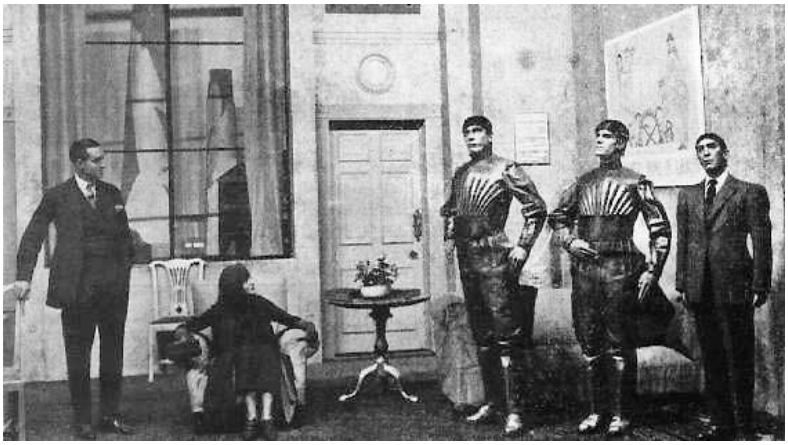


FIGURE 1 – Une scène de *R.U.R (Rossum's Universal Robots)*, pièce de théâtre écrite en 1920 par l'auteur tchèque Karel Čapek et dans laquelle le mot *robot* apparut pour la première fois. Les robots y sont des machines biologiques à l'apparence humaine, qui travaillent et font la guerre à la place des hommes. Lorsque les hommes les dotent de la capacité d'éprouver émotions et sentiments, les robots prennent conscience de leur condition et se révoltent contre l'humanité, qu'ils finissent par remplacer.

C'est donc sans surprise que dans les années 1960, lorsque les premiers robots industriels font leur apparition dans les chaînes de production automobile (*cf.* Figure 2) et que la recherche en intelligence artificielle vient d'entrer dans son premier âge d'or, nombreux sont ceux qui s'attendent à ce que la décennie qui suit voie les robots investir peu à peu tous les domaines de la production et s'occuper des tâches ménagères.

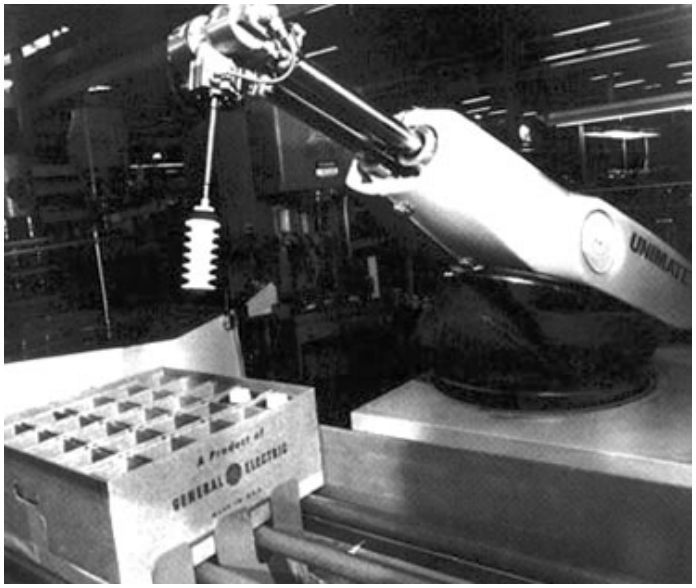


FIGURE 2 – Unimate, le premier robot industriel. Il fut vendu à partir de 1961 par *Unimation*, et est utilisé pour la première fois sur les lignes d'assemblage de *General Motors*. Photographie mise à disposition par http ://www.computerhistory.org

Malheureusement, il n'en est rien. Malgré de nombreux progrès réalisés en intelligence artificielle, notamment dans la manipulation de symboles, la recherche se heurte à un mur : donner à des machines la capacité d'établir des raisonnements logiques poussés ne suffit pas à leur donner une intelligence similaire à l'homme et — contrairement à ce à quoi on pourrait s'attendre — il s'avère bien plus difficile pour une machine d'identifier une pièce d'échec sur une photographie que de décider le prochain coup à jouer. Les chercheurs se voient en effet dans l'incapacité de donner à un robot une "culture générale", qui permettrait de mettre en correspondance le monde réel (perçu via divers capteurs) avec des symboles que celui-ci saurait manipuler (c'est le problème du fondement des symboles, ou *symbol grounding* en anglais). Incapable de répondre aux attentes démesurées qu'il a suscité, le domaine essuie les critiques et les années 1970 marquent le premier hiver de l'intelligence artificielle.

On assiste dans les années 1980 à un renouveau de l'intelligence artificielle, qui rentre dans son second âge d'or avec l'apparition des systèmes experts : bien qu'il reste impossible de donner une "culture générale" à un robot, il est

en revanche possible — en se plaçant dans un environnement contrôlé pour réaliser une tâche bien identifiée — de fournir au robot les connaissances spécifiques, c'est-à-dire l'expertise, dont il a besoin pour réaliser cette tâche. Cette approche permet aux robots d'investir de nouveaux secteurs de la production, mais finit cependant par trouver ses limites : rendre un robot expert dans la réalisation d'une tâche donnée demande du temps et un personnel qualifié, suite à quoi le robot doit travailler dans un environnement contrôlé. La mise en place d'un robot expert est donc lourde, ce qui limite fortement la rentabilité d'une robotisation pour la plupart des postes. Les robots restent par conséquent cantonnés à certaines tâches dans les usines, et ce jusqu'à aujourd'hui. Ils sont encore bien loin de trouver leur place dans nos foyers.

Du robot expert au robot domestique ?

Imaginons un instant qu'il existe des robots domestiques à destination du grand public, qui puissent s'occuper des tâches ménagères ou rendre divers services dans les foyers. Un utilisateur pourrait donner à son robot la consigne suivante : "va à la cuisine me chercher une bouteille de jus de fruits dans le réfrigérateur". Pour être capable d'exécuter cette consigne, d'apparence simple, de nombreux sous-problèmes devraient être résolus par le robot : identifier les mots employés par l'utilisateur, en déduire le sens de la phrase, être capable de se déplacer dans l'espace, *etc*(*cf.* Figure 3). En particulier, il est nécessaire que le robot soit capable d'identifier le réfrigérateur dans la cuisine et d'interagir avec celui-ci.

Dans un environnement contrôlé, où l'on sait exactement de quel modèle est le réfrigérateur (donc sa forme, ses dimensions, sa couleur, *etc*) et ce à quoi ressemble la cuisine, un ingénieur peut utiliser sa connaissance de l'environnement pour donner au robot une expertise spécifique à la situation, et le munir de moyens d'accomplir sa tâche qui reposent sur des connaissance a priori de la cuisine et des objets qui s'y trouvent (exemple : aller à une position prédéfinie, ajuster sa position par rapport au plus grand objet rectangulaire blanc dans le champ de vision de la caméra, réaliser des actions prédéfinies pour ouvrir la porte du réfrigérateur). Tout ceci serait possible si nous avions affaire à un robot d'usine, mais nous sommes ici dans un contexte fondamentalement différent : un robot domestique à destination du grand public, probablement produit en série. L'approche du robot expert n'est plus viable, car le robot doit pouvoir accomplir sa tâche dans n'importe quelle cuisine et être capable d'identifier et d'interagir avec le réfrigérateur indépendamment du modèle de celui-ci (quelles que soient sa forme, sa couleur, ses dimensions, *etc*). Pour pouvoir créer des robots domestiques, il faut donc changer radicalement d'approche.

La plupart des pistes explorées actuellement pour s'affranchir des limites des systèmes experts reposent sur l'apprentissage statistique (*machine lear-*

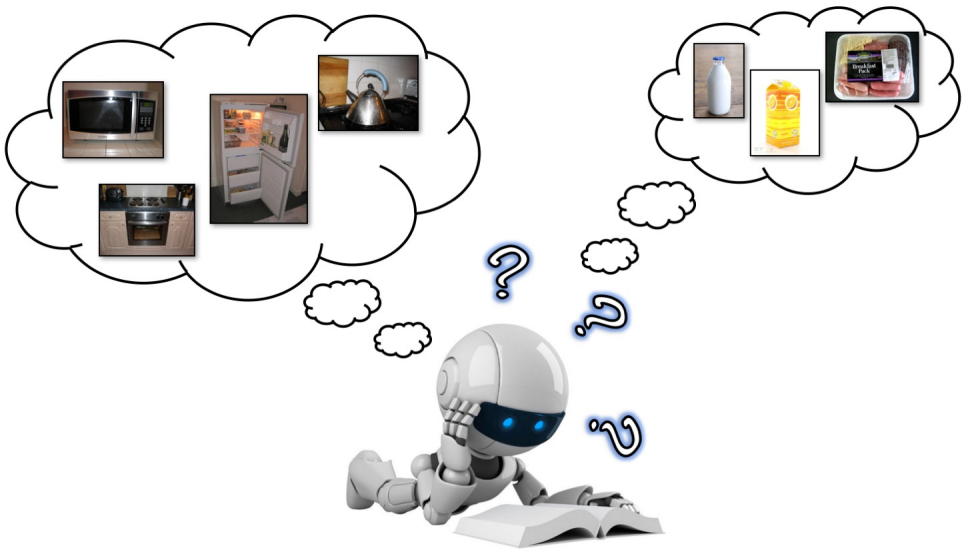


FIGURE 3 – Lorsque l'utilisateur lui demande d'aller lui chercher du jus de fruit au réfrigérateur, le robot doit résoudre de nombreux sous-problèmes, notamment identifier le réfrigérateur dans la cuisine et le jus de fruits à l'intérieur de celui-ci.

ning en anglais), un domaine de l'intelligence artificielle fondé vers la fin des années 1950 et visant à permettre à des machines d'accomplir des tâches pour lesquelles elles n'ont pas été explicitement programmées. On peut distinguer deux utilisations distinctes du *machine learning* en robotique. La première consisterait dans l'exemple précédent à rassembler autant de données que possible sur des réfrigérateurs de modèles divers, puis à entraîner des algorithmes d'apprentissage sur ces données de façon à faire émerger des connaissances générales permettant d'identifier ou d'interagir avec un réfrigérateur de modèle quelconque. L'apprentissage permettrait donc ici l'acquisition d'une expertise bien plus générale que celle qu'un humain pourrait formuler pour un robot expert. Cette approche, qui s'inscrit dans la lignée de celles visant à donner une "culture générale" aux robots pour résoudre le problème du *symbol grounding*, connaît un fort regain d'intérêt ces dernières années grâce à la disponibilité de bases de données de grandes tailles, à l'augmentation de la puissance de calcul des ordinateurs au fil de années et à de nombreux progrès réalisés en apprentissage profond (un sous-domaine du *machine learning*), qui ont ensemble permis d'obtenir des résultats qui semblent très prometteurs.

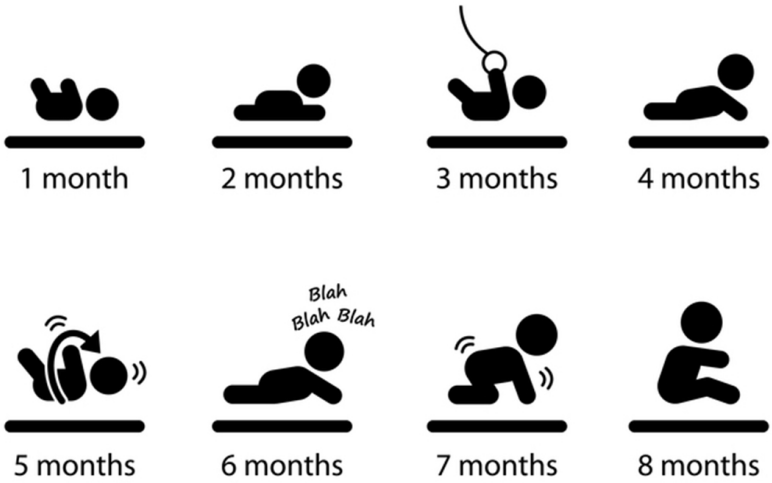
La seconde utilisation possible du *machine learning* en robotique, moins ambitieuse, consiste à utiliser les algorithmes d'apprentissage non pas en amont pour donner une "expertise générale" au robot, mais directement chez l'utilisateur pour permettre au robot d'acquérir sur place l'expertise nécessaire à la réalisation des tâches qui lui sont confiées. Dans l'exemple précédent, le robot

apprendrait donc à identifier et interagir spécifiquement avec le réfrigérateur de l'utilisateur. Si celui-ci dispose de plusieurs réfrigérateurs, une expertise propre à chacun peut être acquise. Bien que cela ne soit pas strictement nécessaire, il est ensuite envisageable de rassembler ces expertises spécifiques en une expertise plus générale.

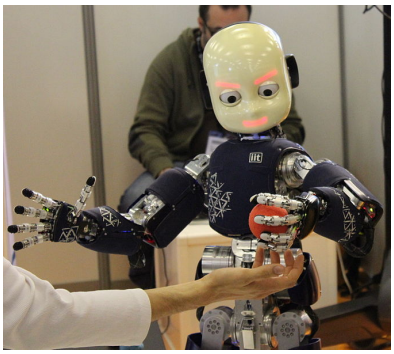
La robotique développementale

S'inscrivant dans le cadre de la seconde approche décrite plus haut, la robotique développementale est un domaine de recherche s'inspirant des mécanismes de la cognition humaine pour permettre à un robot d'apprendre à se représenter son environnement et à interagir avec celui-ci. Similairement à ce que l'on observe lors du développement de l'enfant (*cf.* Figure 4), le but est alors de donner à un robot les moyens d'acquérir progressivement des connaissances de plus en plus complexes à l'aide de comportements exploratoires et d'interactions sociales (un humain pouvant par exemple guider l'apprentissage du robot tel que le ferait un parent ou un professeur).

Une problématique majeure en robotique développementale est celle de l'exploration de l'espace sensorimoteur (c'est-à-dire des perceptions et des actions), et plus précisément de la découverte des contingences sensorimotrices (c'est-à-dire de l'influence des actions du robot sur ses perceptions). L'utilisation de la structuration de ces contingences sensorimotrices constitue une piste prometteuse pour donner à un robot les moyens de mettre en correspondance son environnement avec des symboles qu'il soit capable de manipuler. Ainsi, on peut voir comme l'acquisition d'une compétence l'apprentissage que fait le robot à la fois d'une capacité à prédire les changements que vont provoquer certaines de ses actions sur son environnement et d'une capacité à déterminer comment produire ces changement via ses actions. Cette acquisition de compétences peut ensuite permettre de découvrir les affordances (*i.e.* possibilités d'action) de l'environnement en mettant en relation la perception de certains éléments de celui-ci (*e.g.* , la forme d'une poignée) avec des compétences qui y sont propres (*e.g.* , saisir la poignée). Les objets pourraient alors être définis par les affordances qu'ils offrent au robot.



(a)



(b)

(c)

FIGURE 4 – (a) Un forte inspiration pour la robotique développementale : les étapes du développement de l'enfant. (b et c) Les robots iCub (b) et Poppy (c), deux des plateformes robotiques les plus utilisées en robotique développementale.

Objectifs et contributions

Problème traité

Cette thèse s'inscrit dans la cadre de la robotique développementale. On y considère un robot interagissant avec son environnement selon une démarche exploratoire, et découvrant des contingences sensorimotrices qu'il cherche à structurer en compétences. Les compétences ainsi trouvées peuvent être plus ou moins générales (*e.g.* , des compétences ⌈*soulever de 2cm*⌋ et ⌈*soulever de 5cm*⌋ sont moins générales qu'une compétence ⌈*soulever de n'importe quelle hauteur*⌋, dont elles sont des cas particuliers).

Plus une compétence est générale, plus elle est intéressante à découvrir. En effet, une compétence générale couvre l'ensemble des compétences qui en

sont des cas particuliers, qu'il s'agisse de compétences qui auraient pu être découvertes à partir des interactions que le robot a eu avec son environnement ou même de compétences ne correspondant à aucune interaction passée (en interpolant ou extrapolant à partir des autres cas particuliers).

Ceci pose des questions majeures, auxquelles cette thèse tentera de répondre : quand peut-on regrouper des compétences en une compétence plus générale ? Quand cela n'a t-il pas de sens ? Comment structurer les contingences sensorimotrices en compétences les plus générales possibles ?

Contributions

Cette thèse a été l'occasion de poser plus formellement les notions de "compétence" et de "compétence paramétrée", et de les relier aux travaux visant à partager des connaissances entre différentes situations d'apprentissage. Elle relie également le concept de compétences aux notions de modèles directs et de modèles inverses, permettant ainsi de définir un nouveau type de compétences paramétrées — plus propice à un regroupement en des compétences plus générales — et de ramener le problème de la structuration des contingences sensorimotrices en compétences les plus générales possible à un problème de régression.

Dans le cadre de cette nouvelle approche, apprendre des compétences paramétrées revient à une régression vers une fonction localement régulière. Il s'agit alors de chercher les portions du domaine de définition où celle-ci est régulière, et pour chacun d'entre eux de construire explicitement un modèle local de la fonction. Cette thèse propose pour cela un algorithme reposant sur une minimisation de complexité qui, de façon non supervisée et sans hyperparamètres ou termes de régularisation, permet de réaliser cette régression.

Ces contributions ont fait l'objet d'une publication dans Matricon *et al.* [2016], et une implémentation de l'algorithme en C++ a été mise en ligne sur la plateforme GitHub, où elle est ainsi accessible à tous.

Organisation des chapitres

Le chapitre 1 établit la notion de compétence robotique, et situe les approches de généralisation de compétences parmi d'autres problématiques similaires. Le chapitre 2 montre ensuite les limites de ce formalisme pour le regroupement de compétences en compétences plus générales. Il propose une extension de ce formalisme, qui fait émerger une approche duale permettant de s'affranchir de ces limites et de ramener le problème de la découverte et de la généralisation des compétences robotiques à un problème de régression. Le chapitre 3 dresse alors un état de l'art des algorithmes de régression et questionne leur adéquation dans le cadre de ce problème spécifique. Le chapitre 4 est enfin consacré à l'algorithme d'apprentissage que nous avons développé

pour résoudre le problème — ainsi qu'à une version améliorée de celui-ci — qui se veut plus approprié au cas de l'apprentissage des compétences que les algorithmes vus au chapitre 3. Les résultats obtenus au moyen de celui-ci sont présentés au chapitre 5.

Chapitre 1

Les compétences robotiques

Considérons un agent robotique en interaction avec son environnement. Il perçoit cet environnement à l'aide de capteurs, et influe sur celui-ci à l'aide d'actionneurs. À la conception, on le munit de différentes façons de mettre en œuvre ces actionneurs, qui constituent alors le répertoire d'*actions* du robot. Il peut s'agir d'actions très simples comme des commandes motrices, ou d'actions plus complexes comme des contrôleurs (*e.g.* , pour donner un certain angle à une articulation, placer un membre du robot à une certaine position...). Selon la forme que prennent ces actions, elles peuvent être exécutables depuis toutes les configurations du robot et de son environnement, ou seulement depuis certaines d'entre elles.

Au-delà des actions se trouvent les *compétences* (*skills* en anglais), qui constituent une généralisation du concept d'action et sont généralement apprises par le robot lui-même au cours de son fonctionnement. Il plane cependant un certain flou dans la littérature sur ce concept ; les définitions semblent fluctuer d'un champ de recherche à l'autre voire d'un auteur à l'autre, selon le contexte, les objectifs des travaux... Le terme "compétence" est le plus souvent associé à la notion d'*option*, bien mieux définie dans la littérature scientifique, et nous nous appuierons donc sur celle-ci pour définir formellement ce que nous appellerons une compétence par la suite. Certains auteurs vont jusqu'à assimiler ces deux concepts, mais nous marquerons ici une légère différence entre les deux.

1.1 Au-delà des actions : les options

Lorsqu'un humain se rend dans la pièce voisine, il exécute une séquence d'actions : ⌈*se déplacer jusqu'à la porte*⌋, ⌈*ouvrir la porte*⌋, puis ⌈*franchir la porte*⌋. On peut cependant remarquer que ⌈*ouvrir la porte*⌋ est également une séquence d'actions complexe, qui peut elle-même être décomposée en trois étapes (⌈*placer la main sur la poignée*⌋, ⌈*tourner la poignée*⌋, ⌈*pousser*⌋), pouvant à leur tour être décomposées en sous-séquences. De même, ⌈*se déplacer*

jusqu'à la porte⌋ et ⌈*franchir la porte*⌋ sont des séquences d'actions complexes, et on peut remarquer qu'elles incluent toutes deux des sous-séquences d'actions similaires, correspondant à la marche (*cf.* Figure 1.1).

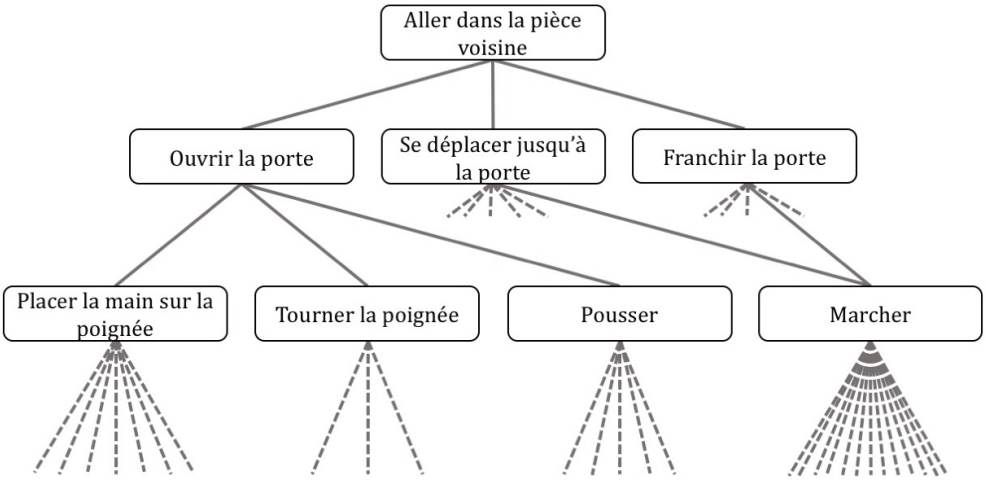


FIGURE 1.1 – Décomposition hiérarchique d'une séquence d'actions complexe. La séquence d'actions décomposée correspond à ⌈*Aller dans la pièce voisine*⌋

Ce qu'illustre cet exemple, c'est que les séquences d'actions complexes souvent mises en œuvre pour effectuer diverses tâches sont généralement composées de sous-séquences plus simples, dont certaines peuvent être retrouvées à différents niveaux de la décomposition. Pour refléter cette réalité, Sutton *et al.* [1999] propose les *options*, une généralisation du concept d'action où les actions généralisées sont construites hiérarchiquement selon une logique de composition.

Plus formellement, considérons un agent qui interagit avec un environnement donné. Il perçoit l'état $s \in \mathcal{S}$ de cet environnement et choisit en fonction de cet état une action $a \in \mathcal{A}_s \subseteq \mathcal{A}$ à effectuer, où $\mathcal{S}$ est l'espace des états, $\mathcal{A}$ l'espace des actions et $\mathcal{A}_s$ le sous-espace des actions de $\mathcal{A}$ qui peuvent être exécutées depuis l'état s. On note $\mathcal{D}_\mathcal{A}$ l'espace des distributions de probabilité sur $\mathcal{A}$.

Définition 1.1. On appelle *politique* toute fonction $\pi : \mathcal{S} \to \mathcal{D}_\mathcal{A}$, qui associe à chaque état de s de $\mathcal{S}$ une distribution de probabilité $\mathbb{P}_s$ sur $\mathcal{A}$ telle que $\forall a \notin \mathcal{A}_s,\ \mathbb{P}_s(a) = 0$. On dit que l'agent *suit la politique* π s'il choisit l'action qu'il exécute depuis tout état s selon la distribution de probabilité $\pi(s)$.

Définition 1.2. On appelle *option* un triplet (I, π, β), où $I \subseteq \mathcal{S}$ est un ensemble (appelé *ensemble d'initiation*) d'états depuis lesquels l'option est disponible, π est une politique qui peut être suivie par l'agent, et $\beta : \mathcal{S} \mapsto [0, 1]$ est une fonction (appelée *condition de terminaison*) qui assigne à chaque état une probabilité de décider après l'avoir atteint de ne pas choisir de nouvelle

action et d'arrêter de suivre la politique. Dans le cas où β n'est non nulle qu'en un unique état $s_\beta \in \mathcal{S}$ atteignable depuis I et vers lequel la politique finit par converger, π peut être vue comme un moyen d'atteindre l'état terminal s_β depuis n'importe quel état de I (*cf.* Figure 1.2).

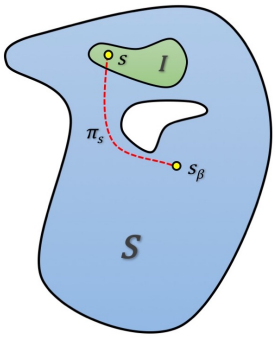


FIGURE 1.2 – Exécution d'une option à partir de l'état $s \in I \subseteq \mathcal{S}$ jusqu'à un état terminal s_β

Les actions de $\mathcal{A}$ (c'est-à-dire le répertoire d'actions à disposition du robot dès sa conception), peuvent être vues comme des cas particuliers d'options : l'action a peut être associée à l'option (I_a, π_a, β_a), où I_a est l'ensemble des états depuis lesquels a est disponible (*i.e.* l'ensemble $\{s \mid a \in \mathcal{A}_s\}$), π_a est la politique qui choisit toujours a (*i.e.* la politique définie par $\forall s \in \mathcal{S},\ \pi_a(s) = \mathbb{P}$, où $\mathbb{P}$ est la distribution de probabilité telle que $\mathbb{P}(a) = 1$) et β_a termine toujours après la première action (*i.e.* la fonction définie par $\forall s \in \mathcal{S},\ \beta_a(s) = 1$). On retrouve alors bien que les options sont des actions généralisées, et il devient possible d'étendre la définition d'une politique (définition 1.1) aux options.

Définition 1.3. Soit $\mathcal{O}$ l'espace des options, $\mathcal{O}_s$ le sous-espace des options de $\mathcal{O}$ disponibles depuis l'état $s \in \mathcal{S}$, et $\mathcal{D}_\mathcal{O}$ l'espace des distributions de probabilités sur $\mathcal{O}$. On appelle *politique* toute fonction $\pi : \mathcal{S} \mapsto \mathcal{D}_\mathcal{O}$, qui associe à chaque état s de $\mathcal{S}$ une distribution de probabilité $\mathbb{P}_s$ sur $\mathcal{O}$ telle que $\forall o \notin \mathcal{O}_s,\ \mathbb{P}_s(o) = 0$. On dit que l'agent *suit la politique* π s'il choisit l'option qu'il exécute depuis tout état s selon la distribution de probabilité $\pi(s)$.

De nouvelles options peuvent alors être acquises via toute méthode permettant d'apprendre des politiques. Stolle et Precup [2002] et Kompella *et al.* [2014] utilisent par exemple des techniques d'apprentissage par renforcement pour générer de nouvelles options, tandis que Konidaris *et al.* [2011] en découvre à partir de la segmentation de trajectoires montrées par l'humain dans un contexte d'apprentissage par imitation.

Ainsi, les actions que l'agent peut initialement exécuter forment un répertoire d'options, qui sert ensuite de base pour construire hiérarchiquement des options de plus en plus complexes qui viennent le compléter. L'agent a alors

à sa disposition de nombreuses actions généralisées, exécutables à différentes échelles de temps et de complexité, qu'il pourra mettre en œuvre pour atteindre les divers objectifs qui lui seront fixés.

1.2 Qu'est-ce qu'une tâche ?

En robotique, ou plus généralement dans un contexte d'apprentissage automatique, l'apprentissage vise à terme à permettre à l'agent d'atteindre des objectifs qui lui sont fixés (*e.g.* , aller chercher un objet, classifier des images, faire des prédictions...). On parle alors de *tâche* que l'agent doit accomplir.

Ces tâches peuvent être formulées explicitement, mais peuvent aussi l'être implicitement via des récompenses données par l'utilisateur au robot et que celui-ci cherchera à maximiser (cette approche, inspirée du dressage des animaux, est appelée *apprentissage par renforcement — reinforcement learning* en anglais — et un exemple en est donné en Figure 1.3).

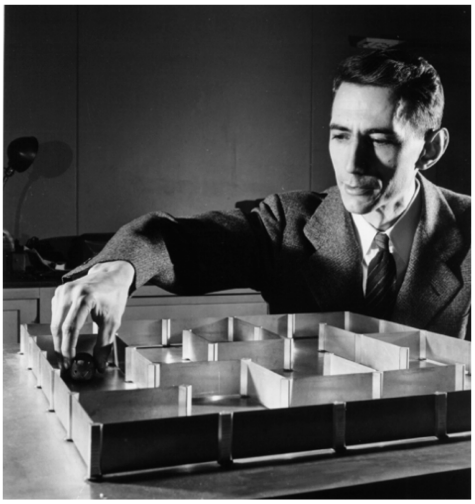


FIGURE 1.3 – Claude Shannon procédant à une expérience d'apprentissage par renforcement en 1950. Un robot-souris est placé dans un labyrinthe et, de même que dans une expérience analogue avec une vraie souris, on attend de celui-ci qu'il atteigne une certaine position du labyrinthe. En cas de succès, il reçoit une récompense.

Dans le cadre spécifique de la robotique, les tâches que l'on considère consistent généralement soit à atteindre un état ou un ensemble d'états donnés (*e.g.* , un état où un certain objet est à une position donnée), soit à adopter un comportement souhaité (*e.g.* , exécuter une danse, éviter un obstacle, suivre une trajectoire désirée).

Dans le formalisme des options, un robot est donc capable d'accomplir une tâche lorsqu'il a acquis une option convergeant vers un état terminal souhaité,

et/ou correspondant à une politique souhaitée (définie implicitement à l'aide des récompenses).

1.3 Vous avez dit compétence ?

Il est maintenant temps de définir le terme *compétence*, qui est central pour cette thèse. De nombreux travaux (*e.g.* , Pastor *et al.* [2009], Kulic *et al.* [2012], Peters et Schaal [2008], Bansal *et al.* [2017]) semblent utiliser indifféremment les termes "compétence", "compétence motrice" et "primitive de mouvement" (respectivement *skill*, *motor skill* et *motion/movement primitive* en anglais). Une *primitive de mouvement* est une politique paramétrée de bas niveau, qui permet à un robot de déterminer les commandes motrices nécessaires à la réalisation d'un mouvement, c'est-à-dire d'une trajectoire dans son espace moteur. Dans le formalisme vu précédemment, les primitives de mouvement viennent former le répertoire $\mathcal{A}$ d'actions initialement à disposition du robot. La disponibilité d'actions de plus haut niveau que les commandes motrices facilite alors d'un point de vue combinatoire la planification de politiques de plus haut niveau.

D'autres travaux définissent simplement le terme "compétence" comme un synonyme d'"option" (*e.g.* , Kompella *et al.* [2014]), et certains définissent indépendamment les compétences comme étant des primitives de mouvement et des options au sein des mêmes travaux (Konidaris *et al.* [2011]). Par-delà les définitions, le concept de compétence se voit aussi parfois fortement associé aux tâches que ces compétences permettent de résoudre, et ce qu'il s'agisse de primitives de mouvement (*e.g.* , Kober *et al.* [2012] désigne les compétences auxquelles il se réfère par les tâches qu'elles résolvent, comme par exemple "la compétence d'atteindre un point précis d'un mur avec une fléchette") ou qu'il s'agisse d'options (*e.g.* , da Silva *et al.* [2012] parle quant à lui d'apprendre des compétences résolvant certaines tâches spécifiques avant de proposer une généralisation des compétences à des classes de tâches reliées entre elles).

Nous avons vu plus haut (*cf.* section 1.1) que le formalisme des options généralisait la notion d'action, et permettait ainsi de construire des politiques plus haut niveau ne se limitant pas au répertoire d'actions initialement à la disposition du robot. Définir les compétences comme des primitives de mouvement — donc des politiques bas niveau destinées à constituer ce répertoire initial d'actions — serait par conséquent très restrictif, et nous y préférerons une définition reposant sur le formalisme des options. Le flou sur la définition nous laissant une certaine marge de manœuvre, nous ferons le choix dans cette thèse d'y intégrer la notion de tâche — qui nous a semblé importante — à l'aide de la définition suivante :

Définition 1.4. On appelle *compétence* tout couple (o, τ) composé d'une option $o = (I, \pi, \beta)$ et d'une tâche τ qu'elle résout. Pour alléger le forma-

lisme, on supposera généralement que la politique π est munie d'une "option" o_{fin} qui met fin à son exécution lorsqu'elle est choisie. On peut alors poser $\forall s \in \mathcal{S},\ \mathbb{P}_s(o_{fin}) = \beta(s)$, où $\mathbb{P}_s = \pi(s)$, ce qui permet d'unifier β et π. On notera alors une compétence comme un triplet (I, π, τ).

1.4 Tâches et compétences paramétrées

Nous avons jusqu'ici parlé de tâches uniques. Il arrive cependant que l'on choisisse de considérer non pas une unique tâche mais une classe de tâches reliées entre elles (*e.g.* , les tâches consistant à mettre un objet à diverses positions données). On peut alors représenter chacune des différentes tâches de cette classe par un vecteur de paramètres, et on appelle *tâche paramétrée* l'ensemble des tâches de cette classe (*cf.* Figure 1.4).

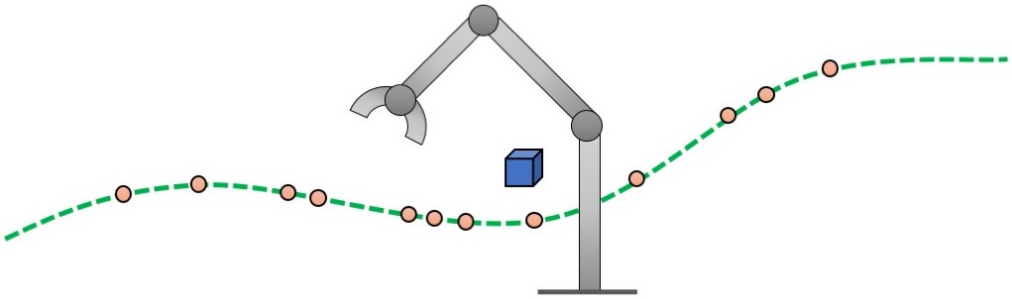


FIGURE 1.4 – Un robot est confronté à diverses tâches individuelles, consistant chacune à placer un cube à une position donnée. Ces diverses tâches sont très similaires, et les positions à atteindre forment une courbe, ce qui permet donc de regrouper toute ces tâches uniques en une tâche paramétrée, toute abscisse curviligne le long de cette courbe pouvant par exemple servir de paramètre.

Une tâche unique peut souvent être résolue par des politiques variées (donc diverses options), et peut être associée à diverses compétences. Une option en revanche ne correspond qu'à une unique politique (résolvant généralement une unique tâche), et ne peut par conséquent être associée qu'à une unique compétence. Une seule option ne peut donc pas résoudre une tâche paramétrée.

C'est pourquoi da Silva *et al.* [2012] introduit la notion de *compétence paramétrée* (*parameterized skill* en anglais), qui se veut une généralisation des options à des politiques paramétrées. Le terme est originellement utilisé pour désigner une fonction qui associe à chaque instance d'une tâche paramétrée une politique qui la résout, mais nous allons cependant adapter ici cette définition pour qu'elle s'intègre mieux à notre formalisme.

Définition 1.5. Soit $\mathcal{T}$ un espace continu de tâches reliées entre elles, paramétrées dans un espace $\overline{\mathcal{T}} \subseteq \mathbb{R}^{|\mathcal{T}|}$. Soit également $\mathcal{P}$ un espace continu de

politiques reliées entre elles, paramétrées dans un espace $\overline{\mathcal{P}} \subseteq \mathbb{R}^{|\mathcal{P}|}$. On appelle *compétence paramétrée* tout quadruplet $(I, \mathcal{P}, \mathcal{T}, f)$, où la fonction $f : \mathcal{T} \mapsto \mathcal{P}$ est une fonction qui associe à chaque instance d'une tâche paramétrée une politique qui la résout, c'est-à-dire une fonction telle que pour toute tâche $\tau \in \mathcal{T}$, le triplet $(I, f(\tau), \tau)$ est une compétence (*cf.* Figure 1.5). On appelle indifféremment *fonction de résolution* cette fonction f et la fonction $\overline{f} : \overline{\mathcal{T}} \mapsto \overline{\mathcal{P}}$ qui lui correspond dans les espaces de paramètres, c'est-à-dire la fonction telle que $\forall \tau \in \mathcal{T},\ \overline{\pi} = \overline{f}(\overline{\tau})$, où $\overline{\tau}$ et $\overline{\pi}$ sont les vecteurs de paramètres associés respectivement à la tâche τ et à la politique $\pi = f(\tau)$.

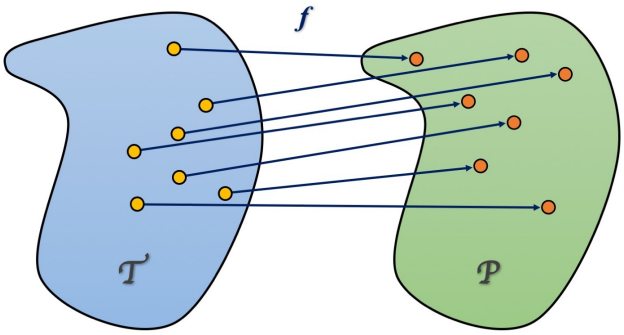


FIGURE 1.5 – La fonction de résolution f associe à chaque tâche individuelle de la tâche paramétrée $\mathcal{T}$ une politique qui la résout dans l'espace de politiques paramétrées $\mathcal{P}$.

Pour apprendre des compétences paramétrées, da Silva *et al.* [2012] procède d'abord aux apprentissages individuels de compétences associées à diverses tâches uniques. Il identifie ensuite les politiques trouvées qui peuvent être groupées entre elles au sein de politiques paramétrées — le long de variétés de dimension plus faible que l'espace des politiques — puis détermine les tâches paramétrées auxquelles ces politiques paramétrées correspondent. Il estime alors pour chacune la fonction de résolution qui associe les tâches individuelles qui la composent aux politiques qui les résolvent.

1.5 Transfert entre plusieurs apprentissages

Si l'on considère un cadre plus large que celui de la robotique, l'apprentissage de compétences paramétrées tel que décrit plus haut s'inscrit dans la lignée de problématiques dites de *transfert*, liées à l'apprentissage de la résolution de tâches. Nous allons donc nous attarder sur celles-ci pour fournir un cadre dans lequel replacer les travaux cités précédemment.

Partant du constat que certaines situations d'apprentissages ne sont pas entièrement indépendantes les unes des autres, le *transfert* consiste à tirer

partie du lien entre les diverses situations pour que l'apprentissage de l'une aide à l'apprentissage de l'autre, et ainsi obtenir de meilleures performances (vitesse d'apprentissage, qualité de ce qui est appris). Les diverses problématiques que cela recoupe se fondent sur deux concepts : les tâches (vues plus haut), et les *domaines*.

Définition 1.6. On appelle *domaine* tout ce qui définit les conditions spécifiques à une certaine situation d'apprentissage. Dans le cas de l'apprentissage d'une tâche de prédiction, le domaine inclura généralement l'espace des caractéristiques à partir duquel les prédictions sont réalisées ainsi que la distribution statistique selon laquelle sont tirées les données. Dans un contexte robotique, le domaine regroupera l'espace d'états, la distribution statistique selon laquelle sont tirés les états de départ, ainsi que tous les autres éléments qui pourraient changer d'une situation d'apprentissage à l'autre (masses des objets, géométrie du robot, *etc*).

De même qu'il existe des classes de tâches reliées entre elles et des tâches paramétrées, il existe des classes de domaines reliés entre eux et des *domaines paramétrés*. Ainsi, pour l'apprentissage d'un jeu similaire au bilboquet (*cf.* Figure 1.6), Stulp *et al.* [2014] place le robot dans deux situations distinctes, correspondant respectivement à des cordes de 25cm et 30cm de long : bien que la tâche reste la même (faire arriver la boule au bon endroit), les deux situations correspondent à des domaines différents. Ces domaines sont néanmoins très similaires, et peuvent être vus comme des instances d'un domaine paramétré par la longueur de la corde.

La question du transfert des connaissances associées à deux situations d'apprentissage se pose donc dans des contextes spécifiques où ces situations mettent en jeu des tâches différentes mais reliées entre elles (*e.g.* , faisant partie de la même tâche paramétrée), et/ou des domaines différents mais reliés entre eux (*e.g.* , faisant partie du même domaine paramétré).

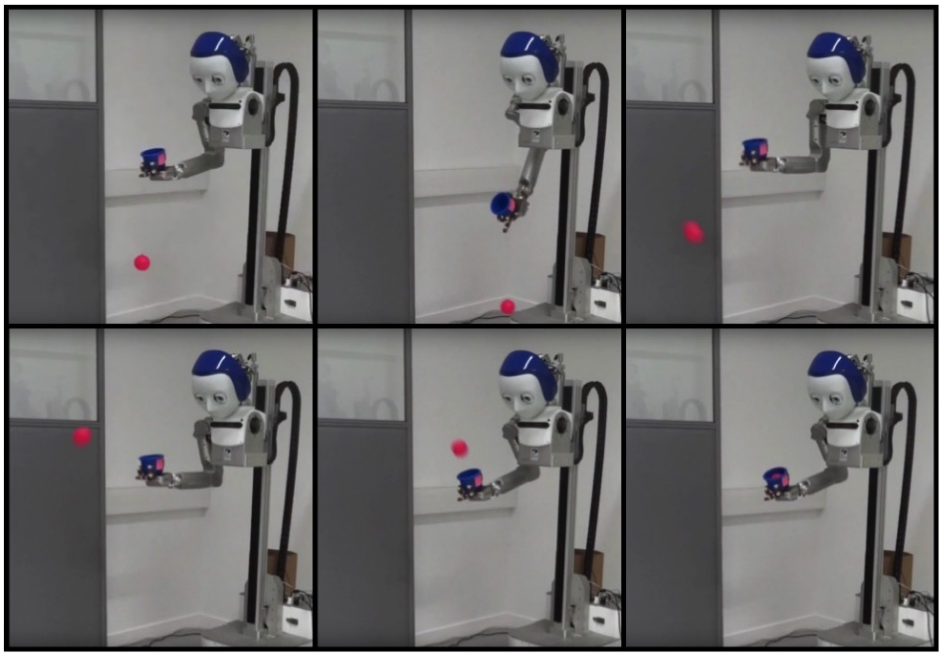


FIGURE 1.6 – Expérience réalisée par Stulp *et al.* [2014] : un robot tient un gobelet auquel est reliée une boule par un fil. La tâche que doit accomplir le robot est d'atteindre un état où la boule est dans le gobelet. L'expérience est réalisée avec des fils de longueur 25cm et 30cm, donc avec deux domaines distincts.

1.5.1 Les problématiques de transfert

En fonction des applications, la question du transfert peut correspondre à diverses problématiques. Passons les principales en revue.

Apprentissage autodidacte

L'*apprentissage autodidacte* (*self-taught learning* en anglais) vise à améliorer les performances d'apprentissage d'une tâche donnée (*tâche cible*), réalisée depuis son domaine associé (*domaine cible*), à l'aide de données provenant d'un autre domaine (*domaine source*) et correspondant possiblement à une autre tâche (*tâche source*). On suppose que les domaines sont reliés entre eux, mais que la tâche source est complètement décorrélée de la tâche cible (on ne cherche donc pas à tirer parti d'une quelconque similarité entre les tâches). Cette problématique est illustrée en Figure 1.7.

Cette problématique est introduite par Raina *et al.* [2007], qui propose un algorithme d'apprentissage pour une tâche de classification. L'algorithme est entraîné sur un jeu d'image étiquetées (donc en quantité relativement limitée), en s'aidant d'un grand nombre d'images non étiquetées en provenance d'internet.

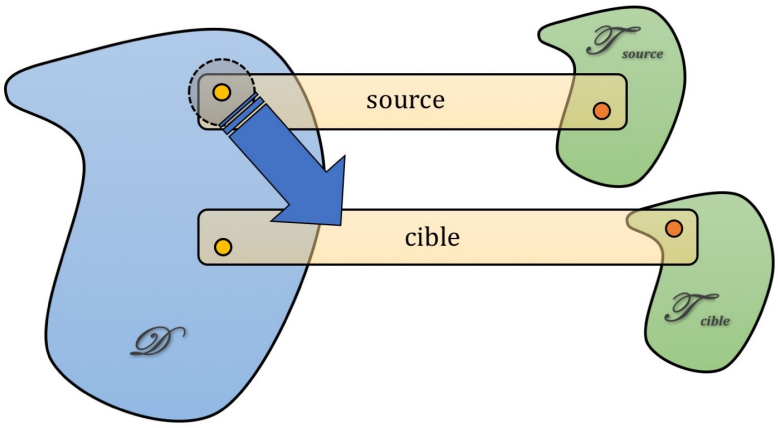


FIGURE 1.7 – On suppose deux situations d'apprentissage, chacune associée à un domaine et à une tâche. Les tâches, supposées sans lien entre elles, sont issues de deux espaces de tâches distincts $\mathscr{T}_{source}$ et $\mathscr{T}_{cible}$ tandis que les domaines appartiennent à un même espace de domaines $\mathscr{D}$ et sont supposés reliés entre eux. L'enjeu de l'apprentissage autodidacte est alors d'utiliser le lien entre les domaines pour transférer des connaissances de l'apprentissage source vers l'apprentissage cible et ainsi obtenir de meilleures performances d'apprentissage.

Apprentissage par transfert

Plus générale que l'apprentissage autodidacte, la problématique sobrement appelée *apprentissage par transfert* (*transfer learning* en anglais) tire également parti des similarités entre les tâches. L'agent apprend séquentiellement à résoudre d'abord une tâche source depuis un domaine source, puis une tâche cible depuis un domaine cible. L'enjeu est alors d'utiliser l'apprentissage source pour améliorer les performances de l'apprentissage cible, en s'appuyant sur un lien entre les domaines et/ou les tâches de ces deux situations d'apprentissage (Pan et Yang [2010]). Cette problématique est illustrée en Figure 1.8.

L'apprentissage par transfert peut se révéler particulièrement utile lorsque les données d'entraînement sont disponibles en faible quantité pour la tâche cible (*e.g.* , peu de données étiquetées pour une tâche de classification). Il constitue également un enjeu important en robotique, où l'acquisition de nouvelles données d'entraînement est coûteuse en temps, et où il peut dans certains cas permettre à un robot de ne pas être entièrement démuni lorsqu'il rencontre une nouvelle tâche.

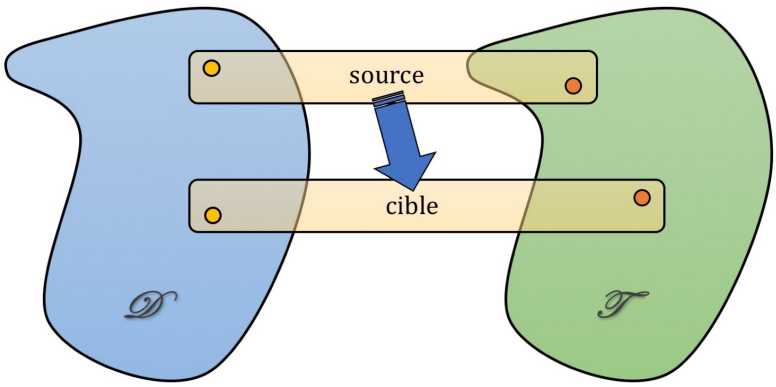


FIGURE 1.8 – On suppose deux situations d'apprentissage, chacune associée à un domaine et à une tâche (on notera $\mathscr{D}$ l'espace des domaines, et $\mathscr{T}$ l'espace des tâches). L'enjeu de l'apprentissage par transfert est alors d'utiliser un lien entre les domaines et/ou entre les tâches pour transférer des connaissances de l'apprentissage source vers l'apprentissage cible et ainsi obtenir de meilleures performances d'apprentissage.

Adaptation de domaine

Dans le cadre de l'*adaptation de domaine* (*domain adaptation* ou *covariance shift* en anglais), l'agent apprend d'abord à résoudre une tâche source depuis un domaine source, puis est confronté à cette même tâche depuis un domaine cible différent du domaine source mais relié à celui-ci (Beijbom [2012], Hoffman *et al.* [2014], Sun *et al.* [2015]). Il s'agit donc d'un cas particulier de l'apprentissage par transfert où la tâche source est identique à la tâche cible. L'enjeu y est le plus souvent de compenser une différence entre les distributions statistiques des données d'entraînement et celles des situations auxquelles l'agent est ensuite réellement confronté. Cette problématique est illustrée en Figure 1.9.

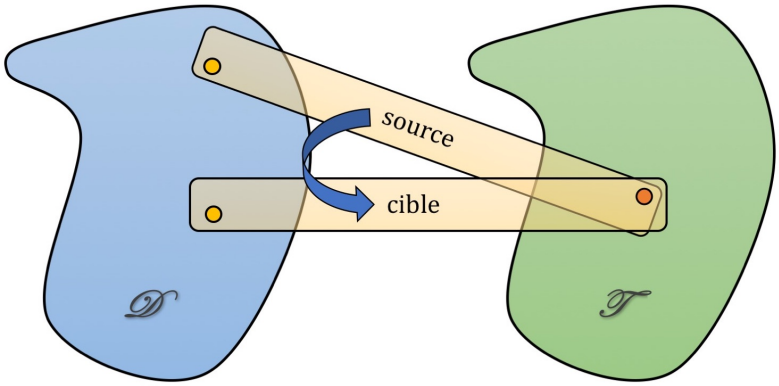


FIGURE 1.9 – On suppose deux situations d'apprentissage, chacune associée à un domaine distinct et à la même tâche (on notera $\mathscr{D}$ l'espace des domaines, et $\mathscr{T}$ l'espace des tâches). L'enjeu de l'adaptation de domaine est alors d'utiliser un lien entre les domaines pour transférer des connaissances de l'apprentissage source vers l'apprentissage cible et ainsi obtenir de meilleures performances d'apprentissage.

Apprentissage multi-tâches et multi-domaines

L'*apprentissage multi-tâches* (*multi-task learning* en anglais) est une problématique très similaire à l'apprentissage par transfert, mais s'affranchissant de la contrainte de séquentialité. L'agent doit apprendre à résoudre diverses tâches depuis un domaine donné (Evgeniou et Pontil [2004], Kumar et III [2012]), et l'enjeu est alors d'obtenir de meilleures performances en réalisant ces apprentissages de façon conjointe que s'ils étaient réalisés indépendamment les uns des autres. Il s'agit pour cela de s'appuyer sur le domaine commun et d'éventuels liens entre les tâches mises en jeu.

Dans l'*apprentissage multi-domaines* (*multi-domain learning* en anglais), l'agent doit à l'inverse apprendre à résoudre une même tâche depuis divers domaines (Dredze *et al.* [2010]), mais l'enjeu reste d'obtenir de meilleures performances en réalisant ces apprentissages de façon conjointe que s'ils étaient réalisés indépendamment les uns des autres. Il s'agit cette fois de s'appuyer sur la tâche commune et d'éventuels liens entre les domaines mis en jeu.

Combinant les deux approches, l'agent est confronté dans l'*apprentissage multi-domaines et multi-tâches* à diverses tâches qu'il doit apprendre à résoudre depuis leurs domaines associés. Il s'agit toujours d'obtenir des performances au moins aussi bonnes pour l'apprentissage conjoint que s'ils étaient faits indépendamment les uns des autres, en s'appuyant sur les éventuels liens entre les domaines et/ou les tâches (Yang et Hospedales [2014], Devin *et al.* [2016]). Cette problématique est illustrée en Figure 1.10.

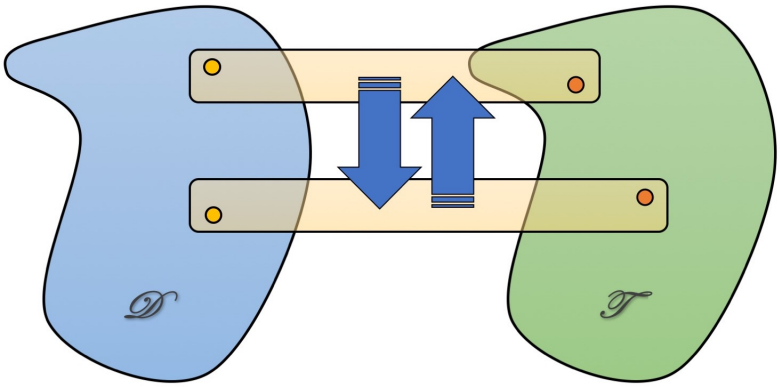


FIGURE 1.10 – On suppose plusieurs situations d'apprentissage, chacune associée à un domaine à une tâche (on notera $\mathscr{D}$ l'espace des domaines, et $\mathscr{T}$ l'espace des tâches). L'enjeu d'un apprentissage multi-tâches et multi-domaines est alors d'utiliser un lien entre les domaines et/ou entre les tâches pour échanger des connaissances entre les divers apprentissages et ainsi obtenir de meilleures performances conjointes que si les apprentissages étaient réalisés indépendamment les uns des autres.

1.5.2 Les modalités du transfert

Intéressons-nous maintenant à comment se fait le transfert en lui-même, c'est-à-dire aux modalités du partage de connaissances entre deux situations d'apprentissage.

Transfert d'instances

Le *transfert d'instance* est le partage de données d'entraînement entre deux situations d'apprentissage (*cf.* Figure 1.11). Quand les données en provenance d'un domaine cible sont peu nombreuses, il est possible d'utiliser les données d'entraînement de l'apprentissage source pour compléter celles de l'apprentissage cible, par exemple en adaptant la technique du *boosting* aux problématiques de transfert (Dai *et al.* [2007]).

Le *boosting* est originellement une technique permettant, dans le cadre d'une tâche de classification, de combiner divers classifieurs en un classifieur plus performant. C'est par exemple ce que fait l'algorithme AdaBoost (Freund et Schapire [1997]), qui entraîne successivement les divers classifieurs, puis combine leurs prédictions à l'aide d'une moyenne pondérée. À chaque fois qu'un nouveau classifieur est entraîné, des poids sont attribués aux divers points de données afin de biaiser l'entraînement en faveur des données précédemment mal classifiées, et de permettre au nouveau classifieur de se concentrer principalement sur la classification de ceux-ci.

Dai *et al.* [2007] adapte la technique du *boosting* pour l'utiliser dans un cadre d'apprentissage par transfert : l'entrainement se fait avec des données

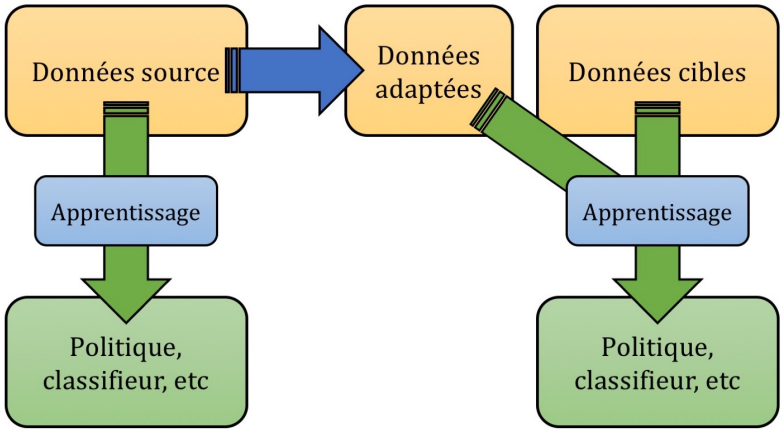


FIGURE 1.11 – Le transfert d'instances consiste à adapter les données d'entraînement d'une situation d'apprentissage source afin de pouvoir les utiliser comme données pour une situation d'apprentissage cible.

en provenance à la fois du domaine cible et du domaine source, en attribuant des poids aux divers points de données afin de biaiser l'entrainement en faveur des données issues du domaine cible. Au sein du domaine source, les poids attribués aux points de données dépendent de leur similarité avec les données d'entraînement de l'apprentissage cible.

Transfert de représentations

En apprentissage automatique, une donnée est généralement représentée par un ensemble de *caractéristiques*, c'est-à-dire de propriétés individuellement mesurables (un poids, une masse, la couleur d'un pixel...). Dans un certain nombres de cas, comme par exemple lorsque l'on travaille avec des images, les caractéristiques brutes des données peuvent s'avérer à la fois redondantes et très nombreuses. Dans ces cas il est courant de réaliser un prétraitement des données, où de nouvelles caractéristiques composites sont construites par combinaison des anciennes, et où les caractéristiques pertinentes sont ensuite sélectionnées. On aboutit alors à une nouvelle représentation des données. Le jeu de caractéristiques utilisé à l'issue de ce prétraitement peut être choisi manuellement, mais il est aussi possible de le choisir automatiquement à partir des données. Le *transfert de représentations* est le partage de représentations et de jeux de caractéristiques entre plusieurs situations d'apprentissage (*cf.* 1.12).

Ainsi, dans le cadre d'une tâche de classification d'images, Raina *et al.* [2007] commence par considérer des données non étiquetées en provenance d'internet (domaine source), et détermine une base de vecteurs sur lesquels ces images peuvent être projetées avec une erreur minimale. Chaque image d'entraînement de l'apprentissage cible est alors représentée par le vecteur qui

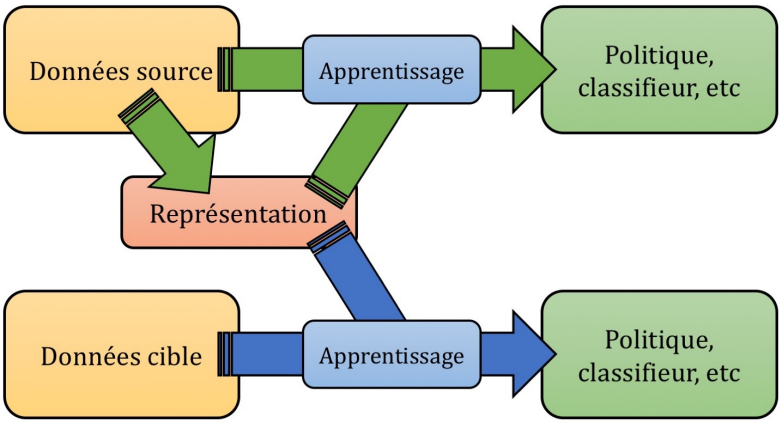


FIGURE 1.12 – Le transfert de représentation consiste à utiliser les représentations découvertes dans le cadre d'un apprentissage source comme jeu de caractéristiques pour un apprentissage cible.

la représente le mieux dans cette base.

Transfert de paramètres

Le *transfert de paramètres* est la mise en commun de certains paramètres entre plusieurs apprentissages, afin qu'ils reflètent les aspects communs de ceux-ci et soient appris conjointement (*cf.* Figure 1.13).

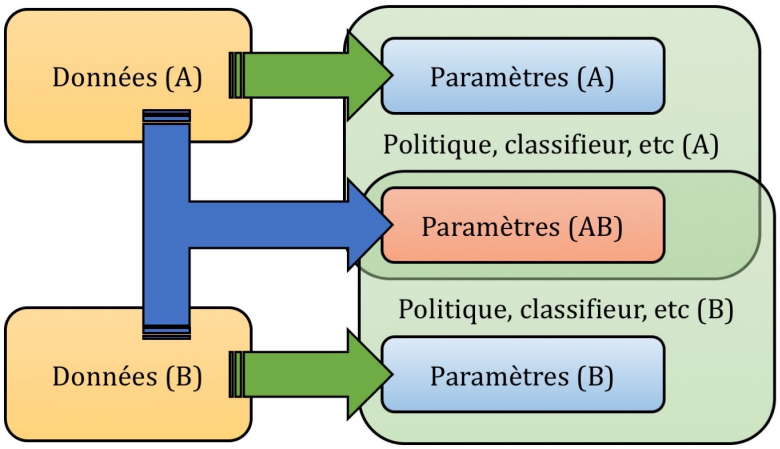


FIGURE 1.13 – Le transfert de paramètres consiste à mettre en commun certains paramètres entre plusieurs apprentissages, afin qu'ils reflètent les aspects communs de ceux-ci et soient appris conjointement.

Kumar et III [2012] fait par exemple l'hypothèse que les diverses tâches qu'il considère sont des combinaisons linéaires d'une base de tâches latentes, dont les solutions peuvent être combinées linéairement pour obtenir celles de

chacune des tâches. La matrice W associant chaque tâche à sa solution est alors exprimée sous la forme d'un produit de deux matrices $W = LS$. Ceci permet de séparer la matrice S, qui permet d'exprimer chaque tâche comme combinaison linéaire de ces tâches latentes, de la matrice L qui associe à chaque tâche latente une solution et dont les coefficients sont appris à l'aide de l'ensemble des données.

Lawrence et Platt [2004] considère quant à lui diverses instances d'une tâche paramétrée, dont les solutions peuvent être déterminées individuellement comme la détermination des noyaux de processus gaussiens. Il suppose alors les vecteurs de paramètres de ces instances comme des tirages d'une variable aléatoire gaussienne, ce qui permet alors de formuler l'apprentissage conjoint de ces diverses instances comme la détermination du noyau d'un unique processus gaussien.

Autres approches

Au-delà des approches présentées plus haut, d'autres peuvent être plus appropriées à certains contextes spécifiques.

Mihalkova *et al.* [2007] propose ainsi dans le cadre de l'apprentissage de réseaux logiques de Markov d'identifier quels prédicats du domaine source sont analogues à des prédicats du domaine cible (*e.g.* , un réalisateur, ses acteurs et le film qu'ils tournent sont analogues respectivement à un professeur, aux élèves qui l'assistent dans ses recherches, et à l'article de recherche qu'ils rédigent) avant de transposer les relations entre ces prédicats d'un domaine à l'autre.

Dans un contexte de recherche de politique pour un apprentissage par renforcement, où l'apprentissage part généralement d'une initialisation arbitraire ou aléatoire, Taylor *et al.* [2005] fait débuter l'apprentissage cible depuis la solution de l'apprentissage source.

Transfert négatif et oubli catastrophique

Comme nous l'avons vu plus haut, les techniques de transfert permettent d'utiliser les similarités entre diverses situations d'apprentissage de façon à ce que l'apprentissage des unes aide à l'apprentissage des autres.

Deux écueils sont cependant à éviter. Le premier est ce que l'on appelle le *transfert négatif* (Rosenstein *et al.* [2005], Pan et Yang [2010]) : si les situations d'apprentissage ne sont pas similaires, supposer une similarité des solutions à tort mène la plupart du temps à une baisse des performances. Le second, que l'on appelle l'*oubli catastrophique* (French [1999], Kirkpatrick *et al.* [2017]), arrive lorsque l'on traite successivement deux situations d'apprentissage mais que le traitement de la seconde entraîne l'oubli complet et abrupt de ce qui avait été appris pour la première (ce qui peut arriver lorsque l'on réalise de l'apprentissage par transfert à l'aide de certains algorithme d'apprentissage en ligne comme par exemple des réseaux de neurones).

Lorsque l'on souhaite transférer des connaissances entre deux situations d'apprentissage, il faut donc pour éviter ces écueils déterminer soigneusement au préalable ce sur quoi doit porter le transfert et quelles en seront les modalités. Il s'agit ainsi de ne transférer les connaissances qu'uniquement là où les tâches ou domaines sont similaires (en évitant donc tout transfert là où ils sont dissimilaires), et de s'assurer que toute modification de paramètres lors de l'apprentissage de nouvelles tâches ou de nouveaux domaines ne nuit pas à la qualité des solutions des apprentissages précédents.

1.5.3 Transfert et compétences

Nous avons maintenant passé en revue les principales problématiques et modalités du transfert, ce qui nous permet de porter un éclairage nouveau sur l'apprentissage de compétences en robotique.

Nous avons vu précédemment (*cf.* section 1.3) qu'une compétence est un triplet (I, π, τ) composé d'une politique π, d'un espace d'initiation I depuis lequel π peut être exécutée, et d'une tâche τ résolue par π. Découvrir des compétences signifie par conséquent apprendre des politiques résolvant diverses tâches depuis divers espaces d'initiation, ce qui revient à une problématique d'apprentissage multi-tâches et multi-domaines de politiques (car lorsque le choix du robot et les conditions expérimentales sont fixés — ce que présuppose implicitement le formalisme des options — le domaine est entièrement défini par l'ensemble d'initiation). Nous proposons de nous appuyer sur ceci pour fournir une définition étendue de "compétence" :

Définition 1.7. Une *compétence* est un triplet (δ, π, τ), où τ est une tâche, δ un domaine, et π une politique qui résout τ depuis δ.

Ainsi les travaux de da Silva *et al.* [2012] sur la découverte de compétences paramétrées, qui généralisent des compétences résolvant des tâches reliées entre elles depuis le même domaine (*cf.* section 1.4), peuvent être vus comme une forme d'apprentissage multi-tâches de politiques. La fonction de résolution d'une compétence paramétrée permet alors un transfert de paramètres entre les diverses compétences uniques qu'elle recouvre. Il nous paraît par ailleurs pertinent d'étendre la définition des compétences paramétrées à un apprentissage multi-tâches et multi-domaines de politiques :

Définition 1.8. Soit $\mathcal{T}$ un espace continu de tâches reliées entre elles, paramétrées dans un espace $\overline{\mathcal{T}} \subseteq \mathbb{R}^{|\mathcal{T}|}$. Soient également $\mathcal{D}$ un espace continu de domaines reliés entre eux, paramétrés dans un espace $\overline{\mathcal{D}} \subseteq \mathbb{R}^{|\mathcal{D}|}$, et $\mathcal{P}$ un espace continu de politiques reliées entre elles, paramétrées dans un espace $\overline{\mathcal{P}} \subseteq \mathbb{R}^{|\mathcal{P}|}$. On appelle *compétence paramétrée* tout quadruplet $(\mathcal{D}, \mathcal{P}, \mathcal{T}, f)$, où la fonction $f : \mathcal{D} \times \mathcal{T} \mapsto \mathcal{P}$ est une fonction qui associe à chaque instance d'un domaine paramétré et chaque instance d'une tâche paramétrée une politique qui résout cette tâche depuis ce domaine, c'est-à-dire une fonction telle

que pour tout domaine $\delta \in \mathcal{D}$ et toute tâche $\tau \in \mathcal{T}$, le triplet $(\delta, f(\delta, \tau), \tau)$ est une compétence au sens de la définition 1.7. On appelle indifféremment *fonction de résolution* cette fonction f et la fonction $\overline{f} : \overline{\mathcal{D}} \times \overline{\mathcal{T}} \mapsto \overline{\mathcal{P}}$ qui lui correspond dans les espaces de paramètres, c'est-à-dire la fonction telle que $\forall(\delta, \tau) \in \mathcal{D} \times \mathcal{T},\ \overline{\pi} = \overline{f}(\overline{\delta}, \overline{\tau})$, où $\overline{\delta}$, $\overline{\tau}$ et $\overline{\pi}$ sont les vecteurs de paramètres associés respectivement au domaine δ, à la tâche τ et à la politique $\pi = f(\delta, \tau)$.

Bien que n'étant pas formulée comme un apprentissage de compétences paramétrées, la question de l'apprentissage multi-tâches et multi-domaines de politiques a été traitée par Devin *et al.* [2016], dans un contexte d'apprentissage par renforcement via un réseau d'apprentissage profond (*deep reinforcement learning* en anglais). Pour une tâche et un robot (domaine) donné, la politique correspondante est apprise au moyen d'un réseau d'apprentissage profond dont les couches sont regroupées en deux modules, l'un dédié à la tâche et l'autre au robot. À chaque fois que sont choisis une tâche et le robot qui doit la résoudre, les modules correspondant à la tâche et au robot sont connectés pour former le réseau représentant la politique associée, tandis que les autres modules sont déconnectés. Ceci est illustré sur la Figure 1.14.

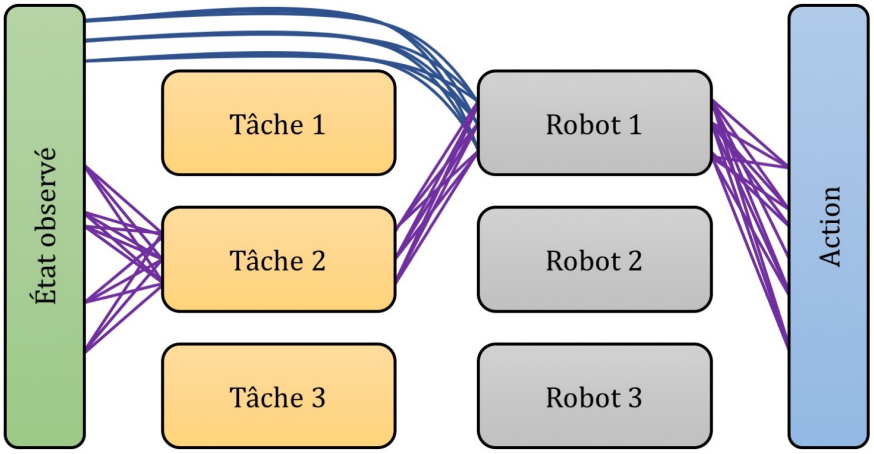


FIGURE 1.14 – Une fois choisis une tâche et un robot (ici la tâche 2 et le robot 1), les modules correspondants sont connectés afin d'obtenir un réseau d'apprentissage profond définissant une politique, qui associe à chaque état observé une action à choisir. Les autres modules sont déconnectés du réseau.

Le module correspondant à un robot donné fait partie de tous les réseaux représentant les politiques réalisées depuis ce robot, et similairement le module correspondant à une tâche donnée fait partie de tous les réseaux visant à résoudre cette tâche. Ceci permet un transfert de paramètres entre les différentes situations d'apprentissage, les connaissances spécifiques à un robot donné ou à une tâche donnée se concentrant dans le module dédié. Des résultats satisfaisants sont obtenus pour des combinaisons robot-tâche non observées l'entraînement.

1.6 Conclusion

Dans ce chapitre, nous avons vu le formalisme des politiques et des options, donné une définition de ce qu'est une tâche, et posé formellement ce qu'est une compétence. Nous avons aussi vu comment tâches et compétences peuvent être étendues en tâches paramétrées et compétences paramétrées.

Nous avons ensuite abordé les différentes problématiques de transfert et leurs modalités, avant de reformuler l'apprentissage de compétences paramétrées comme une technique permettant un transfert de paramètres dans le cadre d'une problématique d'apprentissage multi-tâches et multi-domaines de politiques. Nous avons alors utilisé ce formalisme pour étendre les définitions de "compétence" et "compétence paramétrée" à un cadre un peu plus général.

Chapitre 2

Une nouvelle approche

Nous allons à présent nous intéresser aux limites du formalisme et des approches présentés au chapitre 1 pour la découverte de compétences et leur regroupement en compétences les plus générales possibles. Nous nous baserons alors sur ces considérations pour présenter une extension du formalisme et établir une nouvelle approche.

2.1 Tâches, buts, effets

Au chapitre 1, nous avons vu qu'une compétence peut être définie comme la connaissance commune d'une tâche et d'une option qui la résout (définition 1.4), puis avons donné une définition étendue posant une compétence comme la connaissance conjointe d'une tâche, d'un domaine, et d'une politique qui résout cette tâche depuis ce domaine (définition 1.7). Que l'on prenne l'une ou l'autre définition, elles ont en commun le fait de s'appuyer sur la notion de tâche, et c'est ce qui va nous intéresser ici.

Dans le cadre de cette thèse, nous souhaitons nous placer dans le contexte de la robotique développementale, un cadre dans lequel on attend de l'agent qu'il explore ses contingences sensorimotrices par lui-même, puis qu'il les structure en compétences distinctes. Le robot doit par conséquent être capable d'apprendre des compétences sans qu'un utilisateur extérieur ne formule pour lui des tâches à résoudre, ce qui était le cas dans les travaux traitant d'apprentissage multi-tâches que nous avons abordés au chapitre précédent, et en particulier dans les travaux de Devin *et al.* [2016] (tâches segmentées par l'utilisateur, chacune associée à un module du réseau) et dans ceux de da Silva *et al.* [2012] (espace des tâches paramétrisé par l'utilisateur).

Plutôt que de recevoir des tâches d'une tierce personne, une alternative serait que l'agent détermine par lui-même quelles "tâches" il va devoir chercher à accomplir. Cette idée d'un agent qui détermine lui-même ses objectifs n'est pas neuve, et se retrouve dans de nombreux travaux mettant en jeu une exploration dite *dirigée par les buts* (Oudeyer *et al.* [2007], Rolf et Steil [2012],

Moulin-Frier et Oudeyer [2014]). Voyons comment elle se présente.

Lorsque l'agent exécute une action ou une politique, celle-ci produit des effets observables sur l'environnement du robot (état atteint, récompense donnée par l'utilisateur, *etc*). Quand un utilisateur confie une tâche à l'agent, ce qu'il fait est donc en réalité le choix d'un effet qu'il désire que l'agent produise. Automatiser le processus revient par conséquent de déterminer les modalités d'un tel choix et à laisser l'agent choisir par lui-même quels effets il va chercher à produire. Ces effets sont alors appelés *buts*, pour les distinguer des *tâches* fixées de l'extérieur par un utilisateur.

L'exploration des contingences sensorimotrices commence donc par la sélection par l'agent d'un point de l'espace des effets (*effect space* en anglais, aussi appelé *goal space* signifiant "espace des buts"), suivie de l'estimation d'une politique qui pourrait lui permettre d'atteindre ce but, puis de l'exécution de celle-ci. L'agent sélectionne alors un nouveau but afin de continuer son exploration, et recommence. Cette approche est illustrée en Figure 2.1.

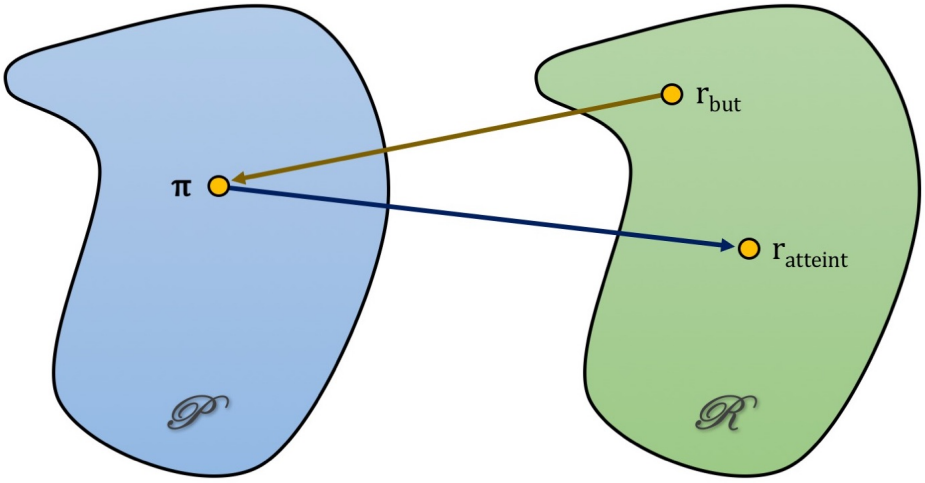


FIGURE 2.1 – L'agent se fixe un but r_{but} dans l'espace des effets $\mathscr{R}$. Il estime ensuite une politique π de l'espace des politiques $\mathscr{P}$ qui pourrait lui permettre de produire l'effet r_{but}, compte-tenu du domaine δ dans lequel il se trouve. L'exécution de π résulte en un effet $r_{atteint}$. La connaissance de $(\delta, \pi, r_{atteint})$ contribue alors à l'apprentissage de l'agent, et l'aide ainsi à estimer des politiques menant à des effets plus proches de ceux désirés les fois suivantes.

Pour nous placer dans le cadre de la robotique développementale, nous proposons une définition adaptée de "compétence", où les buts remplacent donc les tâches :

Définition 2.1. On appelle *compétence* tout triplet (δ, π, r), où r est un effet (désigné par la lettre r pour "résultat") et où π est une politique permettant de produire cet effet depuis le domaine δ.

Et pour les compétences paramétrées :

Définition 2.2. Soient $\mathcal{D}$ un espace continu de domaines reliés entre eux et paramétrés dans un espace $\overline{\mathcal{D}} \subseteq \mathbb{R}^{|\mathcal{D}|}$, $\mathcal{P}$ un espace continu de politiques reliées entre elles et paramétrées dans un espace $\overline{\mathcal{P}} \subseteq \mathbb{R}^{|\mathcal{P}|}$, et $\mathcal{R}$ un espace continu d'effets reliés entre eux et paramétrés dans un espace $\overline{\mathcal{R}} \subseteq \mathbb{R}^{|\mathcal{R}|}$. On appelle *compétence paramétrée* tout quadruplet $(\mathcal{D}, \mathcal{P}, \mathcal{R}, f)$, où la fonction $f : \mathcal{D} \times \mathcal{R} \to \mathcal{P}$ est une fonction qui associe à chaque instance d'un domaine paramétré et chaque instance d'un effet paramétré une politique qui permet de produire cet effet depuis ce domaine, c'est-à-dire une fonction telle que pour tout domaine $\delta \in \mathcal{D}$ et tout effet $r \in \mathcal{R}$, le triplet $(\delta, f(\delta, r), r)$ est une compétence au sens de la définition 2.1. On appelle indifféremment *fonction de résolution* cette fonction f et la fonction $\overline{f} : \overline{\mathcal{D}} \times \overline{\mathcal{R}} \to \overline{\mathcal{P}}$ qui lui correspond dans les espaces de paramètres, c'est-à-dire la fonction telle que $\forall(\delta, r) \in \mathcal{D} \times \mathcal{R}$, $\overline{\pi} = \overline{f}(\overline{\delta}, \overline{r})$, où $\overline{\delta}$, $\overline{r}$ et $\overline{\pi}$ sont les vecteurs de paramètres associés respectivement au domaine δ, à l'effet r et à la politique $\pi = f(\delta, r)$.

2.2 Modèles directs, modèles inverses

Les effets produits par les politiques exécutées par le robot peuvent être décrits à l'aide de ce que l'on appelle des *modèles directs*, qui sont des modèles prédictifs permettant d'associer à des causes — ici une politique et le domaine depuis lequel elle est exécutée — les effets que celles-ci produiront (voir Figure 2.2). Plus formellement, on note $\mathscr{R}$ l'espace des effets, $\mathscr{D}$ l'espace des domaines, $\mathscr{P}$ l'espace des politiques, et $\mathcal{D}_{\mathscr{R}}$ l'espace des distributions de probabilités sur $\mathscr{R}$. Un modèle direct se définit alors de la façon suivante :

Définition 2.3. On appelle *modèle direct* toute fonction f_{dir} qui est de la forme $f_{dir} : DP \subseteq \mathscr{D} \times \mathscr{P} \to \mathcal{D}_{\mathscr{R}}$ et qui associe à chaque couple $(\delta, \pi) \in DP$ la distribution de probabilités sur l'espace des effets $\mathbb{P}_r = f_{dir}(\delta, \pi)$ résultant de l'exécution de la politique π depuis le domaine δ.

La notion de modèles directs est généralement associée à la notion duale de *modèles inverses*, c'est-à-dire de modèles prédictifs permettant d'associer à des effets des causes pouvant les avoir produits. Plus formellement ici :

Définition 2.4. On appelle *modèle inverse* toute fonction f_{inv} qui est de la forme $f_{inv} : R \subseteq \mathscr{R} \to \mathscr{D} \times \mathscr{P}$ et qui associe à chaque effet $r \in R$ un couple $(\delta, \pi) = f_{inv}(r)$ pouvant produire l'effet r.

Dans un contexte plus opérationnel, un modèle inverse se définit cependant un peu différemment : il s'agit alors d'un modèle prédictif permettant d'associer à un effet désiré une action ou politique permettant de produire cet effet depuis des conditions fixées (voir Figure 2.3). On préférera donc ici la définition suivante :

Définition 2.5. On appelle *modèle inverse* toute fonction f_{inv} qui est de la forme $f_{inv} : DR \subseteq \mathscr{D} \times \mathscr{R} \to \mathscr{P}$ et qui associe à chaque couple $(\delta, r) \in DR$ une politique $\pi = f_{inv}(\delta, r)$ permettant de produire l'effet r depuis le domaine δ.

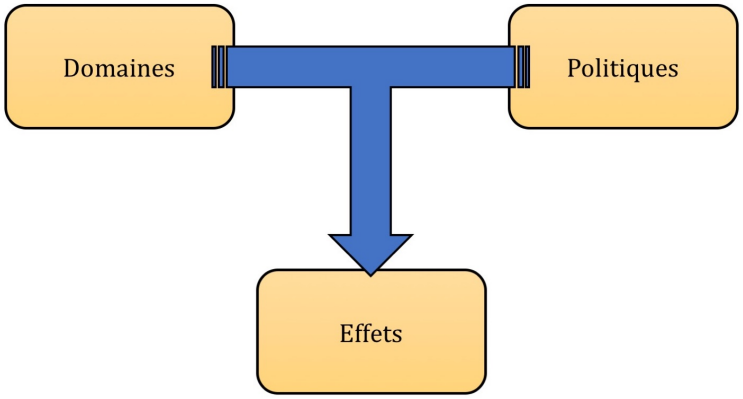


FIGURE 2.2 – Un modèle direct décrit la structure reliant l'espace des domaines, l'espace des politiques et l'espace des effets. Pour chaque couple (δ, π) constitué d'un domaine et d'une politique, il fournit une prédiction du résultat de l'exécution de cette politique depuis ce domaine, sous la forme d'une distribution de probabilité dans l'espace des effets.

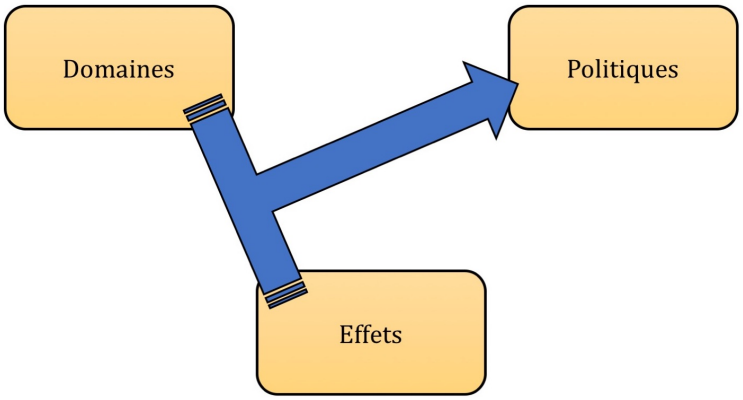


FIGURE 2.3 – Un modèle inverse décrit partiellement la structure reliant l'espace des domaines, l'espace des politiques et l'espace des effets. Pour chaque couple (δ, r) constitué d'un domaine et d'un effet désiré, il détermine une des politiques dont l'exécution depuis ce domaine permet de produire l'effet désiré.

Considérons une compétence paramétrée telle que décrite en définition 2.2. On remarque alors que sa fonction de résolution — qui est donc une fonction f telle que pour tout domaine δ et tout effet r, le triplet $(\delta, f(\delta, r), r)$ est une compétence, c'est-à-dire telle que $f(\delta, r)$ est une politique qui permet de

produire r depuis δ — est un modèle inverse. Réciproquement, chaque instance (δ, π, r) d'un modèle inverse constitue une compétence au sens de la définition 2.1, et il est donc possible de construire une compétence paramétrée dont ce modèle inverse est la fonction de résolution. L'apprentissage de compétences paramétrées revient par conséquent à identifier quelles compétences sont des instances du même modèle inverse, puis à estimer chaque modèle à partir de ses instances et à déterminer les espaces de domaines, de politiques et d'effets correspondants (domaine de définition dans lequel il est valide et image de celui-ci) pour les associer au sein d'une compétence paramétrée. Dans toute la suite, on parlera indifféremment de fonction de résolution d'une compétence paramétrée et de modèle inverse associé à celle-ci.

Il est à noter que pour un couple domaine-effet (δ, r) donné, il peut exister diverses politiques — voire une infinité — permettant de produire r depuis δ (*cf.* Figure 2.4). Par exemple, pour déplacer un objet au sol depuis une position initiale $p_{initiale}$ jusqu'à une position finale p_{finale}, l'agent peut pousser l'objet en ligne droite jusqu'à sa position finale, mais il peut aussi lui faire suivre n'importe quelle autre trajectoire se terminant en p_{finale} — y compris celles où l'agent ramasse l'objet avant de le poser en p_{finale}, celles où il lance l'objet, celles où il fait plusieurs fois le tour de p_{finale} avant d'y poser l'objet, et même celles où il fait une pause pour jongler avec l'objet avant de l'amener à sa position finale. Il n'y a donc pas unicité des modèles inverses, et même un modèle inverse associant une politique à chaque couple $(\delta, r) \in \mathscr{D} \times \mathscr{R}$ ne rend généralement compte que d'une infime partie de la structure reliant l'espace des domaines, l'espace des politiques et l'espace des effets.

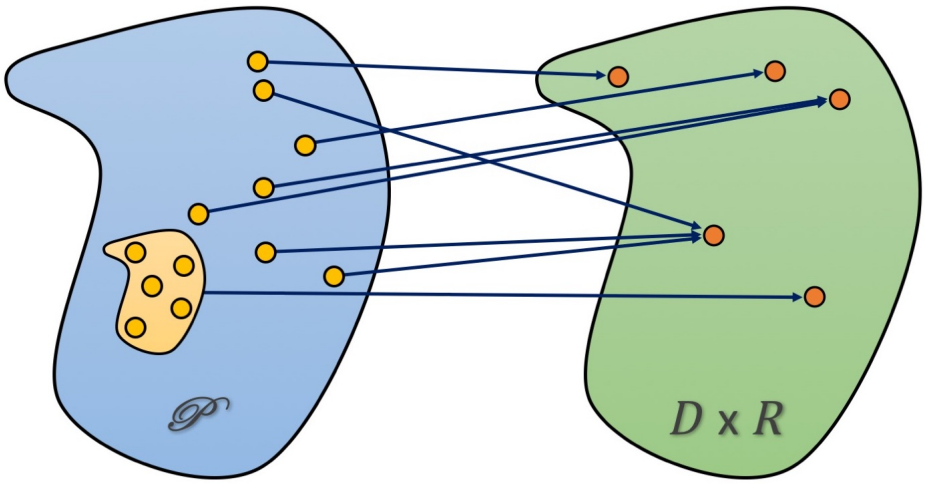


FIGURE 2.4 – Pour un couple domaine-effet $(\delta, r) \in D \times R \subseteq \mathscr{D} \times \mathscr{R}$ donné, diverses politiques de $\mathscr{P}$ (voire une infinité) peuvent produire l'effet r depuis le domaine δ.

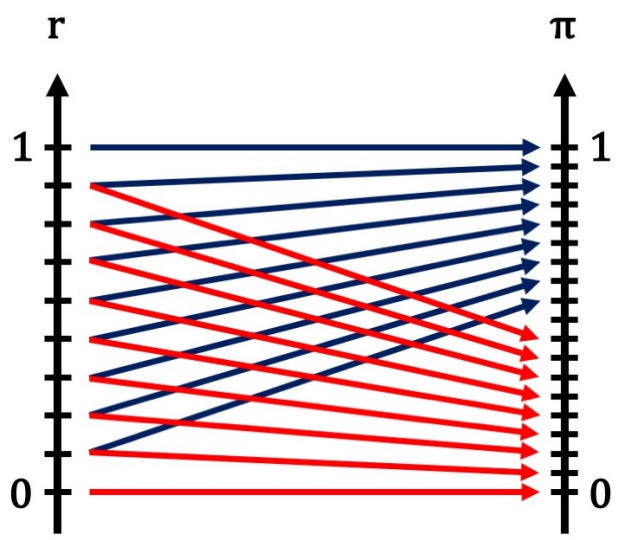


FIGURE 2.5 – On se place dans un cas où le domaine est fixé, et où les effets et les politiques sont paramétrés dans [0,1]. Les compétences (flèches) associent les divers effets aux politiques qui les ont produites. Les compétences rouges peuvent ici être généralisées en une compétence paramétrée dont le modèle inverse associé est $\overline{\pi} = \frac{\overline{r}}{2}$. Similairement, les compétences bleues peuvent être généralisées en une compétence paramétrée dont le modèle inverse associé est $\overline{\pi} = \frac{\overline{r}+1}{2}$. Les compétences rouges ne peuvent cependant pas être généralisées avec les compétences bleues. Si l'on restreint par exemple l'exploration à des politiques où $\overline{\pi}$ est inférieur à $\frac{1}{2}$, on peut se placer dans une situation où l'on ne manipule que des compétences rouges, généralisables entre elles. On se prive cependant de la possibilité d'apprendre des compétences bleues, dont on pourrait avoir l'usage plus tard (par exemple dans une situation où des contraintes extérieures font que seules les politiques correspondant à des valeurs de $\overline{\pi}$ supérieures à $\frac{1}{2}$ peuvent être exécutées).

Regrouper des compétences en une compétence paramétrée — ou des compétences paramétrées en une compétence paramétrée plus générale — signifie que ces compétences doivent être des cas particuliers couverts par la compétence paramétrée qui les généralise. Cependant, la non-unicité des modèles inverses implique que deux compétences apprises indépendamment n'ont a priori pas de raisons d'être des cas particuliers d'une même compétence plus générale. Ceci est illustré en figure 2.5.

Un des enjeux de l'apprentissage de compétences paramétrées est donc généralement de trouver des moyens de contraindre les compétences apprises par l'agent à être des cas particuliers des mêmes compétences paramétrées, par exemple en se limitant aux politiques atteignant les tâches ou buts de façon optimale selon un critère donné, ou en restreignant directement l'espace des politiques dans lequel on cherche des solutions (par exemple les expériences

de da Silva *et al.* [2012] sont réalisées sur un bras robotique sous-actionné, ce qui réduit la taille de l'espace des politiques résolvant la tâche). Ces moyens restent cependant insatisfaisants car ils apportent une dimension arbitraire à la recherche de compétences, et peuvent éliminer d'office des compétences qui auraient pu être très utiles.

2.3 Expérience de pensée

Avant de présenter formellement l'approche que nous développons dans cette thèse, nous allons dans cette section essayer, à l'aide d'une expérience de pensée, de donner une intuition des éléments sur lesquels elle repose.

2.3.1 Situation

Un agent déplace un actionneur sur une table le long d'un axe horizontal x, comme illustré en Figure 2.6. Sur cette table se trouve un objet dont on note x_{objet} la position à tout instant, initialement situé en $x_{initial}$. Les politiques que l'agent effectue mettent en mouvement l'actionneur en direction de l'objet et du bord de la table (situé plus loin en x_{bord}), jusqu'à une position finale $x_{actionneur}$. On supposera que le mouvement de l'actionneur se fait uniquement le long de l'axe horizontal x, toujours dans le sens des x croissants, et que ce mouvement est suffisamment lent pour que la vitesse de l'actionneur n'influe pas sur la position finale de l'objet.

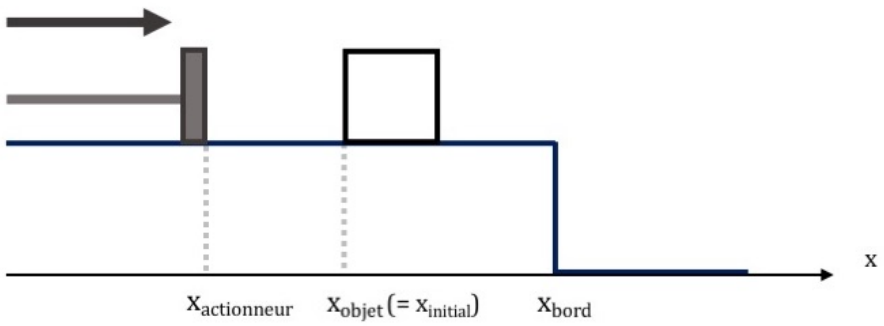


FIGURE 2.6 – Expérience de pensée : un agent déplace un actionneur sur une table le long d'un axe horizontal, en direction d'un objet et du bord de la table situé plus loin.

2.3.2 Formalisme, modèle direct

En utilisant le formalisme développé précédemment, les politiques forment ici un espace $\mathscr{P}$ paramétré par la position $x_{actionneur}$. Les effets produits par

ces politiques forment quant à eux un espace $\mathscr{R}$ paramétré par la valeur de x_{objet} après l'exécution de la politique, tandis que les domaines depuis lesquels sont exécutés ces politiques forment un espace $\mathscr{D}$ paramétré par le couple $(x_{initial}, x_{bord})$.

La figure 2.7 représente un modèle direct global de notre expérience de pensée, que l'on suppose ici déterministe. Pour faciliter la visualisation, on y suppose les valeurs de $x_{initial}$ et x_{bord} fixées (ce qui revient à considérer que l'espace des domaines est réduit à un unique élément), tout modèle direct revenant alors à une fonction prédisant x_{objet} en fonction de $x_{actionneur}$.

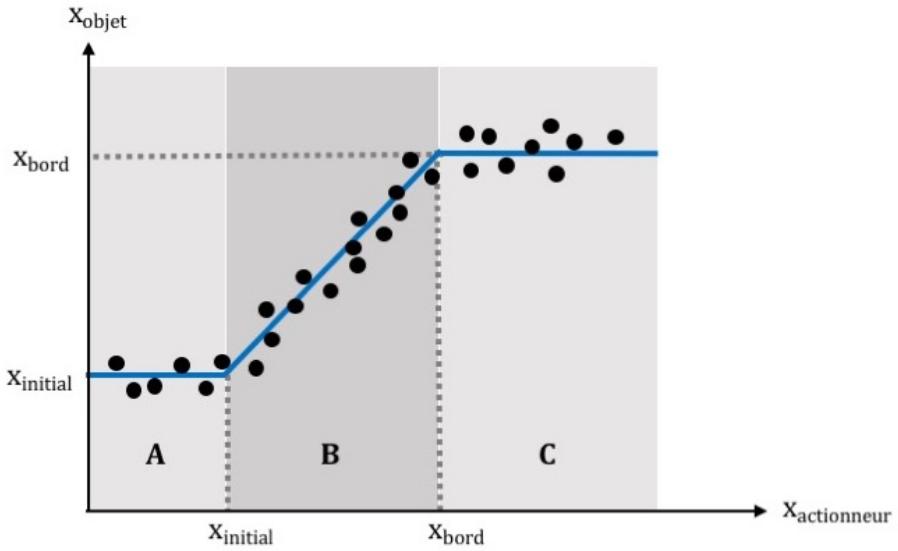


FIGURE 2.7 – En bleu : modèle direct déterministe prédisant x_{objet} en fonction de $x_{actionneur}$ dans le cadre de l'expérience de pensée décrite en section 2.3.1, pour un domaine $(x_{initial}, x_{bord})$ fixé. Les points noirs représentent les valeurs que l'on observerait en pratique en raison de l'imprécision sur les différentes valeurs. On peut distinguer trois sous-espaces de $\mathscr{D} \times \mathscr{P} \times \mathscr{R}$, correspondant respectivement à des paramètres tels que $x_{objet} = x_{initial}$ avec $x_{actionneur} < x_{initial}$ (sous-espace A, où l'actionneur n'atteint pas l'objet), tels que $x_{objet} \in [x_{initial}, x_{bord}]$ avec $x_{initial} \leq x_{actionneur} \leq x_{bord}$ (sous-espace B, où l'actionneur pousse le cube sur la table), et tels que $x_{objet} = x_{bord}$ avec $x_{bord} < x_{actionneur}$ (sous-espace C, où l'actionneur fait tomber le cube de la table).

2.3.3 Modèles inverses et généralisation

À la suite d'une phase d'apprentissage, diverses compétences individuelles (δ, π, r) sont apprises. Ces compétences appartiennent à un espace de compé-

tences $\mathfrak{C} \subseteq \mathscr{D} \times \mathscr{P} \times \mathscr{R}$. On peut alors classer ces compétences en trois catégories, selon le sous-espace de $\mathfrak{C}$ auquel elles appartiennent (*cf.* Figure 2.7) :

— dans le sous-espace où $x_{actionneur} \leq x_{initial}$ (sous-espace A), pour un domaine δ donné, toutes les politiques produisent le même effet r tel que $x_{objet} = x_{initial}$,

— dans le sous-espace où $x_{initial} \leq x_{actionneur} \leq x_{bord}$ (sous-espace B), pour un domaine δ donné, toutes les politiques produisent un effet r distinct tel que $x_{objet} = x_{actionneur}$,

— dans le sous-espace où $x_{bord} \leq x_{actionneur}$ (sous-espace C), pour un domaine δ donné, toutes les politiques produisent le même effet r tel que $x_{objet} = x_{bord}$.

On remarque alors que les compétences du sous-espace B peuvent être généralisées en une compétence paramétrée dont la fonction de résolution est le modèle inverse $f_{inv} : (\delta, r) \rightarrow \pi$ correspondant dans l'espace des paramètres à la fonction $\overline{f_{inv}}$ telle que $\overline{f_{inv}}(x_{initial}, x_{bord}, x_{objet}) = x_{objet}$ pour tout triplet $(x_{initial}, x_{bord}, x_{objet})$ vérifiant $x_{initial} \leq x_{objet} \leq x_{bord}$.

En revanche, les compétences du sous-espace A produisent toutes le même effet, et ne peuvent donc correspondre à un même modèle inverse. Ces compétences ne peuvent par conséquent pas être généralisées en une unique compétence paramétrée dont elles seraient des cas particuliers, et il en va de même pour les compétences du sous-espace C. On retrouve donc bien ici la difficulté mentionnée en section 2.2 : deux compétences apprises indépendamment n'ont a priori pas de raisons d'être des cas particuliers d'une même compétence plus générale.

2.3.4 Une nouvelle forme de généralisation

Nous avons remarqué précédemment (*cf.* section 2.2) que les fonctions de résolution des compétences paramétrées constituaient des modèles inverses. Nous en avons déduit que la généralisation de compétences en compétences paramétrées revenait à identifier quelles compétences sont des instances du même modèle inverse, puis à estimer chaque modèle à partir de ses instances et à déterminer les espaces de domaines, de politiques et d'effets correspondant.

Ce lien entre les compétences paramétrées et les modèles inverses nous ouvre à présent une nouvelle voie de généralisation pour les compétences : s'il existe une dualité entre les modèles inverses et les modèles directs, ne pourrait-on pas construire une forme duale de compétences paramétrées fondées sur les modèles directs ?

En effet, lorsque l'on considère un modèle direct $f_{dir} : (\delta, \pi) \rightarrow r$, chaque instance de f_{dir} forme un triplet (δ, π, r), qui constitue une compétence. Il est donc possible d'identifier quelles compétences sont des instances du même modèle direct, puis d'estimer chaque modèle à partir de ses instances et de déterminer les espaces de domaines, de politiques et d'effets correspondant pour

les associer au sein d'une compétence paramétrée. Nous appellerons ce nouveau type de compétences paramétrées (que nous définirons plus formellement en section 2.4.1) des *compétences paramétrées directes*. Nous les distinguerons des compétences paramétrées que nous avons vues jusqu'ici, que nous appellerons désormais *compétences paramétrées inverses*.

Nous allons donc maintenant nous intéresser à la question de la généralisation à l'aide non pas de modèles inverses mais de modèles directs déterministes. Les difficultés vues précédemment se retrouvent-elles à nouveau ? Considérons les trois sous-espaces vus précédemment :

— dans le sous-espace A, une généralisation devient possible avec un modèle direct $f_{dir}^{A} : (\delta, \pi) \rightarrow r$ correspondant dans l'espace des paramètres à la fonction $\overline{f_{dir}^{A}}$ telle que $\overline{f_{dir}^{A}}(x_{initial}, x_{bord}, x_{actionneur}) = x_{initial}$,

— dans le sous-espace B, une généralisation reste possible avec un modèle direct $f_{dir}^{B} : (\delta, \pi) \rightarrow r$ correspondant dans l'espace des paramètres à la fonction $\overline{f_{dir}^{B}}$ telle que $\overline{f_{dir}^{B}}(x_{initial}, x_{bord}, x_{actionneur}) = x_{actionneur}$,

— dans le sous-espace C, une généralisation devient possible avec un modèle direct $f_{dir}^{C} : (\delta, \pi) \rightarrow r$ correspondant dans l'espace des paramètres à la fonction $\overline{f_{dir}^{C}}$ telle que $\overline{f_{dir}^{C}}(x_{initial}, x_{bord}, x_{actionneur}) = x_{bord}$.

Des modèles directs peuvent donc être appris pour chacun des sous-espaces, et les compétences correspondantes peuvent donc être généralisées localement en compétences paramétrées directes. Ce qui rend ces généralisations possibles est le fait que, dans le cas de modèles directs déterministes, tout couple domaine-politique (δ, π) correspond à un unique effet r produit par l'exécution de π depuis δ. Les difficultés liées à la non-unicité n'ont donc pas lieu d'être, et les généralisations qui n'étaient pas possibles avec les modèles inverses le deviennent.

2.3.5 Pertinence

Nous venons de voir que l'apprentissage de compétences paramétrées s'appuyant sur des modèles directs locaux tels que f_{dir}^{A}, f_{dir}^{B} et f_{dir}^{C} n'est pas confronté à certaines des difficultés que l'on rencontrerait dans le cadre d'une approche s'appuyant sur des modèles inverses. Mais les compétences paramétrées directes sont-elles tout aussi intéressantes à apprendre que les compétences paramétrées inverses ?

Approche directe

Remarquons tout d'abord que de même que les modèles inverses, les modèles directs peuvent être utilisés pour de la planification (ce que l'on trouve par exemple dans les travaux de Ghadirzadeh *et al.* [2016]). La question qui se pose alors est celle de ce qu'apporte une approche directe par rapport à une approche indirecte, ou plus précisément celle de savoir si les compétences

paramétrées directes permettent d'obtenir de meilleures performances pour la résolution de tâches que les compétences paramétrées inverses.

Considérons donc le cas d'un robot qui a appris diverses compétences individuelles durant une phase d'entraînement, compétences qu'il a ensuite généralisées lorsque cela était possible en compétences paramétrées (au sens direct ou inverse). Un utilisateur charge alors ce robot d'accomplir une tâche complexe composée d'une succession de sous-tâches, dont l'une se déroule dans un cadre identique à notre expérience de pensée (mais avec $x_{initial}$ et x_{bord} fixés) et consiste à placer l'objet en x_{bord}.

Pour résoudre cette sous-tâche, l'agent va devoir exécuter une politique. Il doit donc déterminer quelles sont les compétences (compétences individuelles ou instances de compétences paramétrées) correspondant à l'effet r escompté et au domaine δ dans lequel il se trouve. Si plusieurs solutions sont possibles, il pourra ensuite choisir celle dont il exécutera la politique, généralement en fonction d'un critère de qualité pertinent pour la situation (*e.g.* , celle consommant le moins d'énergie, celle amenant l'actionneur le plus loin possible en vue d'une prochaine sous-tâche, etc).

Pour la résolution de notre sous-tâche, supposons dans un premier temps que la généralisation des compétences a été faite via une approche inverse. Les compétences du sous-espace B vues à l'entraînement ont pu être généralisées en une unique compétence paramétrée inverse dont le modèle inverse associé est f_{inv}, qui fournit une unique solution $(\delta, f_{inv}(\delta, r), r)$. Les compétences du sous-espace C vues à l'entraînement n'ont quant à elle pas pu être généralisées en une unique compétence paramétrée inverse (une généralisation est possible par rapport aux domaines mais pas par rapport à l'effet car elle correspondent toutes au même effet r, c'est-à-dire à une position finale de l'objet en x_{bord}). Le robot a donc à sa disposition un florilège de compétences individuelles et de compétences paramétrées inverses permettant de produire l'effet r, dont une partie seulement fournit une solution (et une seule) pour le domaine δ. C'est par conséquent dans un ensemble de compétences assez restreint que l'agent doit choisir celle d'entre elles qui est la meilleure pour le critère retenu.

La situation est en revanche très différente si la généralisation a été faite via une approche directe : s'il n'existe là-encore qu'une unique solution dans le sous-espace B, les différentes compétences de C vues à l'entraînement ont cette fois pu être réunies en une unique compétence paramétrée directe, associée au modèle direct f_C. L'agent fait donc cette fois-ci son choix parmi l'ensemble des instances de celle-ci pour le domaine δ, soit tout un continuum. Ce continuum inclut notamment toutes les solutions fournies par les compétences paramétrées inverses vues précédemment, mais va bien au-delà.

Lorsque l'agent choisit la compétence optimale pour le critère retenu, l'ensemble dans lequel il peut faire ce choix est plus riche dans le cas direct (car il contient son analogue pour le cas inverse) et l'optimum ainsi trouvé sera par conséquent au moins aussi bon. Une approche par les modèles directs a donc ici

un avantage significatif sur une approche par les modèles inverses, un avantage qui est encore plus net quand l'agent est soumis à des contraintes sur les politiques qu'il peut adopter (*e.g.* , $x_{actionneur}$ restreint à un certain intervalle de valeurs) : l'ensemble des compétences dans lequel se fait le choix pourrait alors s'avérer vide pour une approche inverse mais pas pour une approche directe.

Un désavantage qu'ont cependant les approches directes par rapport aux approches inverses est qu'elles requièrent a priori de résoudre des problèmes d'optimisation (possiblement lourds en calculs) pour choisir la compétence optimale au sein d'une compétence paramétrée, alors que pour une approche inverse celle-ci serait directement donnée par le modèle inverse. En échange de quoi, les approches directes ne sont que peu affectées par un changement de critère de sélection ou par l'ajout de contraintes — le problème d'optimisation se trouvant alors simplement modifié — alors que les approches inverses nécessitent de connaître à la fois critères et contraintes en amont de la généralisation voire de l'apprentissage des compétences individuelles.

Localité

Nous avons jusqu'ici parlé de généralisation des compétences au sein des sous-espaces A, B et C de l'espace des compétences $\mathfrak{C}$, et les compétences paramétrées directes qui peuvent être formées ainsi sont par conséquent locales à ces sous-espaces. Cette localité était justifiée dans le cas inverse par les difficultés que la non-unicité des modèles inverses entraînait pour la généralisation, un unique modèle inverse ne pouvant capturer à lui seul la structure de $\mathfrak{C}$ (le modèle inverse f_{inv} par exemple, bien qu'il associe une politique à chaque élément de $\mathscr{D} \times \mathscr{R}$, ne décrit pour autant que la structure des compétences du sous-espace B sans rendre compte de celle des sous-espaces A et C). Dans le cas d'une approche directe, n'étant pas confrontés à des problèmes de non-unicité, y a t-il un toujours un intérêt à distinguer les compétences paramétrées associées aux modèles directs locaux f_{dir}^A, f_{dir}^B et f_{dir}^C plutôt que de les réunir en une compétence plus générale associée à un modèle direct global f_{dir} ?

Commençons par caractériser ce que (A, B, C) a de particulier. Dans chacun des sous-espaces A, B et C, on peut remarquer qu'une faible modification du domaine ou de la politique induit une modification faible de l'effet produit : f_{dir}^A, f_{dir}^B et f_{dir}^C sont des fonctions continues. On peut même aller plus loin et remarquer que ces fonctions sont $\mathscr{C}^\infty$, alors que le modèle global f_{dir} ne l'est pas en raison des discontinuités de la dérivée première aux frontières entre les différents sous-espaces. Partant de là, on peut caractériser (A, B, C) comme étant la partition composée des plus grands sous-espaces possibles de $\mathfrak{C}$ tels que la structure de chacun d'entre eux peut être décrite par un modèle direct $\mathscr{C}^\infty$.

Dans un premier temps, on peut noter qu'une telle partition présente un intérêt pratique : l'apprentissage supervisé s'appuie généralement sur des fa-

milles de fonctions qui sont elles-mêmes $\mathscr{C}^\infty$ (polynômes, gaussiennes, sinus, cosinus, *etc*), et il est donc plus adapté de limiter la généralisation aux sous-espaces sur lesquels de tels modèles peuvent être appris. L'approximation par une famille de fonctions $\mathscr{C}^\infty$ d'une fonction qui n'a pas cette régularité est en effet connu pour poser des difficultés (on pensera par exemple à l'approximation d'une fonction échelon par une série de Fourier, illustrée en Figure 2.8, qui reste très imparfaite même avec de nombreuses harmoniques alors que deux séries de Fourier locales pourraient approximer la fonction avec une harmonique chacune).

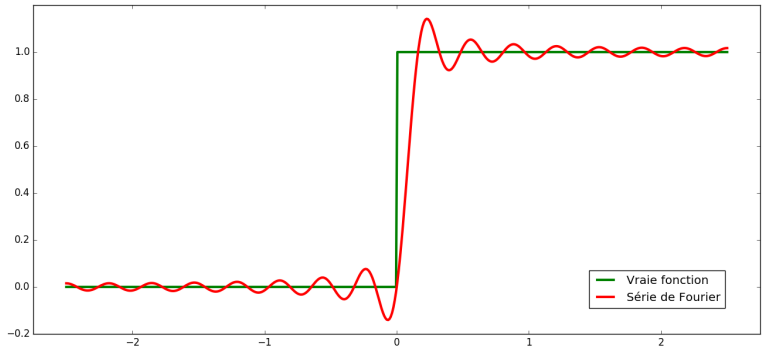


FIGURE 2.8 – Approximation de la fonction échelon d'Heaviside (en vert) par sa série de Fourier à l'ordre 30 (en rouge).

Dans un second temps, on peut remarquer que les ruptures de régularité de f_{dir} — c'est-à-dire les frontières entre les différents sous-espaces — correspondent à des changements de situation dans le monde physique : la frontière entre les compétences de A et celles de B sépare les compétences qui permettent ou non à l'actionneur de mettre l'objet en mouvement, tandis que la frontière entre les compétences de B et celles de C est une frontière entre les compétences qui font tomber ou non l'objet de la table. On constate alors que partitionner $\mathfrak{C}$ en (A, B, C) présente l'avantage de mener à des compétences paramétrées directes ayant un réel sens sémantique : les compétences de A se généralisent donc ainsi en une compétence $\lceil$*ne pas toucher le cube*$\rfloor$, celles de B en une compétence $\lceil$*pousser le cube sur la table sans le faire tomber*$\rfloor$, et celles de C en une compétence $\lceil$*pousser le cube jusqu'à le faire tomber de la table*$\rfloor$.

2.4 Formalisation de l'approche

Nous allons maintenant poser plus formellement l'approche que nous développons dans cette thèse.

2.4.1 Quelques définitions

On appelle *compétence paramétrée inverse* une compétence paramétrée au sens de la définition 2.2, sa fonction de résolution constituant alors un modèle inverse. On définit par ailleurs une *compétence paramétrée directe* de la façon suivante :

Définition 2.6. Soient $\mathcal{D}$ un espace continu de domaines reliés entre eux et paramétrés dans un espace $\overline{\mathcal{D}} \subseteq \mathbb{R}^{|\mathcal{D}|}$, $\mathcal{P}$ un espace continu de politiques reliées entre elles et paramétrées dans un espace $\overline{\mathcal{P}} \subseteq \mathbb{R}^{|\mathcal{P}|}$, et $\mathcal{R}$ un espace continu d'effets reliés entre eux et paramétrés dans un espace $\overline{\mathcal{R}} \subseteq \mathbb{R}^{|\mathcal{R}|}$. On appelle *compétence paramétrée directe* tout quadruplet $(\mathcal{D}, \mathcal{P}, \mathcal{R}, g)$, où la fonction $g : \mathcal{D} \times \mathcal{P} \to \mathcal{R}$ est une fonction qui associe à chaque instance d'un domaine paramétré et chaque instance d'une politique paramétrée un effet produit par cette politique depuis ce domaine, c'est-à-dire une fonction telle que pour tout domaine $\delta \in \mathcal{D}$ et toute politique $\pi \in \mathcal{P}$, le triplet $(\delta, \pi, g(\delta, \pi))$ est une compétence au sens de la définition 2.1. On se référera indifféremment à cette fonction g (qui constitue un modèle direct déterministe) et à la fonction $\overline{g} : \overline{\mathcal{D}} \times \overline{\mathcal{P}} \to \overline{\mathcal{R}}$ qui lui correspond dans les espaces de paramètres, c'est-à-dire la fonction telle que $\forall(\delta, \pi) \in \mathcal{D} \times \mathcal{P},\ \overline{r} = \overline{g}(\overline{\delta}, \overline{\pi})$, où $\overline{\delta}$, $\overline{\pi}$ et $\overline{r}$ sont les vecteurs de paramètres associés respectivement au domaine δ, à la politique π et à l'effet $r = g(\delta, \pi)$.

2.4.2 Hypothèses de travail

Soient $\mathscr{D}$ l'espace des domaines, $\mathscr{P}$ l'espace des politiques, et $\mathscr{R}$ l'espace des effets. Soient également $\overline{\mathscr{D}} \subseteq \mathbb{R}^{|\mathscr{D}|}$, $\overline{\mathscr{P}} \subseteq \mathbb{R}^{|\mathscr{P}|}$, et $\overline{\mathscr{R}} \subseteq \mathbb{R}^{|\mathscr{R}|}$ des paramétrisations de ces espaces.

Hypothèse 1. On suppose dans cette thèse que le modèle direct global stochastique $f_{sto} : \overline{\mathscr{D}} \times \overline{\mathscr{P}} \mapsto \overline{\mathscr{R}}$, qui décrit les interactions du robot avec son environnement, est de la forme $f_{sto} = f_{dir} + \Delta$, où $f_{dir} : \overline{\mathscr{D}} \times \overline{\mathscr{P}} \mapsto \overline{\mathscr{R}}$ est un modèle direct global déterministe tel qu'il existe une partition qui découpe $\overline{\mathscr{D}} \times \overline{\mathscr{P}}$ en sous-espaces sur lesquels f_{dir} est $\mathscr{C}^\infty$, et où $\Delta : \overline{\mathscr{D}} \times \overline{\mathscr{P}} \mapsto \overline{\mathscr{R}}$ est une fonction stochastique décrivant l'incertitude liée à l'imprécision des mesures.

Hypothèse 2. On suppose également qu'il existe une fonction déterministe $B : \overline{\mathscr{D}} \times \overline{\mathscr{P}} \mapsto \overline{\mathscr{R}}$ — qu'il est possible de connaître ou d'estimer — qui décrit la précision des mesures et borne Δ, c'est-à-dire qui vérifie pour toute dimension i de $\overline{\mathscr{R}}$ l'inégalité suivante : $\forall(\overline{\delta}, \overline{\pi}) \in \overline{\mathscr{D}} \times \overline{\mathscr{P}}, \quad \left|\Delta^{(i)}(\overline{\delta}, \overline{\pi})\right| \leq B^{(i)}(\overline{\delta}, \overline{\pi})$.

2.4.3 Discussion des hypothèses

L'hypothèse 1 attribue la totalité de la variabilité de f_{sto} à un bruit sur les mesures. Il ne s'agit généralement pas d'une hypothèse très forte, mais il faut cependant noter qu'elle implique qu'avec une précision infinie, l'exécution

d'une même politique depuis un même domaine produirait toujours le même effet. La validité de cette hypothèse reposera donc sur le fait que le robot interagit avec un monde déterministe (à notre échelle), et sera fortement liée à la variabilité intrinsèque des politiques que peut exécuter l'agent.

Au-delà de l'existence d'un modèle direct global déterministe f_{dir} sous-jacent à f_{sto}, l'hypothèse 1 affirme également l'existence d'une partition qui découpe $\overline{\mathscr{D}} \times \overline{\mathscr{P}}$ en sous-espaces sur lesquels f_{dir} est $\mathscr{C}^\infty$. Cette affirmation repose sur l'idée que la régularité f_{dir} reflète celle du monde avec lequel interagit le robot, et que les frontières entre les différents sous-espaces correspondent à des changements concrets dans la nature de l'interaction (ex : objet atteint par l'actionneur, bord de la table, *etc*).

Si l'hypothèse 1 attribue la totalité de la variabilité de f_{sto} à un bruit sur les mesures, l'hypothèse 2 va plus loin et affirme l'existence d'une borne sur la variabilité de Δ donc de f_{sto}. Cette hypothèse contraste avec l'hypothèse habituelle d'un bruit gaussien, mais reste moins forte que celle-ci car elle ne prétend pas une connaissance complète de la distribution suivie par Δ. Elle exclue en revanche la possibilité de points aberrants (*outliers* en anglais), que l'on sait arriver en pratique. On suppose dans cette approche que les points aberrants sont éliminés en amont ou en aval de l'apprentissage (*e.g.* , points trop différents des autres points dans leur voisinage). En particulier, bien que nous ne nous ne soyons pas intéressés à cette question dans le cadre de cette thèse, il est probable que les points aberrants se distinguent par leur capacité de généralisation, auquel cas il serait possible de formuler une extension de l'algorithme que nous présenterons au chapitre 4 qui s'appuierait sur ce point pour les identifier et les éliminer.

Bien que nous introduisions l'hypothèse 2, nous n'en aurons réellement besoin qu'au chapitre 4. On en retiendra pour l'instant que l'interaction de l'agent avec son environnement résulte en un quadruplet $(\overline{\delta}, \overline{\pi}, \overline{r}, \varepsilon)$, composé du domaine dans lequel se trouvait le robot au début de cette interaction, de la politique qu'il a exécutée, de l'effet qui a résulté de l'exécution de celle-ci, et de la variabilité de cet effet liée à l'imprécision des mesures (ε étant donc une estimation de $B(\overline{\delta}, \overline{\pi})$).

2.4.4 Cœur de l'approche

En nous appuyant sur l'hypothèse 1, nous pouvons remarquer que notre problématique, qui vise à réunir les compétences individuelles apprises par l'agent en compétences les plus générales possibles, peut être formulée comme un problème de régression vers une fonction localement $\mathscr{C}^\infty$: la régression vers f_{dir}. En effet, une fois f_{dir} approximée, les restrictions de celle-ci aux plus grands sous-espaces sur lesquels elle est $\mathscr{C}^\infty$ pourront être associées à leur domaine de définition et à l'image de ceux-ci au sein de compétences paramétrées.

Plus concrètement, il s'agira donc de rechercher une partition découpant l'espace des compétences $\mathfrak{C} \subseteq \mathscr{D} \times \mathscr{P} \times \mathscr{R}$ en sous-espaces les plus grands possibles sur lesquels des modèles directs locaux $\mathscr{C}^\infty$ peuvent être formulés et d'estimer de ces modèles.

2.5 Conclusion

Dans ce chapitre, nous avons adapté le formalisme des compétences et des compétences paramétrées au cas d'un robot qui se fixe ses propres buts durant la phase d'apprentissage. Nous avons ensuite relié les compétences paramétrées au formalisme des modèles inverses — nous parlerons désormais de compétences paramétrées inverses — avant de nous servir de la dualité entre modèles inverses et modèles directs pour élaborer une nouvelle forme de compétences paramétrées, duale de celle que nous avions vue jusqu'alors : les compétences paramétrées directes.

Nous avons également montré dans ce chapitre les limites des approches inverses pour le regroupement des compétences en compétences les plus générales possibles, et illustré par une expérience de pensée ce que pourrait apporter une approche directe pour la résolution de cette problématique. Nous avons alors proposé une approche directe qui consiste, dans une cadre plus général que celui de l'expérience de pensée, à ramener ce regroupement à un problème de régression vers une fonction localement $\mathscr{C}^\infty$ (détermination des plus grands sous-espaces de $\mathfrak{C}$ sur lesquels des modèles directs $\mathscr{C}^\infty$ locaux peuvent être trouvés et estimation de ces modèles).

Chapitre 3

Les algorithmes de régression

Nous avons donc ramené notre problème à un problème de régression, et il s'agit alors de déterminer les plus grands sous-espaces du domaine de définition sur lesquels la fonction est $\mathscr{C}^\infty$ et de déterminer des approximations locales sur ces sous-espaces. Nous allons à présent passer en revue diverses approches de régression et voir si elles sont adaptées à notre cas.

3.1 Enjeu des méthodes de régression

L'enjeu des méthodes de régression est de découvrir la relation liant une variable cible t (pour *target*, signifiant "cible" en anglais), possiblement multidimensionnelle et que l'on cherche à prédire, à une autre variable x (possiblement multidimensionnelle également). On se donne pour cela un échantillon $(x_i, t_i)_{i \in [\![0,N[\![}$ de N observations conjointes des valeurs de x et t (jeu de données d'entraînement).

Dans les cas où la relation entre x et t est déterministe, elle peut être décrite par une fonction $f : x \mapsto t$. Il s'agira donc de trouver une fonction rendant compte des données d'entraînement, qui constituera une approximation de f et permettra de prédire les valeurs de t associées à des valeurs de x non encore observées. La qualité de l'approximation sera alors déterminée par la qualité des prédictions qu'elle fournit.

Lorsque la relation entre x et t est de nature stochastique, on considérera que les valeurs prises par x et t sont des tirages de variables aléatoires X et T respectivement, et leur relation sera alors décrite par la distribution de probabilités conditionnelles $\mathbb{P}(T|X)$. Il s'agira alors d'approximer cette distribution ou certaines de ses propriétés. Le plus souvent, on cherchera une fonction qui rende raisonnablement compte des données d'entraînement compte tenu de la variabilité, et qu'on utilisera comme approximation de la fonction $f : x \mapsto t = \mathbb{E}(T \,|\, X = x)$. Là encore, la qualité de l'approximation sera déterminée par la qualité des prédictions qu'elle fournit.

Dans la suite de ce chapitre, nous supposerons t unidimensionnelle pour simplifier les notations. Les approches présentées s'étendent cependant sans grande difficulté au cas d'une variable cible multidimensionnelle.

3.2 Régression linéaire

La forme la plus basique de régression est la régression vers un modèle linéaire, où l'on suppose que t est une combinaison linéaire des composantes de x : $t = \beta^T x$. Il existe alors de nombreuses approches pour choisir β.

Bien que les algorithmes de régression linéaire puissent sembler complètement inadaptés à l'approximation de fonctions non-linéaires, ils est en réalité assez aisé de s'affranchir de cette contrainte. En effet, si on se donne une famille de fonctions $(f_j)_{j \in [\![0,M[\![}$, il suffit d'évaluer les $f_j(x_i)$ pour tout i et pour tout j puis d'utiliser n'importe quel algorithme de régression linéaire pour chercher une solution sous la forme :

$$t = \sum_j w_j \cdot f_j(x)$$

On pourra ainsi par exemple réaliser une régression polynomiale (Gergonne [1974]) en choisissant comme f_j les différents termes d'un polynôme (les w_j étant alors les coefficients de celui-ci). On pourra également chercher une approximation comme combinaison linéaire de termes sinus et cosinus (similairement à une série de Fourier), comme une somme de termes exponentiels, *etc.* Lorsque nous parlerons de régression linéaire dans ce chapitre, il faudra donc garder en tête que chacune d'entre elle peut être étendue à une régression vers des combinaisons linéaires de fonctions pour gagner en expressivité.

Nous allons à présent passer quelques approches de régression linéaire en revue. Bien que la régression linéaire soit insuffisante pour traiter notre problème (ce que nous verrons plus loin), de nombreux algorithmes plus élaborés et auxquels nous nous intéresserons se reposent sur celle-ci et y font directement appel. Présenter ces approches nous permettra également d'introduire certaines problématiques qui reviendront par la suite.

3.2.1 Moindres carrés

Pour un vecteur β donné, on appelle *résidus* les écarts en valeur absolue entre les valeurs des t_i du jeu d'entraînement et celles prédites à partir des x_i correspondants :

$$r_i = \left| t_i - \beta^T x_i \right|$$

Posons M_x la matrice dont les lignes sont les x_i, et v_t le vecteur des t_i. Lorsque le problème est déterministe et que le jeu d'entraînement est assez grand, il existe une unique valeur de β telle que tous les résidus sont nuls.

Celle-ci est donc l'unique solution du système d'équations $v_t = M_x\beta$, qui peut donc être simplement trouvée en inversant M_x :

$$\hat{\beta} = M_x^{-1} v_t$$

Lorsque le problème n'est pas déterministe (par exemple en raison d'un bruit stochastique), ce système d'équation peut devenir sur-contraint et ne pas admettre de solution. On peut toutefois remarquer que l'égalité $v_t = M_x\beta$ implique $M_x^T v_t = M_x^T M_x \beta$. où $M_x^T M_x$ est alors inversible. Cette seconde équation admet alors une unique solution :

$$\hat{\beta} = (M_x^T M_x)^{-1} M_x^T v_t$$

Il est possible de prouver que dans les deux cas, $\hat{\beta}$ est la valeur de β qui minimise la somme des carrés des résidus :

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_i r_i^2 = \underset{\beta}{\operatorname{argmin}} \sum_i (t_i - \beta^T x_i)^2$$

$\hat{\beta}$ est donc une valeur de β relativement simple à calculer et minimisant une mesure d'erreur (la somme des carrés des résidus) sur le jeu de données. Ceci en fait par conséquent un choix très courant pour la valeur de β. Un exemple en est donnée en Figure 3.1.

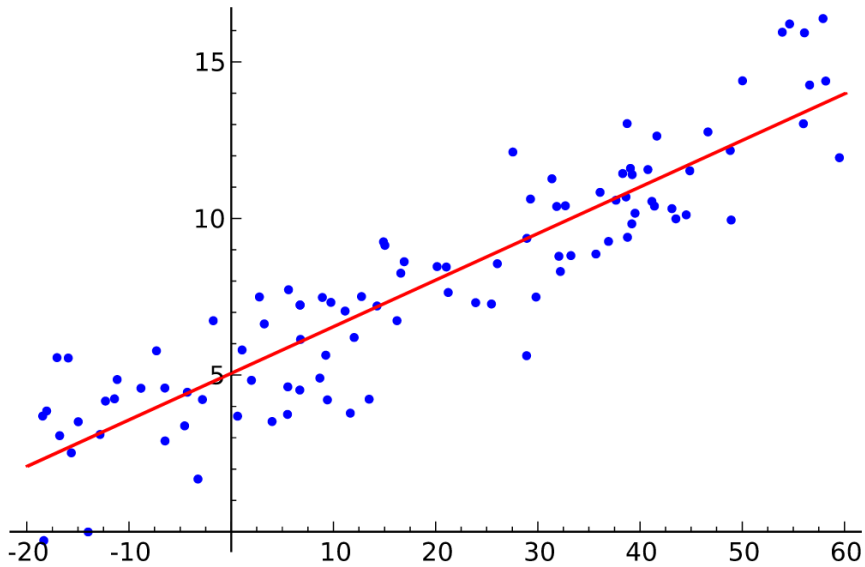


FIGURE 3.1 – Exemple de régression linéaire. Les points bleus forment le jeu de données d'entraînement. Il permettent via la méthode des moindres carrés de déterminer les paramètres du modèle, et d'obtenir ainsi le modèle linéaire en rouge.

3.2.2 Maximum de vraisemblance

Faisons l'hypothèse que toute l'indétermination sur les valeurs des t_i est le résultat d'un bruit stochastique, le bruit sur chaque point de données étant indépendant de celui sur les autres points de données et tiré suivant une même distribution gaussienne centrée $\mathcal{N}(0, \sigma)$.

La *vraisemblance* d'une valeur que peut prendre un paramètre est définie comme la probabilité d'un ensemble d'observations qui dépend de ce paramètre, à supposer que ce paramètre soit égal à cette valeur. Ici, la vraisemblance d'une valeur de β donnée est donc la probabilité d'observer les t_i pour cette valeur :

$$\mathbb{P}(t_0, t_1, ..., t_N, x_0, x_1, ..., x_N \mid \beta)$$

La méthode du maximum de vraisemblance consiste alors à choisir la valeur de β qui maximise cette probabilité :

$$\begin{aligned}
\hat{\beta} &= \operatorname*{argmax}_{\beta} \mathbb{P}(t_0, t_1, ..., t_N, x_0, x_1, ..., x_N \mid \beta) \\
&= \operatorname*{argmax}_{\beta} \mathbb{P}(t_0, t_1, ..., t_N \mid \beta, x_0, x_1, ..., x_N)\, \mathbb{P}(x_0, x_1, ..., x_N \mid \beta) \\
&= \operatorname*{argmax}_{\beta} \mathbb{P}(t_0, t_1, ..., t_N \mid \beta, x_0, x_1, ..., x_N)\, \mathbb{P}(x_0, x_1, ..., x_N) \\
&= \operatorname*{argmax}_{\beta} \mathbb{P}(t_0, t_1, ..., t_N \mid \beta, x_0, x_1, ..., x_N) \\
&= \operatorname*{argmax}_{\beta} \prod_i \mathbb{P}(t_i \mid \beta, x_i) \\
&= \operatorname*{argmax}_{\beta} \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\dfrac{(t_i - \beta^T x_i)^2}{2\sigma^2}} \\
&= \operatorname*{argmax}_{\beta} \prod_i e^{-\dfrac{(t_i - \beta^T x_i)^2}{2\sigma^2}} \\
&= \operatorname*{argmax}_{\beta} e^{-\dfrac{\sum_i (t_i - \beta^T x_i)^2}{2\sigma^2}} \\
&= \operatorname*{argmin}_{\beta} \sum_i (t_i - \beta^T x_i)^2
\end{aligned}$$

On remarque ici que la solution du maximum de vraisemblance coïncide avec celle de la méthode des moindres carrés, lui donnant ainsi une justification théorique.

3.2.3 Dilemme biais-variance, régression d'arête

Nous avons vu que la solution de la méthode des moindres carrés est celle qui rend le mieux compte des données d'entraînement (en tant que maximum de vraisemblance). Une difficulté qui se présente ensuite est que cette adéquation aux données d'entraînement ne garantit pas nécessairement une approximation de bonne qualité, c'est-à-dire qui fournisse des prédictions satisfaisantes hors du jeu de données d'entraînement. Ceci s'inscrit dans ce qu'on appelle le *dilemme biais-variance*, une problématique centrale dans le domaine de l'apprentissage supervisé. Voyons de quoi il s'agit et comment celui-ci peut être pris en compte.

Le dilemme

Le dilemme biais-variance se fonde sur deux constats. Le premier de ces constats est que plus un modèle est versatile, plus ses paramètres sont sensibles aux fluctuations du jeu de données d'entraînement (forte variance), et le modèle commence alors à rendre compte du bruit présent dans ces données au détriment de la relation qu'il cherche à décrire. On observe dans ce cas un fort écart entre la qualité des prédictions sur le jeu de données d'entraînement et en dehors de celui-ci (faible capacité de généralisation), et on parle alors de *sur-apprentissage* (*overfitting* en anglais).

Le second constat est que chercher à l'inverse à réduire la versatilité du modèle revient à faire des hypothèses plus fortes sur la nature de la solution, ce qui augmente donc les chances d'y introduire un biais (c'est-à-dire une erreur causée par des hypothèses erronées). Le modèle peut ainsi ne plus être suffisamment expressif pour rendre compte de la relation qu'il cherche à décrire, ce qui se traduit par une faible qualité des prédictions à la fois sur le jeu de données d'entraînement et en dehors de celui-ci. On parle alors de *sous-apprentissage* (*underfitting* en anglais).

Partant de ces deux constats, ce que l'on appelle le *dilemme biais-variance* est le problème consistant à chercher à minimiser simultanément les erreurs liées au biais et à la variance, alors même que les deux démarches sont antagonistes. Ceci est illustré en Figure 3.2, et un exemple concret du dilemme biais-variance pour l'approximation d'une fonction cosinus par une régression polynomiale est présenté en Figure 3.3.

Régularisation Tikhonov

Comment alors prendre en compte ce dilemme biais-variance ? Dans un premier temps, on peut introduire dans le processus d'apprentissage un ou plusieurs hyper-paramètres, dont la valeur détermine à quel point le modèle est versatile. Ceci peut ainsi se traduire par une limitation de la complexité du modèle (celle-ci impactant directement la variance du modèle) via des

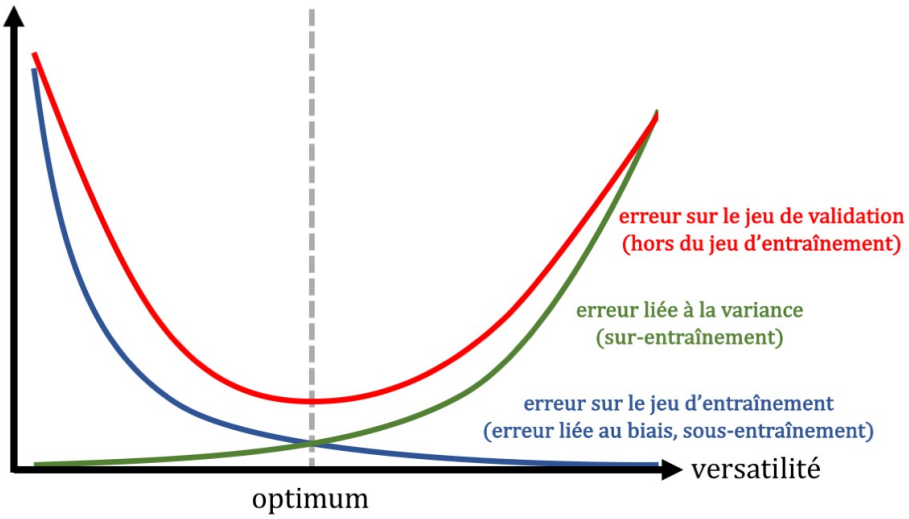


FIGURE 3.2 – Lorsque le modèle est peu versatile, on observe une forte erreur à la fois sur le jeu d'entraînement et hors de celui-ci. Il s'agit de l'erreur liée au biais, et nous sommes dans une situation de sous-entraînement. Lorsque l'on fait varier les hyper-paramètres pour augmenter la versatilité du modèle, l'erreur liée au biais ne fait que diminuer, ce que l'on retrouve sur le jeu d'entraînement, mais l'erreur liée à la variance augmente progressivement jusqu'à atteindre une situation de sur-entraînement. La qualité de l'approximation diminue donc, ce qui se manifeste par une erreur sur le jeu de validation. Entre sous-entraînement et sur-entraînement, il existe un optimum où la qualité de l'approximation est la meilleure.

hyper-paramètres définissant par exemple le degré maximum d'un polynôme, le nombre maximum d'harmoniques d'une série de Fourier, *etc.*

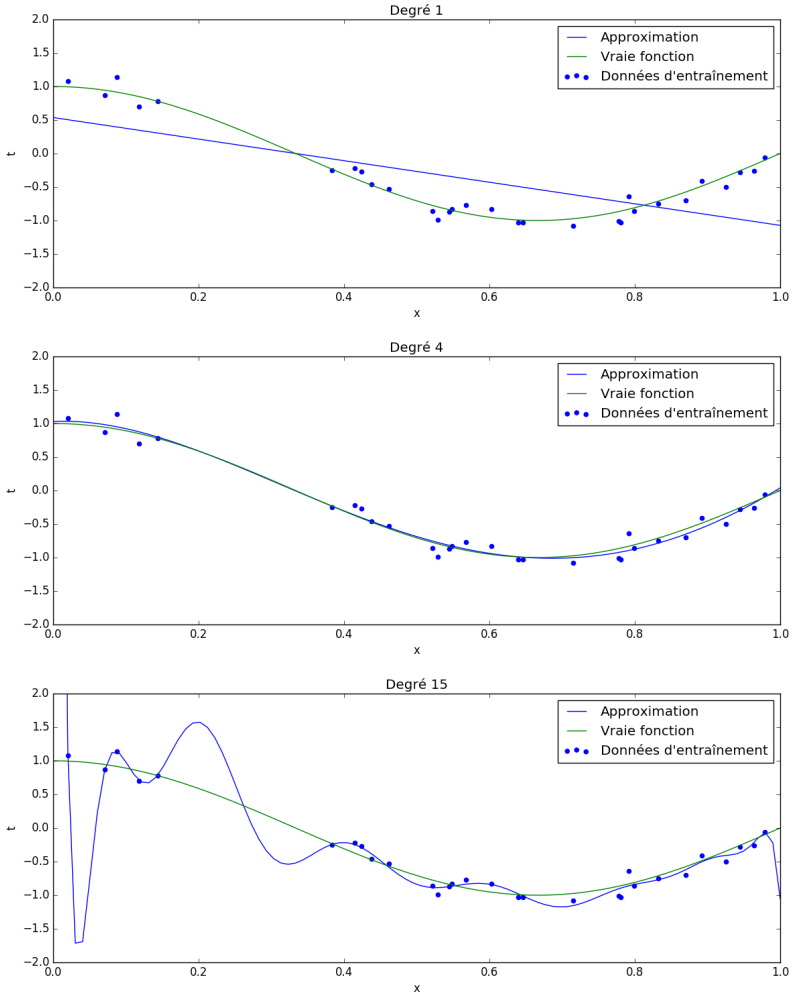


FIGURE 3.3 – On tente d'approximer la fonction $t = cos(\frac{3\pi}{2}x)$ à l'aide d'une régression polynomiale, c'est-à-dire sous la forme $t = \sum_j w_j \cdot f_j(x)$ où les f_j sont les termes polynomiaux, les w_j les coefficients. Un degré maximum est fixé pour le polynôme, puis on calcule les $f_j(x)$ et on réalise une régression linéaire en moindre carrés pour déterminer les w_j (*cf.* section 3.2.1). Le dilemme biais-variance se manifeste ici dans le choix du degré maximum du polynôme, qui entraîne du sous-apprentissage quand celui-ci est faible (en haut) et du sur-apprentissage quand il est élevé (en bas). Un bon compromis entre biais et variance (c'est-à-dire un degré maximum adapté) permet d'obtenir une approximation s'approchant raisonnablement de la fonction que l'on cherche à approximer (au milieu).

Lorsque l'apprentissage passe par une minimisation d'erreur sur les données d'entraînement, un hyper-paramètre peut aussi être le ratio déterminant l'importance relative dans la valeur minimisée de l'erreur et d'un nouveau terme — dit de *régularisation* — visant à pénaliser les solutions jugées moins aptes à se généraliser à de nouvelles données. C'est le cas notamment dans la *régularisation Tikhonov*, qui vient modifier l'objectif de la minimisation des moindres carrés d'une régression linéaire, qui devient ainsi :

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_i (t_i - \beta^T x_i)^2 + \|\Gamma\beta\|^2$$

On parle alors de *régression d'arête* (*ridge regression* en anglais). Il est courant de choisir Γ comme un multiple de la matrice identité, et la minimisation devient donc :

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_i (t_i - \beta^T x_i)^2 + \alpha\|\beta\|^2$$

où α est un hyper-paramètre qu'il reste alors à choisir.

Si l'on choisit de prendre $\alpha = 0$, on se ramène à la méthode des moindres carrés et on cherche $\hat{\beta}$ n'importe où dans $\mathbb{R}^M$, où M est le nombre de composantes de x. À l'inverse, si l'on choisit de prendre une valeur élevée pour α, s'éloigner de $\beta = 0$ entraîne rapidement un coût prohibitif, et l'espace de recherche est alors réduit au voisinage 0. Jouer sur α permet donc ainsi d'influer sur la taille de l'espace de recherche, et par conséquent sur la variabilité du modèle.

Par ailleurs, le bais introduit par le terme de régularisation $\|\beta\|^2$ peut s'interpréter comme une préférence, entre deux solutions fournissant des prédictions de qualité similaire sur le jeu de données d'entraînement, pour celle minimisant la somme des carrés des pentes, jugée plus apte à se généraliser à de nouvelles données. De nombreuses situations de sur-entraînement étant associées à des solutions dont certaines pentes sont très élevées, minimiser les carrés des pentes permet ainsi de leur préférer des solutions dont les pentes sont plus faibles.

Validation croisée

Si la valeur optimale des hyper-paramètres peut être choisie avec soin par un expert ayant une bonne connaissance du problème, il est aussi possible de la déterminer automatiquement via un procédé dit de *validation croisée* (*cross-validation* en anglais). Ce procédé consiste à séparer les données du jeu d'entraînement en deux jeux de données distincts, l'un constituant le nouveau jeu d'entraînement, et l'autre constituant ce qu'on appelle le *jeu de données de validation*. On peut alors fixer la valeur des hyper-paramètres, entraîner le modèle sur le nouveau jeu de données d'entraînement, utiliser ce modèle pour

faire des prédictions sur le jeu de données de validation, puis mesurer l'erreur et ainsi estimer la qualité de l'approximation obtenue.

Comme illustré en Figure 3.2, lorsque la versatilité du modèle augmente on s'attend à observer d'abord une amélioration de la qualité de l'approximation (l'erreur liée au biais diminue plus vite que l'erreur liée à la variance ne monte), puis une baisse de celle-ci (l'erreur liée au biais diminue plus lentement que l'erreur liée à la variance ne monte). On peut donc faire varier la valeur des hyper-paramètres, et utiliser le jeu de données de validation pour chercher empiriquement l'optimum. Notons cependant que la validation croisée reste coûteuse en données (car il faut assez de données pour constituer à la fois les jeux de données d'entraînement et de validation) ainsi qu'en temps de calcul (car il faut entraîner plusieurs fois le modèle, jusqu'à trouver l'optimum).

3.2.4 Maximum a posteriori

Similairement à la régression d'arête, la méthode du *maximum a posteriori* permet d'introduire un biais dans le modèle pour limiter les risques de surentraînement, en s'appuyant cette fois-ci sur une approche probabiliste.

On suppose que l'on a une connaissance a priori des valeurs de β qui peuvent avoir généré les données. Pour l'exprimer, nous supposerons que les valeurs réellement prises par β sont des tirages d'une variable aléatoire B, dont nous connaissons donc la distribution de probabilité $\mathbb{P}(B)$. On supposera donc, avant même de voir les données, que certaines valeurs de β ont plus de chances d'être solution que d'autres et on parlera de *distribution de probabilités a priori*.

Plutôt que de maximiser la vraisemblance, qui avec B s'écrit donc ici $\mathbb{P}(t_0, t_1, ..., t_N, x_0, x_1, ..., x_N \mid B = \beta)$, on s'intéressera à la *distribution de probabilités a posteriori* de B, qui exprime la probabilité qu'une valeur donnée soit solution :

$$\mathbb{P}(B = \beta \mid t_0, t_1, ..., t_N, x_0, x_1, ..., x_N)$$

Celle-ci combine la distribution a priori et l'information provenant des données d'entraînement :

$$\begin{aligned}
&\mathbb{P}(B = \beta \mid t_0, t_1, ..., t_N, x_0, x_1, ..., x_N) \\
&\quad = \frac{\mathbb{P}(B = \beta, t_0, t_1, ..., t_N, x_0, x_1, ..., x_N)}{\int_b \mathbb{P}(B = b, t_0, t_1, ..., t_N, x_0, x_1, ..., x_N)} \\
&\quad = \frac{\mathbb{P}(t_0, t_1, ..., t_N, x_0, x_1, ..., x_N \mid B = \beta)\, \mathbb{P}(B = \beta)}{\int_b \mathbb{P}(t_0, t_1, ..., t_N, x_0, x_1, ..., x_N \mid B = b)\, \mathbb{P}(B = b)}
\end{aligned}$$

La solution choisie par la méthode du maximum a posteriori est alors celle

ayant la plus forte probabilité a posteriori :

$$\begin{aligned}\hat{\beta} &= \underset{\beta}{\operatorname{argmax}}\, \mathbb{P}(B = \beta \,|\, t_0, t_1, ..., t_N, x_0, x_1, ..., x_N) \\ &= \underset{\beta}{\operatorname{argmax}}\, \frac{\mathbb{P}(t_0, t_1, ..., t_N, x_0, x_1, ..., x_N \,|\, B = \beta)\, \mathbb{P}(B = \beta)}{\int_b \mathbb{P}(t_0, t_1, ..., t_N, x_0, x_1, ..., x_N \,|\, B = b)\, \mathbb{P}(B = b)} \\ &= \underset{\beta}{\operatorname{argmax}}\, \mathbb{P}(t_0, t_1, ..., t_N, x_0, x_1, ..., x_N \,|\, B = \beta)\, \mathbb{P}(B = \beta)\end{aligned}$$

On remarque alors que si $\mathbb{P}(B)$ est une distribution de probabilités uniforme (c'est-à-dire lorsqu'elle ne reflète pas réellement un biais), on retrouve la même solution que pour la méthode du maximum de vraisemblance. En pratique, on lui préférera cependant une distribution a priori $\mathbb{P}(B)$ introduisant une régularisation, que l'on exprimera sous une forme facilitant la recherche d'une solution analytique.

3.2.5 Régression d'angle minimal

La *régression d'angle minimal* (*Least-Angle Regression* en anglais, généralement abrégé en LARS) est un algorithme itératif introduit par Efron *et al.* [2004] et qui consiste à supposer initialement que toutes les composantes de β sont nulles (*i.e.* $\forall j,\ \beta^{(j)} = 0$), puis à modifier la valeur des $\beta^{(j)}$ en fonction de la corrélation des $x_i^{(j)}$ aux résidus pour améliorer la qualité de la solution.

Pour cela, on sélectionne d'abord la dimension j_0 telle que les $x_i^{(j_0)}$ ont la plus forte corrélation avec les résidus $r_i = t_i - \beta^T x_i$. On modifie alors $\beta^{(j_0)}$ de façon à diminuer cette corrélation, jusqu'à ce qu'il existe une autre dimension j_1 telle que la corrélation des $x_i^{(j_0)}$ et des $x_i^{(j_1)}$ avec les r_i soient égales.

On continue ensuite en modifiant simultanément $\beta^{(j_0)}$ et $\beta^{(j_1)}$ pour réduire cette corrélation, jusqu'à ce qu'il existe une dimension j_2 telle que la corrélation des $x_i^{(j_0)}$, des $x_i^{(j_1)}$ et des $x_i^{(j_2)}$ avec les r_i soient égales, et ainsi de suite jusqu'à ce que toutes les composantes des x_i aient la même corrélation avec les r_i et qu'on ne puisse plus modifier simultanément les $\beta^{(j)}$ pour diminuer celle-ci.

Cette méthode est relativement peu coûteuse en calculs, et permet de gérer efficacement des situations où le nombre de dimensions est significativement plus grand que le nombre de points de données, en fournissant un moyen de sélectionner les composantes pertinentes des x_i et de laisser $\beta^{(j)} = 0$ pour les autres. Elle est cependant très sensible au bruit sur les t_i, qui peut influer sur les diverses corrélation et mener à la sélection en priorité de composantes peu pertinentes.

3.2.6 Régression à vecteurs de support

La *régression à vecteurs de support* (*Support Vector Regression* en anglais, généralement abrégé en SVR) approche le dilemme biais-variance assez dif-

féremment des autres méthodes que nous avons vu jusqu'ici. Pour éviter de rendre compte du bruit, on considère que les observations des t_i sont bruitées et que leurs valeurs réelles appartiennent à des intervalles $[t_i - \varepsilon,\, t_i + \varepsilon]$ (Drucker *et al.* [1997], Smola et Schölkopf [2004]).

Plutôt que de minimiser une erreur, il s'agit alors de trouver une solution $\hat{\beta}$ permettant d'atteindre tous ces intervalles :

$$\begin{aligned} \forall i,\ \hat{\beta}^T x_i &\in [t_i - \varepsilon,\, t_i + \varepsilon] \\ i.e.\ \forall i,\ |t_i - \hat{\beta}^T x_i| &\leq \varepsilon \end{aligned}$$

Si plusieurs solutions existent, on choisit alors celle qui minimise un terme de régularisation :

$$\hat{\beta} = \underset{\substack{\beta \\ \forall i,\ t_i - \beta^T x_i \leq \varepsilon \\ \forall i,\ \beta^T x_i - t_i \leq \varepsilon}}{\operatorname{argmin}} \|\beta\|^2$$

En pratique, il peut cependant arriver qu'il n'existe pas de solution (à cause de points aberrants par exemple). On préfèrera alors une version relâchée du problème :

$$\hat{\beta} = \underset{\substack{\beta \\ \forall i, \exists \zeta_i, \zeta_i^* \geq 0 \\ t.q.\ \begin{cases} t_i - \beta^T x_i \leq \varepsilon + \zeta_i \\ \beta^T x_i - t_i \leq \varepsilon + \zeta_i^* \end{cases}}}{\operatorname{argmin}} \|\beta\|^2 + \alpha \cdot \sum_{0 \leq i \leq N} (\zeta_i + \zeta_i^*)$$

où α est un hyper-paramètre déterminant l'importance relative du terme de régularisation et de la tolérance aux résidus supérieurs à ε. Cette version relâchée du problème est illustrée en Figure 3.4.

Le terme $\sum_{0 \leq i \leq N} (\zeta_i + \zeta_i^*)$ rappelle fortement une minimisation d'erreur en norme L_1 (somme des valeurs absolues des résidus) dans laquelle toute erreur inférieure à ε serait négligée, ce qui revient donc à considérer que tous les points dans un tube de rayon ε autour de l'approximation correspondent à un coût nul dans la minimisation tandis que ceux en dehors du tube ont un coût proportionnel à leur distance à celui-ci (la pente étant déterminée par l'hyperparamètre α).

3.2.7 Les autres approches

Les approches présentées ci-dessus ne constituent qu'une petite partie des approches de régression linéaire, que nous ne verrons pas toutes car elles sont trop nombreuses. Présentons cependant certains aspects que celles-ci permettent de couvrir.

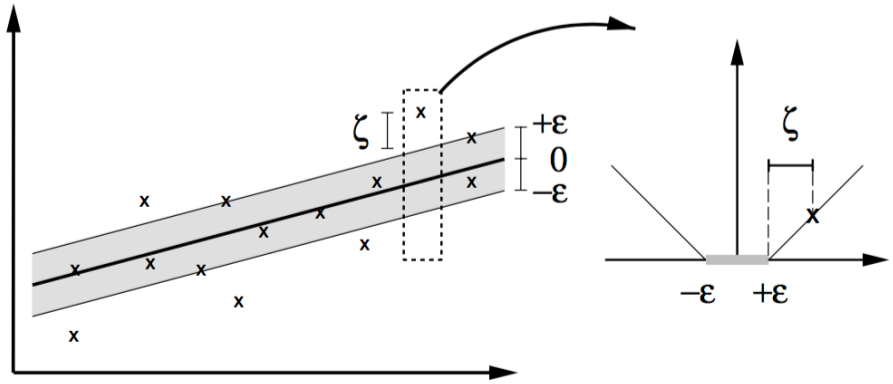


FIGURE 3.4 – Figure tirée de Smola et Schölkopf [2004]. Toute erreur inférieure à ε est négligée, ce qui revient à considérer que tous les points dans un tube de rayon ε autour de l'approximation correspondent à un coût nul dans la minimisation. À l'inverse, les points en dehors du tube ont un coût proportionnel à leur distance à celui-ci (la pente étant déterminée par l'hyperparamètre α).

Hétéroschédasticité

Les approches que nous avons présentées jusqu'ici supposaient une *homoschédasticité* des données, c'est-à-dire que les variables aléatoires qui les décrivent aient toutes la même variance finie. Il existe cependant des approches supposant l'*hétéroschédasticité* des données, c'est-à-dire que chaque point de données soit associé à sa propre variance.

On trouvera notamment parmi ces méthodes celle des moindres carrés pondérés (*Weighted Least Squares* en anglais), dans laquelle le terme minimisé dépendra cette fois de poids w_i, les inverses des variances σ_i^2 sur les valeurs que peuvent prendre les t_i :

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_i w_i \cdot r_i^2 = \underset{\beta}{\operatorname{argmin}} \sum_i \frac{(t_i - \beta^T x_i)^2}{\sigma_i^2}$$

Approches incrémentales

Les approches que nous avons présentées jusqu'ici supposaient également que toutes les données soient disponibles dès le départ. La disponibilité soudaine de nouveaux points de données pourrait donc en principe nécessiter de repartir de zéro pour déterminer la nouvelle solution optimale. Il existe cependant des approches dites *incrémentales*, qui supposent que les données arrivent au fur et à mesure et que chaque ajout de nouveaux points de données doit permettre d'améliorer progressivement la solution.

Ceci pourra notamment être fait à l'aide d'une descente de gradient stochastique (*Stochastic Gradient Descent* en anglais, généralement abrégé en SGD),

qui consiste à améliorer itérativement la solution β en calculant le gradient de la somme des carrés des résidus par rapport à celle-ci, c'est-à-dire le vecteur dont la j-ème composante est $\frac{\partial\left(\sum_i r_i^2\right)}{\partial\beta^{(j)}}$, puis en modifiant β le long de ce gradient dans le sens qui fait diminuer l'erreur jusqu'à atteindre un minimum. À l'ajout de nouveaux points de données, on pourra donc prendre en compte ces derniers dans le calcul du gradient afin d'atteindre un nouveau minimum.

3.2.8 Limites de ces approches

Le problème de régression qui nous intéresse dans cette thèse (*cf.* chapitre 2) consiste à chercher à approximer une fonction qui est $\mathscr{C}^\infty$ localement mais pas globalement, sans connaître à l'avance les frontières entre les régions où la fonction est $\mathscr{C}^\infty$.

Pour qu'une fonction de la forme $\sum_j w_j \cdot f_j(x)$ ne soit pas $\mathscr{C}^\infty$ en un point, il faut qu'au moins une des fonctions f_j ne soit pas $\mathscr{C}^\infty$ en ce point. Lors du choix de la famille de fonctions $(f_j)_{j\in[\![0,M]\!]}$, ne connaissant pas encore les points qui constituent les frontières, il nous est impossible de choisir des f_j qui soient $\mathscr{C}^\infty$ en-dehors de celles-ci sans être globalement $\mathscr{C}^\infty$. Les approches que nous avons vues en section 3.2 ne sont donc pas directement utilisables pour chercher une approximation qui soit $\mathscr{C}^\infty$ localement mais pas globalement (mais nous verrons par la suite qu'elles peuvent être utilisées au sein d'autres approches).

3.3 Régression non paramétrique

Pour nous affranchir de ce problème, nous allons à présent nous intéresser à un autre type d'approches : la *régression non paramétrique*. Plutôt que d'utiliser le jeu d'entraînement pour chercher une approximation sous une forme donnée f_{approx} en déterminant le meilleur vecteur ϑ tel que $t = f_{approx}(\vartheta, x)$, les approches de régression non paramétrique construisent directement l'approximation sur les données d'entraînement.

Il s'agit ainsi d'approches intrinsèquement locales, où l'estimation de t en un point dépend principalement des valeurs prises par les points du jeu de données d'entraînement les plus proches. Cette localité rend ces méthodes particulièrement adaptées pour l'approximation de fonctions qui ne sont pas globalement $\mathscr{C}^\infty$, voire qui ne sont pas continues. Voyons-en quelques unes.

3.3.1 Algorithme des k plus proches voisins

La régression via l'*algorithme des k plus proches voisins* (*k-nearest neighbour* en anglais, généralement abrégé en k-NN) est la forme la plus simple de

régression non-paramétrique. Elle consiste, étant donnés un entier k et une distance d sur l'espace des x_i, à estimer la valeur de t en chaque x en déterminant les k points du jeu de données d'entraînement pour lesquels $d(x_i, x)$ prend les plus faibles valeurs, puis à moyenner la valeur des t_i correspondants (Altman [1992]).

Cet algorithme est illustré dans le cas de l'approximation de la fonction $x \mapsto |sin(x)|$ en Figure 3.5. On peut à cette occasion remarquer que l'approximation fournie par l'algorithme des plus proches voisins est localement $\mathscr{C}^\infty$ (là où elle est constante) mais pas globalement. On peut également remarquer que k est un hyper-paramètre qui joue sur le compromis entre biais et variance : choisir $k = 1$ mène à une situation de sur-entraînement (ce que l'on peut voir dans la partie supérieure de la Figure 3.5), alors qu'une valeur de k très élevée mènerait à l'inverse à une situation de sous-entraînement où l'approximation est une fonction constante qui associe chaque x à la moyenne de tous les t_i.

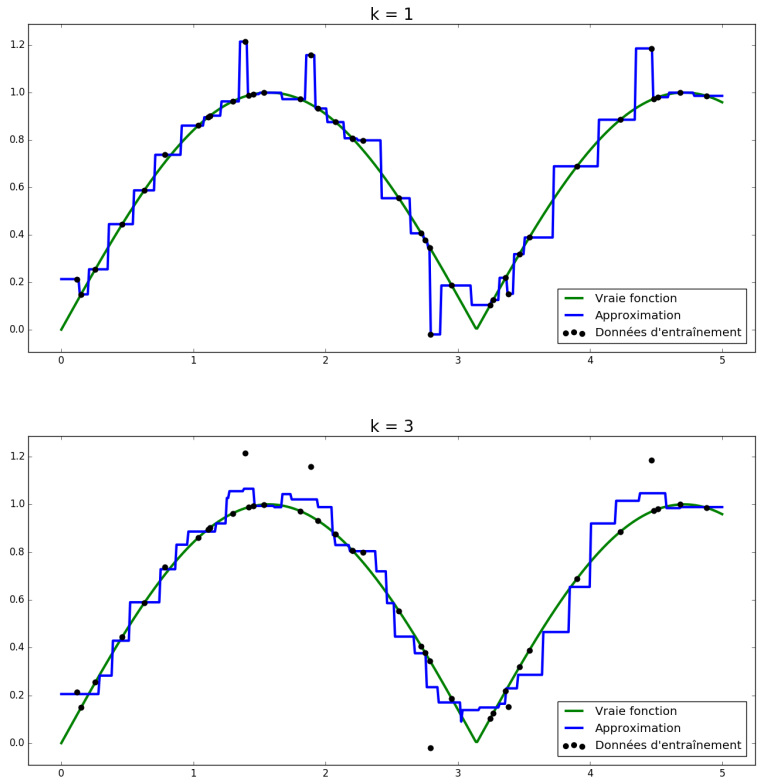


FIGURE 3.5 – Approximation de la fonction $x \mapsto |sin(x)|$ sur $[0, 5]$ à l'aide de l'algorithme des k plus proches voisins, pour $k = 1$ (en haut) et $k = 3$ (en bas).

3.3.2 Régression par la méthode du noyau

La *régression par la méthode du noyau* (*Kernel Regression* en anglais) est une généralisation de l'algorithme des k plus proches voisins. Dans cette nouvelle approche, plutôt que de calculer une moyenne à partir de quelques points contribuant à part égale, on s'intéresse à une moyenne pondérée dans laquelle tous les points contribuent à l'estimation en fonction de leur proximité (Altman [1992]).

Pour estimer t en x, chaque point (x_i, t_i) du jeu d'entraînement est donc associé à un poids en fonction de la proximité entre x et le x_i, qui détermine l'importance de t_i dans la moyenne pondérée :

$$t = \frac{\sum_i k(x, x_i) \cdot t_i}{\sum_i k(x, x_i)}$$

où k est une fonction dite *fonction noyau*, qui mesure la similarité entre ses arguments et généralement choisie sous forme gaussienne :

$$k(a, b) = \exp\left(-\frac{1}{2}(a-b)^T \Lambda^{-1}(a-b)\right)$$

où Λ est une matrice diagonale attribuant un facteur d'échelle à chaque dimension, déterminant pour celle-ci à quelle vitesse le poids doit diminuer avec la distance entre les points.

Cet algorithme est illustré dans le cas de l'approximation de la fonction $x \mapsto |sin(x)|$ en Figure 3.6. On peut remarquer que les valeurs sur la diagonale de Λ sont des hyper-paramètres jouant sur la localité de la régression, et que similairement à l'algorithme du plus proche voisin elles permettent de jouer sur le compromis entre biais et variance.

3.3.3 LOESS et régression localement pondérée

LOESS (pour *LOcal regrESSion* en anglais) est un algorithme proche de l'algorithme des k plus proches voisins mais qui — plutôt que de moyenner les valeurs des t_i de ces voisins — effectue une régression linéaire en moindre carrés sur ceux-ci pour en déduire une approximation locale et estimer la valeur de t (Cleveland et Devlin [1988]).

De même que l'algorithme des k plus proches voisins pouvait être généralisé pour donner la régression par la méthode du noyau, la méthode LOESS peut être généralisée pour donner un algorithme appelé *régression localement pondérée* (*Locally Weighted Regression* en anglais). La régression localement pondérée s'appuie à nouveau sur une fonction noyau pour attribuer un poids à chaque point du jeu d'entraînement (Schaal et Atkeson [1994]), celle-ci étant là encore généralement choisie sous forme gaussienne :

$$k(a, b) = \exp\left(-\frac{1}{2}(a-b)^T \Lambda^{-1}(a-b)\right)$$

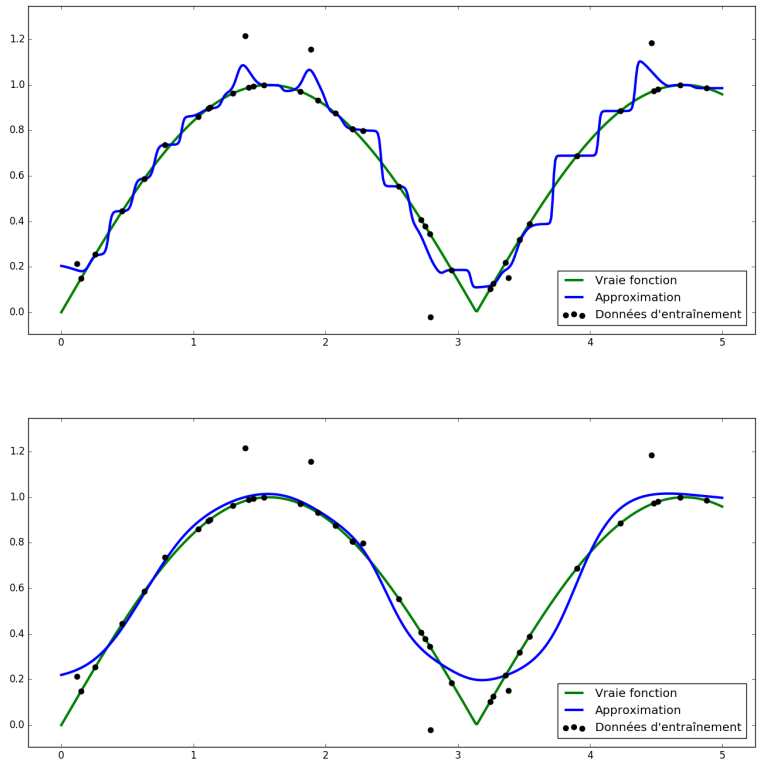


FIGURE 3.6 – Approximation de la fonction $x \mapsto |sin(x)|$ sur $[0, 5]$ à l'aide d'une régression par la méthode du noyau, pour $\Lambda = \left(\frac{1}{15}\right)$ (en haut) et $\Lambda = \left(\frac{1}{3}\right)$ (en bas).

où Λ est une matrice diagonale attribuant un facteur d'échelle à chaque dimension, déterminant pour celle-ci à quelle vitesse le poids doit diminuer avec la distance entre les points.

Pour estimer la valeur de t en x, il s'agit alors de réaliser une régression linéaire en moindre carrés pondérés dans laquelle chaque point (x_i, t_i) du jeu d'entraînement se voit attribuer le poids $k(x, x_i)$. Il s'agit donc d'une régression de la forme :

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_i k(x, x_i) \cdot r_i^2 = \underset{\beta}{\operatorname{argmin}} \sum_i k(x, x_i) \cdot (t_i - \beta^T x_i)^2$$

Cette régression fournit alors un approximation locale que l'on peut utiliser pour estimer t en x.

Cet algorithme est illustré dans le cas de l'approximation de la fonction $x \mapsto |sin(x)|$ en Figure 3.7. On peut remarquer que là-encore les valeurs sur la

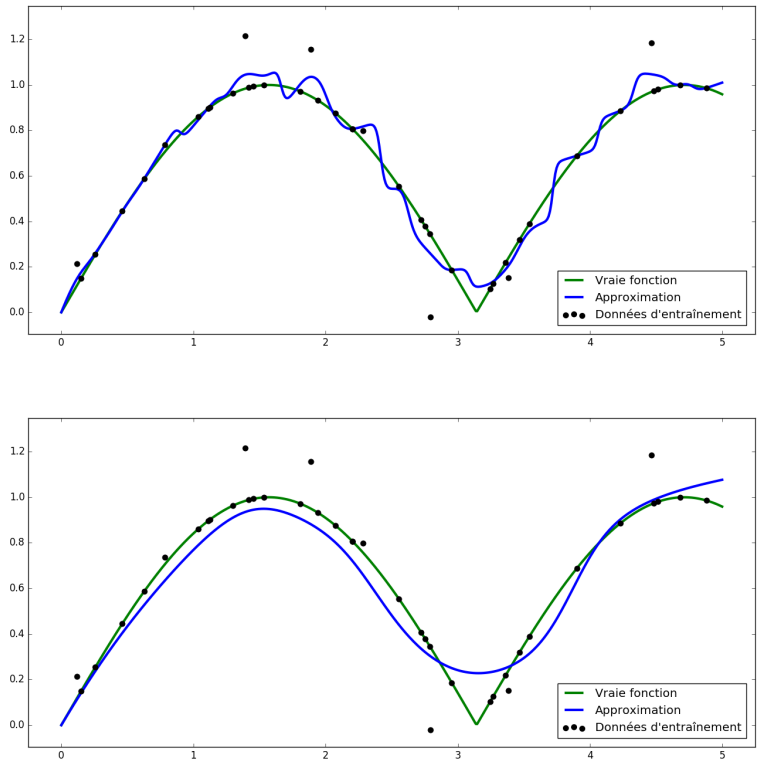


FIGURE 3.7 – Approximation de la fonction $x \mapsto |sin(x)|$ sur $[0, 5]$ à l'aide d'une régression localement pondérée, pour $\Lambda = \left(\frac{1}{10}\right)$ (en haut) et $\Lambda = \left(\frac{1}{2}\right)$ (en bas).

diagonale de Λ (hyper-paramètres de la fonction noyau) permettent de jouer sur la localité de la régression, et ainsi sur le compromis entre biais et variance.

3.3.4 Processus gaussiens

Dans la *régression vers un processus gaussien* (*Gaussian Process Regression* en anglais), aussi appelée *krigeage* (*kriging* en anglais), l'estimation de la valeur de t au point x repose sur l'hypothèse suivante :

$$\begin{bmatrix} v_t \\ t \end{bmatrix} \sim \mathcal{N}\left(0, K + \sigma^2 I_{N+1}\right)$$

où v_t est le vecteur des t_i, σ^2 est la variance du bruit (supposé gaussien), et K est une matrice de covariance décrivant le couplage entre les différentes observations (Lázaro-Gredilla *et al.* [2012]). K et σ jouent alors le rôle de paramètres du modèle : si leurs valeurs sont connues, on peut calculer analy-

tiquement l'espérance de la distribution et ainsi obtenir notre estimation de t en x.

On fait l'hypothèse supplémentaire que la matrice K peut être exprimée à l'aide d'une fonction k connue, appelée *fonction de covariance*, de la façon suivante :

$$K_{i,j} = \begin{cases} k(x,x) & \text{si } i = j = N \\ k(x_i,x) & \text{si } 0 \leq i < N \text{ et } j = N \\ k(x,x_j) & \text{si } i = N \text{ et } 0 \leq j < N \\ k(x_i,x_j) & \text{si } 0 \leq i < N \text{ et } 0 \leq j < N \end{cases}$$

La fonction k est alors généralement choisie sous forme gaussienne :

$$k(a,b) = \sigma_0^2 \exp\left(-\frac{1}{2}(a-b)^T \Lambda^{-1}(a-b)\right)$$

où σ_0^2 est la variance sur la valeur d'un point donné et Λ est une matrice diagonale attribuant un facteur d'échelle à chaque dimension, déterminant pour celle-ci à quelle vitesse la corrélation diminue avec la distance entre les points.

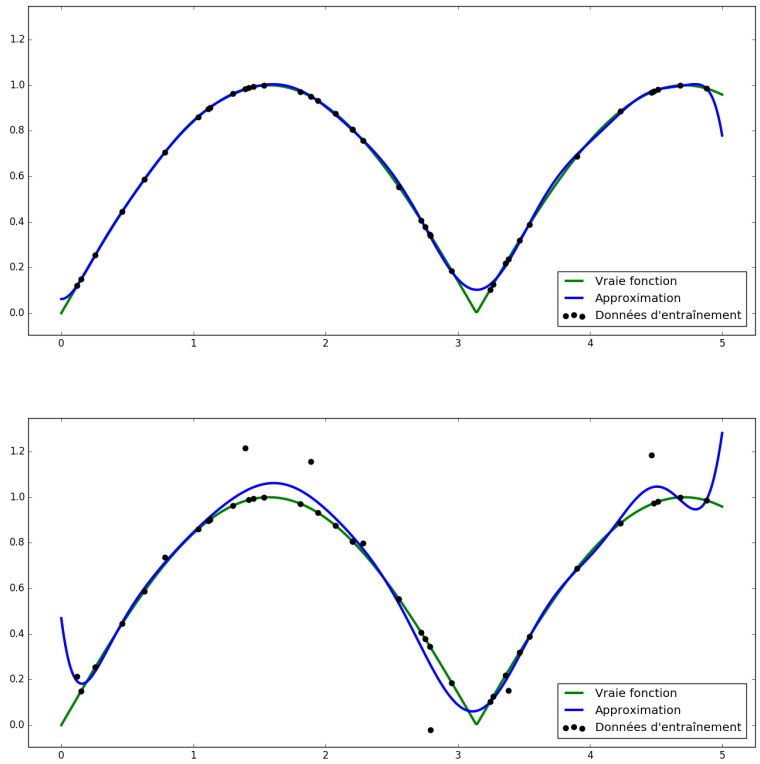


FIGURE 3.8 – Approximation de la fonction $x \mapsto |sin(x)|$ sur $[0,5]$ à l'aide d'une régression vers un processus gaussien, pour des données non bruitées (en haut) et bruitées (en bas).

Avec cette hypothèse, les paramètres de notre modèle sont désormais σ, σ_0 et les termes diagonaux de Λ. Il reste alors à déterminer les valeurs pour ces paramètres pour pouvoir calculer l'espérance et la variance de t en x. On peut pour cela suivre une logique de maximum de vraisemblance ou de maximum a posteriori, par exemple à l'aide d'une descente de gradients. Cet algorithme est illustré dans le cas de l'approximation de la fonction $x \mapsto |sin(x)|$ en Figure 3.8.

3.3.5 Limite de ces approches

Nous venons donc de voir quelques approches de régression non paramétrique, intrinsèquement locales, et que celles-ci fournissent d'assez bonnes approximations malgré le fait que la fonction que l'on cherche à approximer n'est pas $\mathscr{C}^\infty$ sur l'ensemble de son domaine de définition.

Si ces méthodes pourraient par conséquent fournir une piste pour approximer un modèle direct global, force est de constater qu'elles ne permettent pas d'en partitionner le domaine de définition pour en extraire des modèles directs locaux tels que décrits en chapitre 2, et dont on pourrait tirer une interprétation sémantique.

De plus, si la localité permet d'aboutir à une approximation qui n'est pas globalement $\mathscr{C}^\infty$, elle force aussi l'estimation de t en un point x à être déterminée par le voisinage de x, en n'utilisant pas ou peu les autres points d'entraînement. On peut s'attendre à ce que des méthodes qui partitionneraient le domaine de définition et associeraient chaque sous-domaine à une approximation $\mathscr{C}^\infty$ puissent à l'inverse tirer partie de cette régularité pour permettre aux approximations locales d'utiliser tous les points de données d'entraînement appartenant au sous-domaine correspondant. De telles méthodes pourraient donc être plus efficaces en termes de données d'entraînement (*i.e.* mener à des approximations de meilleure qualité à l'aide de moins de points de données).

3.3.6 Cas des réseaux de neurones

Un autre type d'approches qui souffre des mêmes limites pour notre problème couvre les approches s'appuyant sur des *réseaux de neurones artificiels* (*artificial neural networks* en anglais).

Un *neurone artificiel* est une fonction de la forme $x \mapsto \phi(\sum_i w_i x_i)$, où les x_i sont les diverses coordonnées de x, les w_i des poids attribués à chacune de ces coordonnées, et ϕ une fonction $\mathbb{R} \to \mathbb{R}$ (on parle de *fonction d'activation* du neurone).

Des fonctions plus complexes peuvent être construites à partir de nombreux neurones suivant une logique de composition, les valeurs d'entrée des uns étant un vecteur des valeurs de sortie de plusieurs autres. Ce qu'on appelle un *réseau de neurones* est alors le résultat d'une telle composition (Goodfellow *et al.*

[2016]), et on le représente généralement par un graphe orienté dont les nœuds sont les neurones et dont les arcs reflètent la structure des compositions.

L'utilisation de compositions permet aux réseaux de neurones d'être des modèles très expressifs, et en particulier ceux mettant en jeu de nombreuses compositions successives et que l'on appelle *réseaux profonds* (*deep networks* en anglais), et ils sont par conséquent très utilisés dans le domaine de l'apprentissage automatique et notamment en régression. Le nombre de neurones, la structure des compositions, et les fonctions d'activations des neurones sont alors généralement fixées en amont de l'apprentissage, tandis que les poids sont déterminés à l'aide du jeu de données d'entraînement. On utilise pour cela une descente de gradient stochastique (*cf.* 3.2.7), c'est-à-dire que l'on améliore itérativement les poids depuis des valeurs initiales arbitraires (souvent prises aléatoirement) le long du gradient des résidus.

L'expressivité des réseaux de neurones leur permet de fournir d'assez bonnes approximations même lorsque la fonction que l'on cherche à approximer n'est pas $\mathscr{C}^\infty$ sur l'ensemble de son domaine de définition, et pourraient donc également fournir une piste pour approximer un modèle direct global.

Malheureusement, comme nous le disions plus haut, les approches s'appuyant sur les réseaux de neurones souffrent de limitations similaires à celles des approches non-paramétriques : elles sont réputées pour être très gourmandes en données d'entraînement, et elles ne permettent pas de partitionner le domaine de définition pour en extraire des modèles directs locaux tels que décrits en chapitre 2, et dont on pourrait tirer une interprétation sémantique.

3.4 Mélange d'experts

Une autre façon d'aborder la localité est celles des approches dites de *mélange d'experts* (*mixture of experts* en anglais). Ces approches reposent sur l'entraînement de différents *experts*, — des approximations locales centrées sur une position de l'espace — couplés à des fonctions déterminant l'influence de chaque expert en chaque point de l'espace (Jacobs *et al.* [1991]). Lorsque l'on estime la valeur de t en x, chaque expert fournit donc sa propre estimation, puis ces estimations sont ensuite combinées en fonction de leurs influences relatives en x.

De telles approches peuvent être appliquées à l'apprentissage de modèles directs (Wolpert et Kawato [1998]) et constituent une piste prometteuse pour résoudre notre problème, les approximations locales $\mathscr{C}^\infty$ que nous recherchons pouvant en principe être exprimées sous la forme d'un ensemble d'experts bien choisis et correctement localisés dans l'espace. Passons en revue quelques unes de ces méthodes.

3.4.1 Régressions locales pondérées et régression pondérée par des champs récepteurs

La *méthode des régressions locales pondérées* (à ne pas confondre avec la régression localement pondérée vue en 3.3.3, avec laquelle elle partage souvent le nom de *Locally Weighted Regression* en anglais) est une forme simple de mélange d'experts, dans laquelle les experts sont obtenus en choisissant arbitrairement un nombre quelconque M de points $(c_j)_{j \in [\![0,M[\![}$ du domaine de définition, puis en réalisant une régression en moindre carrés pondérés autour de chacun de ces points.

Dans la régression centrée sur chacun des c_j, les poids des points d'entraînement sont déterminés par leur proximité avec c_j similairement à ce que nous avons vu plus haut dans le cas de la régression localement pondérée (*cf.* 3.3.3), c'est-à-dire à l'aide d'une fonction k généralement choisie sous forme gaussienne. Les solutions $(\hat{\beta}_j)_{j \in [\![0,M[\![}$ de ces régressions définissent alors chacune une approximation, un expert. L'estimation de t au point x s'obtient ensuite à l'aide d'une moyenne pondérée des estimations fournies par ces experts, les poids étant là-encore fournis par k (Stulp et Sigaud [2015]) :

$$t = \frac{\sum_j k(x, c_j) \cdot \hat{\beta}_j^T x}{\sum_j k(x, c_j)}$$

La *régression pondérée par des champs récepteurs* (*Receptive Field Weighted Regression* en anglais, généralement abrégé en RFWR) est un algorithme incrémental (Schaal et Atkeson [1997]) qui peut être vu comme une version incrémentale la méthode des régressions locales pondérées. Cette incrémentalité se retrouve à trois niveaux : au niveau des régressions locales, qui sont désormais réalisées à l'aide d'un algorithme de moindre carrés incrémental (algorithme des moindres carrés récursifs), au niveau des poids des points d'entraînement dans ces régressions, qui sont attribués à l'aide de gaussiennes centrées sur les c_j et dont les matrices de covariances peuvent être adaptées, et au niveau des centres des régressions eux-mêmes (appelés *champs récepteurs*), les positions des c_j pouvant être modifiées et de nouveaux centres ajoutés.

3.4.2 Réduction de la dimensionnalité et régression projective localement pondérée

L'*analyse en composantes principales* (*Pincipal Component Analysis* en anglais, généralement abrégé en PCA) est une méthode qui consiste à projeter les valeurs de x vers un espace de plus faible dimension (Wold *et al.* [1987]). On dit alors qu'il s'agit d'une méthode de *réduction de la dimensionnalité*. Pour cela, on calcule la matrice de covariance des valeurs de x, puis on en cherche les valeurs et vecteurs propres. Chaque valeur propre ainsi trouvée correspond

à la variance des données d'entraînement le long de son vecteur propre associé. On peut alors projeter les valeurs de x sur la base des vecteurs propres dont les valeurs propres sont non nulles (ce qui revient à éliminer les redondances), voire sur celle des vecteurs propres dont les valeurs propres sont les plus grandes (qui sont souvent jugées plus pertinents).

La méthode des *moindre carrés partiels*, aussi appelée *projection sur la structure latente* (*Partial Least Squares* ou *Projection to Latent Structure* en anglais, généralement abrégés en PLS), est une autre méthode de réduction de la dimensionnalité qui consiste, lors d'une régression linéaire, à projeter les valeurs de x et t sur de nouveaux espaces avant de réaliser la régression en moindre carrés (Geladi et Kowalski [1986]). Contrairement à PCA, qui projette les valeurs de x sur les directions de l'espace le long desquelles elles ont les plus fortes variances, c'est aux directions de l'espace le long desquelles x et t sont les plus corrélées que PLS s'intéresse.

PLS peut être combinée avec d'autres formes de régression. On pensera notamment à la méthode des *moindre carrés partiels localement pondérés* (*Locally Weighted Partial Least Squares* en anglais, généralement abrégé en LW-PLS), une variation de la régression localement pondérée dans laquelle on utilise PLS avant chaque régression locale (Kim *et al.* [2011]).

La *régression projective localement pondérée* (*Locally Weighted Projection Regression* en anglais, généralement abrégé en LWPR) est quant à elle une forme de mélange d'experts combinant RFWR et PLS (Stulp et Sigaud [2015]). Plus précisément, il s'agit d'une variation de RFWR dans laquelle on utilise la méthode des *moindre carrés partiels itératifs non-linéaires* (*Nonlinear Iterative Partial Least Squares* en anglais, généralement abrégé en NIPALS) — une forme itérative de PLS (Geladi et Kowalski [1986]) — pour que les régressions locales soient effectuées après avoir projeté les valeurs de x et t (Vijayakumar et Schaal [2000]).

3.4.3 XCSF

XCSF est une forme de mélange d'experts assez similaire à RFWR, en cela qu'elle détermine automatiquement des centres autour desquels sont réalisés des régressions locales pour entraîner des experts (Sigaud *et al.* [2011]). Elle comporte cependant quelques différences avec RFWR, la plus notable étant l'utilisation d'un algorithme génétique pour gérer la population des centres (et de leurs matrices de covariances associées).

Une autre différence notable est la façon de combiner les estimations fournies par les différents experts. Si l'estimation de t en x résulte ici aussi de la combinaison linéaire des estimations fournies par les divers experts, les gaussiennes qui définissent les influences relatives de ces experts ne servent plus directement de poids dans cette combinaison linéaire. Au lieu de ça, elles servent de fonctions d'activation, les modèles dont la fonction d'activation dépasse un

certain seuil en x étant ainsi considérés actifs et ceux dont elle reste en-dessous de ce seuil étant considérés inactifs. Pour estimer t en x, on attribue alors un poids 0 à tous les modèles inactifs en x, tandis que le poids des autres modèles correspond à leur *fitness*, c'est-à-dire à leur score dans l'algorithme génétique (le score d'un modèle est mis à jour à chaque fois que celui-ci est actif durant l'entraînement, augmentant lorsqu'il fournit une estimation de bonne qualité et diminuant dans le cas contraire).

3.4.4 Utilisation de la distribution des données

Certaines approches s'appuyant sur le mélange d'experts cherchent à tirer parti du fait que les valeurs prises par x et t n'occupent pas toujours uniformément les espaces dans lesquelles celles-ci sont représentées. On citera par exemple Oudeyer *et al.* [2007], dont l'algorithme itératif IAC entraîne un modèle linéaire par région de l'espace des x (possiblement une seule initialement), et la subdivise dès que le nombre de points de données d'entraînement qu'elle contient dépasse un certain seuil — ce qui permet d'obtenir une approximation plus fine là où les données sont plus denses.

On pourra citer également da Silva *et al.* [2012], qui suppose que les valeurs prises par t appartiennent à des surfaces (ou plus précisément des variétés) de plus petites dimensions que l'espace dans lequel elles sont représentées, et utilise l'algorithme ISOMAP (Tenenbaum *et al.* [2000]) pour essayer de découvrir ces surfaces et ainsi découper l'espace des t en régions. Il suppose ensuite que chacune de ces régions est l'image d'une région de l'espace des x par la fonction à approximer, ce qui lui permet d'une part d'entraîner un algorithme de classification pour segmenter l'espace des x en ces régions, et d'autre part d'entraîner un expert sur chacune d'entre elles.

3.4.5 Limites des méthodes existantes

Le problème de régression qui nous intéresse dans cette thèse consiste à chercher les sous-espaces les plus grands sur lesquels la fonction à approximer est $\mathscr{C}^\infty$ et à trouver des approximations locales sur ces sous-espaces (*cf.* chapitre 2). Toute solution à ce problème étant composée d'un ensemble d'approximations locales et d'un découpage de l'espace en régions où une seule de ces approximations est active, elle prend *de facto* la forme d'un mélange d'experts.

Cependant, tout mélange d'experts $\mathscr{C}^\infty$ rendant compte des données d'entraînement n'est pas nécessairement la solution que nous cherchons. Il est par exemple possible — quelle que soit la fonction à approximer — de sur-entraîner un modèle $\mathscr{C}^\infty$ très versatile pour rendre compte de l'ensemble des points d'entraînement et ainsi aboutir à un unique expert. À l'inverse, une régression par la méthode des k plus proches voisins (*cf.* section 3.3.1) avec $k = 1$ aboutit à

un expert $\mathscr{C}^\infty$ pour chaque point d'entraînement. Il s'agit donc non seulement de trouver un mélange d'experts $\mathscr{C}^\infty$ rendant compte des données d'entraînement, mais aussi que ce mélange d'experts correspondent à un bon compromis entre la taille des régions et la versatilité des approximations locales pour que les régions trouvées correspondent bien aux sous-espaces les plus grands sur lesquels la fonction à approximer est $\mathscr{C}^\infty$.

Les algorithmes de mélange d'experts que nous venons de voir — et pour autant que nous puissions juger, ceux présents dans la littérature également — semblent assez peu adaptés à la recherche de ce meilleur compromis. La plupart d'entre eux s'appuient sur des approximations locales dont la versatilité est fixée et se focalisent sur la détermination du nombre de ces experts et leur assignation à des régions pour rendre compte au mieux des données d'entraînement, ce qui peut aisément mener à des modèles trop versatiles qui couvrent une région trop grande, ou à plusieurs modèles trop peu versatiles pour couvrir ce qui ne devrait être qu'une seule région. L'alternative n'est guère mieux : déterminer des régions avant d'entraîner les approximations locales s'appuie soit sur la distribution des valeurs de x — qui est a priori indépendante de la fonction que l'on cherche à approximer — soit sur celle des valeurs de t mais (comme nous avons pu le voir dans la section précédente) en s'appuyant sur des hypothèses très fortes qui ne peuvent pas être faites dans le cas général.

3.5 Conclusion

Dans ce chapitre nous avons passé en revue différents algorithmes de régression pour en chercher un adapté au problème qui nous intéresse : déterminer les sous-espaces les plus grands sur lesquels la fonction à approximer est $\mathscr{C}^\infty$ et trouver des approximations locales sur ces sous-espaces.

Nous nous sommes d'abord intéressés aux algorithmes de régression linéaire, qui bien que non directement applicables à notre problème nous ont permis d'aborder certains enjeux des problèmes de régressions. Ces algorithmes peuvent d'autre part être étendus à des cas non-linéaires, et sont à la base de la plupart des algorithmes plus avancés. Nous avons cependant vu que même étendus à des cas non-linéaires, ils restent insuffisant pour traiter le problème qui nous intéresse car ils ne sont pas assez expressifs pour approximer de façon satisfaisante des fonctions qui sont localement $\mathscr{C}^\infty$ mais pas globalement.

Nous avons donc ensuite considéré les algorithmes de régression non paramétrique, particulièrement adaptés à l'approximation de fonctions qui ne sont pas globalement $\mathscr{C}^\infty$, mais ces méthodes se sont avérées trop locales pour tirer parti efficacement des données, et ne permettent de plus pas réellement de partitionner le domaine de définition en sous-espaces distincts. Nous nous sommes par conséquent tournés vers une autre façon d'aborder la localité : les algorithmes de mélange d'experts. Si cette approche semble parfaite pour

notre problème, nous avons toutefois constaté que les algorithmes existants s'intéressent peu à la question du compromis entre le nombre des experts et la versatilités de ceux-ci, qui paraît pourtant cruciale à notre cas d'application. Tout ceci est résumé dans le tableau présenté en Table 3.1.

Dans le prochain chapitre, nous nous intéresserons plus en détail à ce compromis entre nombre et versatilité des experts et verrons comment ceci peut mener à un nouvel algorithme de mélange d'experts.

	Approximation de fonction non globalement $\mathscr{C}^\infty$	Segmentation de l'espace en sous-espaces distincts (où l'approximation locale est $\mathscr{C}^\infty$)	Segmentation découverte automatiquement	Segmentation correspondant aux ruptures de régularité de la fonction à approximer
Régression linéaire	✗	✗		
k plus proches voisins	✓	✓	✓	✗
Méthode du noyau	✓	✗		
LOESS	✓	✗		
Régression localement pondérée	✓	✗		
Régression vers un processus gaussiens	✓	✗		
Réseaux de neurones artificiels	✓	✗		
Méthode des régressions locales pondérées	✓	✓	✗	✗
Régression pondérée par des champs récepteurs	✓	✓	✓	✗
Moindre carrés partiels localement pondérés	✓	✓	✗	✗
Régression projective localement pondérée	✓	✓	✓	✗
XCSF	✓	✓	✓	✗
IAC	✓	✓	partiellement (subdivision de l'espace)	✗
ISOMAP + classifieur	✓	✓	✓	✓ (sous des hypothèses fortes)

TABLE 3.1 – Comparaison des différentes approches présentées

Chapitre 4

L'algorithme COCOTTE

Précédemment, nous avons ramené notre problème d'apprentissage de compétences paramétrées à un problème de régression vers une fonction localement $\mathscr{C}^\infty$ (chapitre 2), l'enjeu étant alors de déterminer les plus grands sous-espaces du domaine de définition sur lesquels elle est $\mathscr{C}^\infty$ et de déterminer des approximations locales à ces espaces.

Nous avons ensuite passé en revue divers algorithmes de régression (chapitre 3) et établi qu'une approche de type mélange d'experts serait appropriée, mais que les algorithmes présents dans la littérature s'intéressent peu à une question qui semble ici un enjeu essentiel : le compromis entre le nombre d'experts et la versatilité de ceux-ci.

Dans ce chapitre, nous allons nous intéresser plus en détail à cette question, puis nous introduirons COCOTTE, l'algorithme de mélange d'experts que nous avons développé pour y répondre.

4.1 Le compromis entre le nombre et la versatilité des experts

4.1.1 Rappel du problème

Reprenons l'exemple vu en section 2.3.5 de l'approximation d'une fonction échelon. La fonction échelon comporte une unique discontinuité, de part et d'autre de laquelle elle est localement $\mathscr{C}^\infty$ et constante. Nous souhaitons donc d'une part identifier cette discontinuité pour partitionner l'espace le long de celle-ci, et d'autre part trouver une approximation constante locale sur chacun des sous-espaces ainsi créés. Supposons maintenant que nous n'avons pas accès à la fonction échelon mais seulement à un jeu de données d'entraînement : comment alors caractériser ce mélange d'experts que nous venons de décrire ? Plus généralement, comment caractériser le mélange d'experts qui partitionne l'espace en ses plus grands sous-espaces sur lesquels la fonction à approximer

est localement $\mathscr{C}^\infty$, en s'appuyant uniquement sur les points de données d'entraînement pour la caractérisation ?

Comme nous avons vu au chapitre précédent (*cf.* section 3.4.5), trouver des régions sur lesquelles des experts $\mathscr{C}^\infty$ peuvent rendre compte localement des points de données ne suffit pas (auquel cas de nombreux mélanges d'experts seraient solution, y compris celui fourni par l'algorithme des k plus proches voisins avec $k = 1$, voir Figure 4.1). Il ne s'agit pas non plus simplement de chercher les plus grandes régions sur lesquelles des approximations locales $\mathscr{C}^\infty$ rendant compte des points de données peuvent être trouvées (auquel cas la série de Fourier de notre fonction échelon à un ordre assez élevé serait solution, voir Figure 4.1 également).

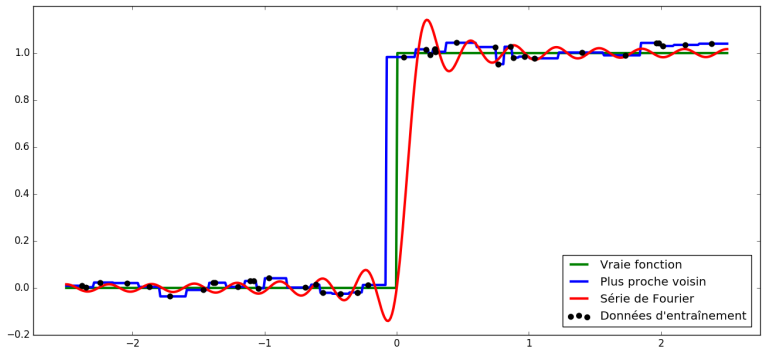


FIGURE 4.1 – Approximation de la fonction échelon d'Heaviside (en vert) par sa série de Fourier à l'ordre 30 (en rouge), et par la méthode des 1-plus proches voisins (en bleu).

Nous savons donc que divers mélanges d'experts $\mathscr{C}^\infty$ permettent de rendre compte des points de données d'entraînement, et que parmi ceux-ci il est possible de tomber dans deux écueils : avoir un nombre trop important d'experts peu versatiles (comme dans le cas du plus proche voisin) ou avoir un nombre pas assez important d'experts trop versatiles (comme dans le cas de la série de Fourier). Éviter le premier écueil requiert de donner aux experts la versatilité dont ils ont besoin pour pouvoir rendre compte des données au sein des régions qui ne doivent pas être scindées, mais éviter le second requiert de donner suffisamment peu de versatilité à ces experts pour qu'ils ne puissent pas fournir une approximation $\mathscr{C}^\infty$ là où la fonction à approximer ne l'est pas.

Nous sommes donc face à une situation assez similaire au dilemme biais-variance (*cf.* section 3.2.3), et la question que nous devrons nous poser pour caractériser le mélange d'experts que nous recherchons est donc la suivante : comment choisir la versatilité des divers experts pour éviter de tomber dans ces écueils ?

4.1.2 Et si on utilisait la validation croisée ?

Dans le cas du dilemme biais-variance, nous avons vu qu'une approche possible était de lier la versatilité à des hyper-paramètres, puis de recourir à la validation croisée (*cf.* section 3.2.3) : scinder les données d'entraînement en deux jeux de données distincts, utiliser le premier pour entraîner un modèle successivement avec diverses valeurs des hyper-paramètres, et le second pour évaluer la qualité des approximations ainsi obtenues. On peut ainsi rechercher empiriquement le compromis biais-variance qui mène aux meilleures performances hors de son jeu d'entraînement.

Dans le cas des mélanges d'experts, les hyper-paramètres que l'on ferait varier seraient d'une part ceux qui déterminent la versatilité des experts (degré maximum d'un polynôme, nombre d'harmoniques d'une série de Fourier, *etc*), et d'autre part le nombre d'experts. Pourrait-on alors simplement explorer l'espace de ces hyper-paramètres jusqu'à trouver ceux qui minimisent l'erreur sur le jeu de validation ?

Tout d'abord, nous devons nous demander si la versatilité des divers experts serait déterminée individuellement pour chacun d'entre eux par un jeu d'hyper-paramètres dédié, ou collectivement par un jeu d'hyper-paramètres communs. Nous savons que les experts que nous souhaitons trouver doivent correspondre à des restrictions de la fonction à approximer aux sous-espaces sur lesquels celle-ci est $\mathscr{C}^\infty$. Si nous ne faisons pas d'hypothèses supplémentaires sur cette fonction, il n'y a pas de raison a priori de penser que ces restrictions ont la même forme. Certaines peuvent par exemple être des fonctions constantes tandis que d'autres sont des fonctions d'une grande complexité. Fixer la versatilité des experts via un même jeu d'hyper-paramètres entraînerait donc un risque d'avoir simultanément du sur-entraînement pour certains experts et du sous-entraînement pour d'autres, voire d'avoir simultanément un expert qui est suffisamment versatile pour approximer la fonction là où elle n'est pas $\mathscr{C}^\infty$ et plusieurs experts au lieu d'un là où elle l'est. Pour recourir à la validation croisée, il faudrait donc fixer la versatilité de chaque expert indépendamment de celles des autres.

Rappelons cependant que la validation croisée est un procédé coûteux en ressources. Elle requiert d'une part un nombre important de données (pour pouvoir constituer à la fois le jeu de données d'entraînement et celui de validation), données qui sont coûteuses à acquérir dans un contexte où chaque point de données est le résultat d'une interaction entre le robot et son environnement pouvant prendre quelques secondes voire quelques dizaines de secondes. La validation croisée est d'autre part coûteuse en puissance de calcul, car chaque évaluation d'une instance d'hyper-paramètres implique de réaliser une régression sur le jeu d'entraînement entier ainsi que d'estimer la fonction en chaque point du jeu de données de validation. La taille de l'espace de recherche augmentant exponentiellement avec le nombre d'hyper-paramètres,

fixer la versatilité de chaque expert indépendamment de celles des autres devient rapidement prohibitif. La validation croisée semble donc peu adaptée à l'utilisation que nous souhaitons en faire, et nous allons par conséquent nous orienter vers une autre approche.

4.1.3 Utilisation de la complexité : le rasoir d'Ockham

Le rasoir d'Ockham

Le principe duquel nous allons nous inspirer est un principe connu de longue date, appelé le *rasoir d'Ockham*. Le rasoir d'Ockham est un rasoir philosophique (c'est-à-dire d'un principe permettant de trancher entre diverses explications d'un même phénomène) aussi appelé "principe de simplicité", "principe d'économie" ou "principe de parcimonie", et qui s'exprime souvent en ces termes : "lorsque diverses explications permettent de rendre compte d'un ensemble observations, il faut privilégier la plus simple". Le rasoir d'Ockham a servi d'inspiration à de nombreux travaux, menant à diverses façons d'exprimer la complexité d'un modèle (qui lui donne sa versatilité), ainsi qu'à divers résultats théoriques s'appuyant sur celles-ci.

Il est important de noter que toutes les définitions de complexité ne se valent pas, et que certaines peuvent revêtir un caractère arbitraire. De plus, des familles de fonctions ayant une forte complexité pour certaines définitions peuvent en avoir une faible pour d'autres, et inversement. On pourrait par exemple comparer la VC-dimension (voir plus loin) d'une famille de fonctions à une complexité arbitraire telle que le nombre de paramètres à l'aide duquel les fonctions de cette famille s'expriment. À titre d'exemple, si on suppose des fonction $\mathbb{R} \to \mathbb{R}$ permettant de choisir la classe du point x parmi deux classes selon le signe du résultat, la famille des fonctions $x \mapsto sin(\alpha \cdot x)$ avec $\alpha \in \mathbb{R}$ correspond à une VC-dimension infinie bien qu'il n'y ait qu'un seul paramètre, alors que la familles des fonctions $x \mapsto \alpha \cdot x + \beta + \gamma$ avec $\alpha, \beta, \gamma \in \mathbb{R}$ correspond à une VC-dimension égale à 2 bien qu'il y ait cette fois 3 paramètres.

Notons cependant que l'arbitraire de certaines définitions de complexité ne les empêche pas nécessairement de coïncider en pratique avec d'autres définitions plus rigoureuses. Ainsi, si on se restreint à une certaine forme de familles de fonctions comme la famille des série de Fourier de rang i ou moins (avec i un entier propre à chaque famille), une mesure de complexité telle que le nombre de paramètres augmente avec le rang maximal i et permet donc d'exprimer correctement les complexités relatives des différentes familles de fonctions.

La dimension de Vapnik-Chervonenkis

La définition de complexité la plus connue est la *dimension de Vapnik-Chervonenkis* (généralement abrégée en *VC-dimension* en anglais), qui permet

d'estimer la capacité d'un algorithme de classification à obtenir de bonnes performances sur de nouvelles données (Vapnik [1999]).

La VC-dimension d'une famille de fonctions est définie comme le cardinal du plus grand ensemble de points de leur domaine de définition qui peut être *pulvérisé* par cette famille de fonctions, c'est-à-dire le cardinal du plus grand ensemble de points tel que quelles que soient les catégories auxquelles sont assignés ces points il existe une fonction dans cette famille qui les classifie correctement (voir Figure 4.2 pour un exemple). Informellement, la VC-dimension chiffre la capacité à "déformer" le classifieur (en se déplaçant au sein de la famille de fonctions) pour changer la catégorisation dans le voisinage d'un point donné sans la changer dans le voisinage des autres points.

Lorsqu'un classifieur est confronté à de nouvelles données, sous la forme d'un jeu de test tiré indépendamment du jeu de données d'entraînement et identiquement distribué, Vapnik a prouvé que l'on a pour tout $\eta \in [0, 1]$:

$$\mathbb{P}\left(E_{test} \leq E_{entrainement} + \sqrt{\frac{D\left(log(\frac{2N}{D}) + 1\right) - log(\frac{\eta}{4})}{N}} \right) = 1 - \eta$$

où E_{test} est l'erreur sur le jeu de test, $E_{entrainement}$ l'erreur sur le jeu d'entraînement, N le nombre de points de données d'entraînement et D la VC-dimension de la famille de fonction qui sert de modèle pour cet apprentissage.

On retrouve là de façon plus rigoureuse le principe du rasoir d'Ockham : pour une même erreur d'entraînement, une complexité (VC-dimension) plus faible est préférable car elle mène à une borne supérieure probabiliste plus faible de l'erreur sur le jeu de test (*i.e.* une meilleure capacité de généralisation).

Autres utilisations du rasoir d'Ockham

Les travaux inspirés du rasoir d'Ockham sont nombreux. On pensera notamment à tout ce qui concerne le domaine la théorie de la distorsion du débit (*Rate Distortion Theory* en anglais), une branche de la théorie de l'information qui s'intéresse à la question de la représentation d'un signal avec aussi peu de bits que possible d'une façon qui permette de le reconstruire sans dépasser une certaine distorsion (Shannon [1959]). L'enjeu y est donc de chercher la solution la plus simple (le nombre de bits mesurant la complexité) capable de rendre compte des données (borne supérieure sur l'erreur).

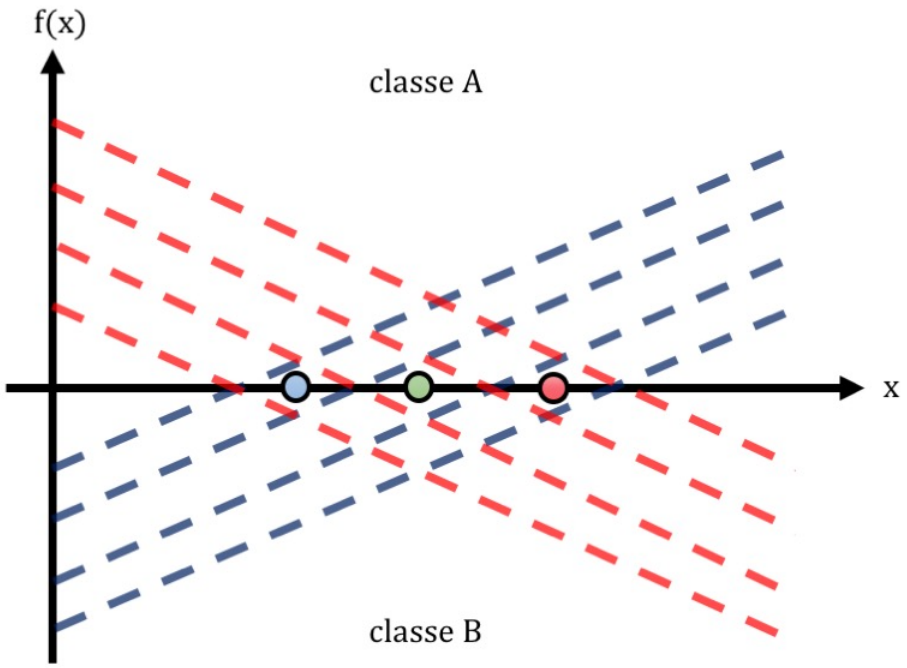


FIGURE 4.2 – Illustration de la VC-dimension sur un cas d'exemple où l'on souhaite apprendre une fonction dont le signe permette de répartir des points de données ayant une unique dimension x en deux classes, et où l'on cherche cette fonction au sein de la famille $\mathscr{F}$ des fonctions de la forme $f : x \mapsto ax + b$. Si on considère un ensemble de deux points distincts (points bleu et rouge sur le schéma), les fonctions dessinées en pointillés bleus permettent d'assigner à la classe A les sous-ensembles $\emptyset$, {bleu}, ou {bleu, rouge}, tandis que les fonctions dessinées en pointillés rouges permettent d'y assigner les sous-ensembles $\emptyset$, {rouge}, ou {bleu, rouge}. L'ensemble {bleu, rouge} est donc pulvérisé par $\mathscr{F}$: des fonctions de $\mathscr{F}$ permettent d'assigner la classe A à n'importe lequel de ses sous-ensembles (et la classe B au reste).
Si on considère maintenant un ensemble de trois points (points bleu, vert et rouge), les fonctions en pointillés permettent d'assigner à la classe A les sous-ensembles $\emptyset$, {bleu}, {rouge}, {bleu, vert}, {vert, rouge}, {bleu, vert, rouge} mais pas les sous-ensembles {vert} ou {bleu, rouge}. L'ensemble {bleu, vert, rouge} n'est donc pas pulvérisé par $\mathscr{F}$. La taille maximale d'un ensemble de points pulvérisé par $\mathscr{F}$, sa VC-dimension, est par conséquent de 2.

Plus récemment dans le domaine de la planification de mouvement, Lemme *et al.* [2014] utilise une bibliothèque de primitives de mouvement dynamiques (*Dynamic Motion Primitives* en anglais, généralement abrégé en DMP) pour approximer des trajectoires observées. Pour cela, il minimise le nombre de DMPs utilisées sous la contrainte que l'erreur doit rester en-dessous d'un certain seuil. Là encore, parmi les solutions possibles (délimitées par le seuil sur l'erreur), celle qui est choisie est celle dont la complexité est la plus faible (mesurée par le nombre de DMPs utilisées ; les DMPs de la bibliothèque peuvent en elles-mêmes être composés de plusieurs DMPs plus simples, mais celle-ci est construite de façon à ne pas contenir de DMPs complexes si des DMPs plus simples ont suffi à obtenir de bonnes performances sur les trajectoires vues jusque là, ce qui est également une forme de minimisation de complexité dans la limite d'une erreur acceptable).

Application à l'approximation de la fonction échelon

Revenons à la question qui nous intéresse : en quoi le rasoir d'Ockham pourra t-il nous aider à traiter la question du compromis entre nombre et versatilité des experts ? Comme nous l'avons vu en section 4.1.1, la difficulté à laquelle nous sommes confrontés est de choisir un mélange d'experts parmi les diverses possibilités qui permettent de rendre compte des points de données d'entraînement. Ce que nous fournit le rasoir d'Ockham, c'est un critère de choix : parmi ces mélanges d'experts, on choisira celui dont la complexité est la plus faible.

Vérifions que ce critère de choix fonctionne sur notre exemple de l'approximation d'une fonction échelon par un mélange d'approximations par des séries de Fourier (possiblement constantes) : minimiser la complexité nous permet-il d'éviter les écueils dont nous avons parlé plus haut ?

Tout d'abord, nous savons que la solution que nous cherchons à obtenir (mélange de deux experts constants, de part et d'autre de la discontinuité) est préférable pour ce critère à un mélange d'un même nombre d'experts plus versatiles, de par le principe même de minimisation de complexité. Similairement, si on se limite à des experts constants, minimiser la complexité revient à minimiser le nombre d'experts et la solution est donc préférable pour ce critère à tout mélange d'experts constants mais plus nombreux (tel que celui résultant de la méthode des k plus proches voisins avec $k = 1$).

Qu'en est-il des mélanges d'experts dont à la fois le nombre et la complexité des experts sont différents de la solution souhaitée ? Nous ne nous attarderons pas sur le fait que celle-ci est préférable pour le critère de complexité à tout mélange d'experts à la fois plus nombreux et plus complexes, et allons donc considérer le cas d'un mélange d'experts moins nombreux mais plus complexes, c'est-à-dire ici celui d'une unique série de Fourier comportant de nombreuses harmoniques. La meilleure série de Fourier au sens de notre critère est celle

qui compte le moins d'harmoniques parmi celles qui peuvent rendre compte des données d'entraînement. On supposera que notre définition de complexité est telle qu'il existe un nombre d'harmoniques au-dessus duquel une unique série de Fourier est considérée comme plus complexe qu'un mélange de deux experts constants (dans le cas contraire, on aboutira toujours à l'expert unique le plus simple permettant de rendre compte des données et jamais à un mélange de plusieurs experts). Or, une série de Fourier ne pouvant parfaitement approximer une fonction échelon, il est toujours possible d'ajouter des points au jeu d'entraînement qui augmentent le nombre d'harmoniques nécessaires à ce qu'elle puisse rendre compte de ces points. Il s'ensuit qu'avec un jeu d'entraînement assez grand et divers, l'approximation d'une fonction échelon par une unique série de Fourier mènera toujours à une complexité plus élevée qu'un mélange de deux experts constants : la solution souhaitée est bien celle qui est la meilleure pour une minimisation de complexité.

4.2 Hypothèses et définitions

Nous allons donc nous inspirer du rasoir d'Ockham pour notre algorithme de mélange d'experts, et utiliser les sauts de complexité induits par les ruptures de régularité de la fonction à approximer pour découper l'espace en régions. Avant de rentrer dans le vif du sujet, prenons le temps de poser les bases sur lesquelles l'algorithme est construit.

4.2.1 Retour sur les hypothèses

En section 2.4.2, nous avions fait l'hypothèse que le modèle direct global stochastique qui décrit les interactions du robot avec son environnement était la somme d'un terme déterministe f_{dir} supposé $\mathscr{C}^\infty$ localement mais pas globalement, et d'un terme stochastique Δ (hypothèse 1).

Nous avions ensuite supposé l'existence d'une borne déterministe B sur Δ qu'il est possible de connaître ou d'estimer (hypothèse 2). Cette hypothèse nous permet de supposer qu'une interaction de l'agent avec son environnement résulte en un quadruplet $(\overline{\delta}, \overline{\pi}, \overline{r}, \varepsilon)$, composé du domaine dans lequel se trouvait le robot au début de cette interaction, de la politique qu'il a exécutée, de l'effet qui a résulté de l'exécution de celle-ci, et de la variabilité de cet effet liée à l'imprécision des mesures (ε étant donc une estimation de $B(\overline{\delta}, \overline{\pi})$).

Étant donné que nous développons notre algorithme de mélange d'experts inspiré du rasoir d'Ockham pour pouvoir l'appliquer à la recherche des composantes $\mathscr{C}^\infty$ de f_{dir}, nous pouvons supposer que nos donnée d'entraînement sont des quadruplets tels que décrits ci-dessus, ce qui signifie dans le formalisme de régression que nos données d'entraînement ne sont pas simplement de la forme $(x_i, t_i)_{i \in [\![0,N[\![}$, mais de la forme $(x_i, t_i, \varepsilon_i)_{i \in [\![0,N[\![}$ où les ε_i expriment

l'imprécision sur les valeurs des t_i.

4.2.2 Complexité

Dans nos travaux, nous définirons la complexité de la façon suivante :

Définition 4.1. Soit S un ensemble de familles de fonctions. On appelle *fonction de complexité sur* S toute fonction c de la forme $c : S \to \mathbb{N}^*$.

Dans cette définition, S est un ensemble de familles de fonctions quelconque, par exemple l'ensemble contenant la famille des fonctions polynomiales de degré 3 ou moins sur les deux premières dimensions de l'espace des x, la famille des séries de Fourier de rang 3 ou moins sur une des dimensions de l'espace des x, et la famille des fonction qui s'expriment comme la somme de 5 fonctions gaussiennes sur les dimensions d'indice pair de l'espace des x.

Une fonction de complexité est simplement une fonction définie sur un tel ensemble et qui associe à chaque famille de fonctions $\mathscr{F}$ qui le composent un entier non nul $c(\mathscr{F})$ qu'on appelera la *complexité* de cette famille de fonctions. Il s'agit là d'une définition assez générale de la complexité, qui pose celle-ci comme une propriété des familles de fonctions et non des fonctions individuelles.

Une telle fonction de complexité pourra ainsi être la VC-dimension, mais nous ne nous restreindrons pas ici à une fonction de complexité en particulier. À titre d'exemple, si on considère l'ensemble qui contient pour chaque $i \in \mathbb{N}$ la famille des séries de Fourier de rang i ou moins sur une des dimensions de l'espace des x, une fonction de complexité sur cet ensemble pourra aussi être la fonction qui associe chacune de ces familles la complexité $i + 1$ (nota : pas simplement i car cela impliquerait d'associer une complexité 0 à la famille des séries de Fourier de rang 0, *i.e.* des fonctions constantes).

Pour pouvoir parler de la complexité d'une fonction individuelle, nous nous appuierons sur la définition suivante :

Définition 4.2. Soit S un ensemble de familles de fonctions. On note $\hat{S}$ l'ensemble des fonctions qui appartiennent à une famille de S. Toute fonction de complexité c sur S induit sur $\hat{S}$ une *fonction de complexité fonctionnelle* $\hat{c}$ (que l'on notera simplement c lorsqu'il n'y a pas d'ambigüité) définie de la façon suivante :

$$\forall\, f \in \hat{S},\ \hat{c}(f) = \min_{\substack{\mathscr{F} \in S \\ t.q.\ f \in \mathscr{F}}} c(\mathscr{F})$$

En d'autres termes, la complexité d'une fonction individuelle est la complexité de la famille de fonctions la plus simple à laquelle elle appartient, ce qui est illustré en Figure 4.3.

Revenons pour illustrer cela à l'exemple précédent : l'ensemble S y est celui qui contient pour chaque $i \in \mathbb{N}$ la famille des séries de Fourier de rang i ou

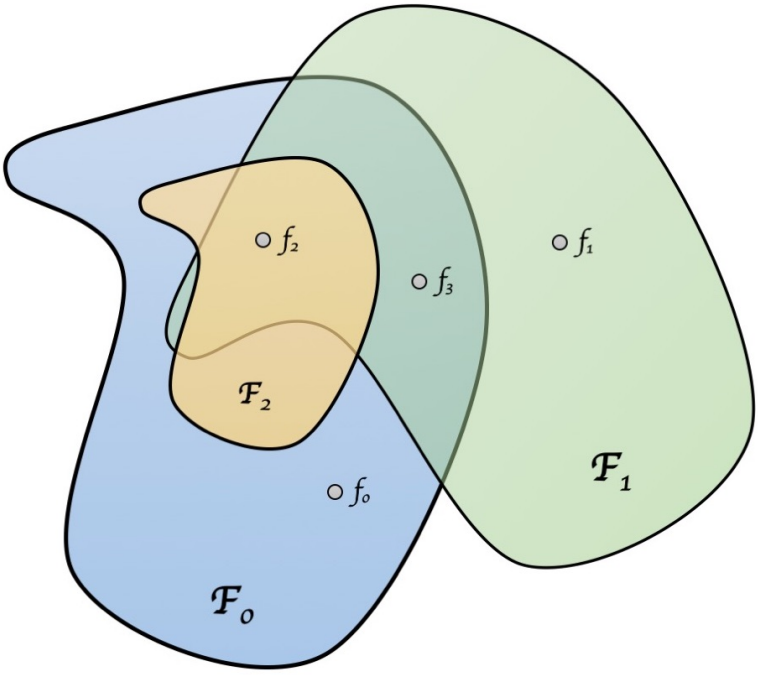


FIGURE 4.3 – On considère un ensemble S composé de 3 familles de fonctions $\mathscr{F}_0$, $\mathscr{F}_1$ et $\mathscr{F}_2$, qui sont d'intersections non vides deux à deux et telles que $\mathscr{F}_2 \subseteq \mathscr{F}_0$. S est muni d'une fonction de complexité c, qui est telle que $c(\mathscr{F}_2) < c(\mathscr{F}_1) < c(\mathscr{F}_0)$. Les fonctions qui n'appartiennent qu'à une seule famille ont une complexité fonctionnelle égale à la complexité de cette famille : $\hat{c}(f_0) = c(\mathscr{F}_0)$ et $\hat{c}(f_1) = c(\mathscr{F}_1)$. Celles qui appartiennent à plusieurs familles ont une complexité fonctionnelle égale à la complexité de la famille plus simple d'entre elles : $\hat{c}(f_2) = \min(\{c(\mathscr{F}_0), c(\mathscr{F}_1), c(\mathscr{F}_2)\}) = c(\mathscr{F}_2)$ et $\hat{c}(f_3) = \min(\{c(\mathscr{F}_0), c(\mathscr{F}_1)\}) = c(\mathscr{F}_1)$

moins sur une des dimensions de l'espace des x, et la fonction de complexité c celle qui associe à une telle famille la complexité $i + 1$. Dans cet exemple, l'ensemble $\hat{S}$ contient toutes les fonctions s'exprimant comme une série de Fourier sur une des dimensions de l'espace des x, et c induit sur celui-ci une complexité $\hat{c}$ qui attribue à toute série de Fourier de rang j une complexité $j+1$ (car cette série de Fourier appartient à toutes les familles des séries Fourier de rang i ou moins avec $i \geq j$, qui sont associées à des complexités $i + 1$).

En ce qui concerne les mélanges d'experts, on définira leur complexité de la façon suivante :

Définition 4.3. La *complexité d'un mélange d'experts* est égale à la somme des complexités des experts qui le composent.

Toujours dans notre cas d'exemple, on retrouve donc bien là qu'un mélange de 2 experts constants est associé à une complexité plus faible (égale à 2) que

le mélange d'experts issu d'un algorithme du plus proche voisin sur N points de données (complexité N), ou qu'une unique série de Fourier de rang k élevé (complexité k), conformément à ce qui a été dit en section 4.1.3.

4.2.3 Rendre compte de données

Le rasoir d'Ockham étant une minimisation de complexité parmi les explications qui rendent compte de toutes les observations, il nous faut également préciser le sens de cette contrainte :

Définition 4.4. Soit un jeu de N points de données $S = (x_i, t_i, \varepsilon_i)_{i \in [\![0,N[\![}$. On dit qu'une fonction f *rend compte de* S (ou qu'elle *explique* S) si elle vérifie :

$$\forall\, i \in [\![0, N[\![,\ |t_i - f(x_i)| \leq \varepsilon_i$$

Si toutes les bornes sur le bruit sont strictement positives, on peut formuler une définition alternative :

Définition 4.5. Soit un jeu de N points de données $S = (x_i, t_i, \varepsilon_i)_{i \in [\![0,N[\![}$ tel que $\varepsilon_i > 0$ pour tout i. On dit qu'une fonction f *rend compte de* S (ou qu'elle *explique* S) si elle vérifie $\|E(f)\|_\infty \leq 1$, où $E(f)$ est un vecteur d'erreur tel que

$$\forall\, i,\ E(f)_i = \left| \frac{t_i - f(x_i)}{\varepsilon_i} \right|$$

4.3 COCOTTE : algorithme et implémentation

Nous allons à présent présenter notre algorithme, appelé COCOTTE (pour COnstrained Complexity Optimisation Through iTerative merging of Experts). Comme dit précédemment, celui-ci s'inspire du rasoir d'Ockham et cherche à obtenir un mélange d'experts de complexité minimale parmi ceux qui rendent compte des points de données.

L'idée générale derrière COCOTTE est de partir d'un mélange d'experts atomique, où chaque point de données se voit attribuer un expert dédié (similairement à ce que fournirait un algorithme du plus proche voisin), puis de remplacer itérativement ces approximations locales disjointes par des approximations conjointes (fusion d'experts) afin d'obtenir des approximations de moins en moins locales et de plus en plus complexes. Localement, chaque expert doit alors rendre compte des points qui lui sont associés avec une complexité minimale, et le saut de complexité qu'entraînerait la fusion de deux experts de part et d'autre d'une rupture de régularité permet de détecter celle-ci et d'empêcher la fusion.

Plutôt que de rentrer directement dans le détail du fonctionnement de COCOTTE (qui sera donné en section 4.3.4), nous commencerons par nous

intéresser aux sous-problèmes qu'il est nécessaire de savoir résoudre pour pouvoir écrire notre algorithme. Ainsi, nous verrons d'abord comment déterminer si une famille de fonctions contient au moins une fonction qui permette de rendre compte d'un jeu de données (*cf.* section 4.3.1). À l'aide de la solution de ce sous-problème, nous verrons ensuite comment rechercher un expert de complexité minimale pour rendre compte d'un tel jeu de données (*cf.* section 4.3.2). Nous verrons alors comment déterminer s'il est pertinent de remplacer deux approximations disjointes par une approximation conjointe (*cf.* section 4.3.3), avant de détailler enfin le fonctionnement de COCOTTE.

Dans la suite, on supposera donnés un ensemble de familles de fonctions S_f, ainsi qu'une fonction de complexité COMP sur cet ensemble. On notera $S_{f,min}$ la complexité minimale au sein de S_f (possiblement 1), et $S_{f,max}$ la complexité maximale au sein de S_f (possiblement infinie) :

$$S_{f,min} = \min_{\mathscr{F} \in S_f} \text{COMP}(\mathscr{F}) \qquad S_{f,max} = \max_{\mathscr{F} \in S_f} \text{COMP}(\mathscr{F})$$

4.3.1 Recherche au sein d'une famille de fonctions

Dans le cadre de notre algorithme, nous allons avoir besoin de déterminer si une famille de fonctions $\mathscr{F} \in S_f$ donnée contient au moins une fonction qui rend compte d'un ensemble de points de données $S_d = (x_i, t_i, \varepsilon_i)_{i \in [\![0,N[\![}$, et le cas échéant de renvoyer une de ces fonctions (par exemple celle qui minimise $\|E(f)\|_\infty$). Ceci sera accompli par GETFITTINGFUNCTION($\mathscr{F}$, S_d), une routine qui renverra une fonction de S_f qui rend compte de S_d s'il en existe, et NONE dans le cas contraire.

Cette routine s'exprime différemment selon la forme des familles de fonctions contenues dans S_f. Dans le cas de familles de fonctions de la forme $t = \sum_j w_j \cdot f_j(x)$, où les f_j sont des fonctions et les w_j des paramètres réels, nous proposons de s'inspirer de l'algorithme de SVR (*cf.* section 3.2.6) et de nous ramener au problème d'optimisation suivant :

$$\underset{\substack{\zeta \in \mathbb{R}^+,\ \omega_0, \dots, \omega_{M-1} \in \mathbb{R} \\ t.q. \forall i \in [\![0,N[\![\ \begin{cases} t_i - \sum_j w_j \cdot f_j(x_i) \leq \zeta * \varepsilon_i \\ \sum_j w_j \cdot f_j(x_i) - t_i \leq \zeta * \varepsilon_i \end{cases}}}{\text{argmin}} \zeta$$

Il s'agit alors d'un simple programme linéaire, qui peut par conséquent être résolu par des algorithmes connus tels que l'algorithme du simplexe ou la méthode du point intérieur. Une fois la solution trouvée, la valeur de ζ nous indique si la fonction définie par les w_j rend compte des points de S_d (auquel cas on a $\zeta \leq 1$) ou non (auquel cas on a $\zeta > 1$).

Le Pseudo-code 4.1 donne le détail de GETFITTINGFUNCTION pour ce cas particulier. Pour faciliter la lecture, l'utilisation du solveur y est décrite avec une syntaxe orientée objet : le programme linéaire est représenté par un objet créé à l'aide de GENERATELINEARPROGRAM, qui dispose d'une méthode

addVariables() permettant d'ajouter des variables, de méthodes setCostFunction() et setConstraints() pour définir respectivement la fonction de coût et les contraintes (exprimées à l'aide de ces variables), ainsi que d'une méthode solve() pour le résoudre et renvoyer la solution. On supposera par ailleurs disposer d'une routine APPLYPRIMITIVES() qui évalue chacune des fonctions f_j en un point donné, et d'une routine GENERATEFUNCTION qui renvoie la fonction de $\mathscr{F}$ associée à une instance des w_j.

Pseudo-code 4.1 La routine GETFITTINGFUNCTION, quand $\mathscr{F}$ est une famille de combinaisons linéaires de fonctions primitives $(f_j)_{i \in [\![0,M[\![}$

```
function GETFITTINGFUNCTION(𝓕, S_d)
    (x, t, ε) ← S_d
    LP ← GENERATELINEARPROGRAM()
    (ζ, ω_0, ... , ω_{M−1}) ← LP.addVariables(M+1)
    LP.setCostFunction(1 · ζ)
    constraints ← ∅
    constraints.append(ζ ≥ 0)
    for i ∈ [[0, N[[ do
        (f_{i,0}, ..., f_{i,M−1}) ← APPLYPRIMITIVES(𝓕, x[i])      ▷ calcul des f_j(x_i)
        constraints.append(t[i] − Σ_j w_j · f_{i,j} ≤ ζ · ε[i])
        constraints.append(Σ_j w_j · f_{i,j} − t[i] ≤ ζ · ε[i])
    end for
    LP.setConstraints(constraints)
    sol ← LP.solve()
    if sol.get(ζ)> 1 then
        return NONE
    else
        return GENERATEFUNCTION(𝓕, sol.get(ω_0), ..., sol.get(ω_{M−1}))
    end if
end function
```

4.3.2 Recherche d'un expert de complexité minimale

Un autre sous-problème que nous allons rencontrer dans le cadre de notre algorithme est le suivant : étant donné un ensemble de point S_d, quelle est la fonction de $\hat{S}_f$ de complexité minimale permettant de rendre compte de ces points ?

Nous allons donc écrire une routine GETBESTFIT pour résoudre ce problème. En plus de S_f et S_d, celle-ci prendra en argument deux entiers c_{min} et c_{max} permettant de spécifier respectivement une complexité minimale pour la solution que l'on cherche (qui vaudra par défaut $S_{f,min}$ mais pourra prendre des valeurs plus élevées si l'on sait déjà qu'il n'y a pas de solution en-dessous

d'une certaine complexité) et une complexité maximale pour celle-ci (qui vaudra par défaut $S_{f,max}$ mais pourra prendre des valeurs plus basses si l'on décide de rejeter les fonctions ayant une complexité plus élevée que celle du mélange d'experts qu'elles visent à remplacer). Après avoir déterminé la solution, GETBESTFIT renverra un couple (c, f), où f est la fonction de complexité minimale et où c est cette complexité minimale. Si aucune solution n'est trouvé, NONE sera renvoyé.

Une approche naïve consiste à parcourir les familles de fonctions de S_f par ordre croissant de complexité, et d'utiliser GETFITTINGFUNCTION pour déterminer si ces familles contiennent une fonction qui permet de rendre compte des points. Ceci est illustré en Pseudo-code 4.2.

Pseudo-code 4.2 Algorithme naïf pour la routine GETBESTFIT

```
function getBestFit(S_f, S_d, c_min = S_{f,min}, c_max = S_{f,max})
    sortByComplexity(S_f)
    for 𝓕 ∈ S_f do
        f ← getFittingFunction(𝓕, S_d)
        if f ≠ None then
            c ← comp(𝓕)
            return (c, f)
        end if
    end for
    return None
end function
```

On peut toutefois remarquer qu'au sein de notre ensemble S_f de familles de fonctions, certaines familles peuvent être incluses dans d'autres plus complexes. Par exemple, la famille des séries de Fourier de rang 3 ou moins de la première dimension de x est incluse dans celles des séries de Fourier de rang 4 ou moins de cette première dimension. Si une famille de fonctions ne suffit pas à rendre compte de S_d, on peut donc en déduire que toutes les familles incluses dans celle-ci sont également incapables de rendre compte de S_d.

Ceci nous permet donc d'adopter une approche plus efficace de recherche par dichotomie : pour explorer un intervalle entier de complexités possibles $[\![\text{low}, \text{high}]\!]$, nous déterminerons un intervalle entier $I = [\![\text{rangeMin}, \text{rangeMax}]\!]$ de petite taille et qui coupe $[\![\text{low}, \text{high}]\!]$ en sous-intervalles $[\![\text{low}, \text{rangeMin}[\![$ et $]\!]\text{rangeMax}, \text{high}]\!]$ de tailles similaires. Nous chercherons ensuite une fonction permettant de rendre compte de S_d au sein des familles de fonctions dont la complexité est dans I. Si de telles fonctions sont trouvées, on pourra éliminer l'ensemble $]\!]\text{rangeMax}, \text{high}]\!]$ (qui ne contient que des fonctions plus complexes que les solutions trouvées) et restreindre la recherche à $[\![\text{low}, \text{rangeMin}[\![$. Dans le cas contraire, on pourra à l'inverse éliminer $[\![\text{low}, \text{rangeMin}[\![$ et restreindre la recherche à $]\!]\text{rangeMax}, \text{high}]\!]$, sous-réserve que I soit choisi de telle manière

que si aucune fonction de $\{f \in \hat{S}_f \mid \text{COMP}(f) \in I\}$ n'explique S_d, alors aucune fonction de $\hat{S}_f$ de complexité inférieure ou égale à rangeMax non plus. Ceci est illustré en Figure 4.4.

FIGURE 4.4 – On cherche un intervalle $I = [\![\text{rangeMin, rangeMax}]\!]$ inclus dans $[\![\text{low, high}]\!]$ tel que si les fonctions dont la complexité est dans I n'expliquent pas S_d, alors aucune fonction dont la complexité est dans $[\![\text{low, rangeMin}[\![$ n'explique S_d non plus. Dans ce cas, on continue la recherche dans $]\!]\text{rangeMax, high}]\!]$. À l'inverse, si des solutions sont trouvées, on continue la recherche dans $[\![\text{low, rangeMin}[\![$ pour déterminer si d'autres solutions de complexité plus faible peuvent être trouvées.

Pour obtenir un intervalle $I = [\![\text{rangeMin, rangeMax}]\!]$ vérifiant cette propriété, on utilisera une routine COMPLEXITYRANGEMIN(S_f, rangeMax) qui, étant donné une valeur de rangeMax, renvoie rangeMin comme la complexité minimale parmi les familles de fonctions de S_f qui ne sont incluses dans aucune autre et ont une complexité inférieure ou égale à rangeMax (elle renverra 0 s'il n'y en a aucune). Ceci est illustré en figure 4.5.

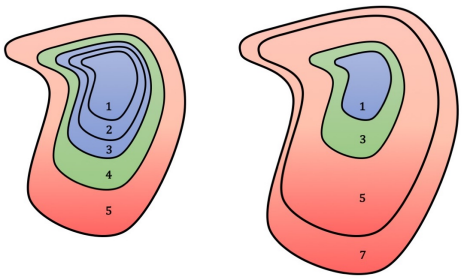


FIGURE 4.5 – Illustration de la routine COMPLEXITYRANGEMIN(S_f, rangeMax) lorsque rangeMax vaut 4 et que S_f contient 5 familles de fonctions de complexités respectives 1, 2, 3, 4 et 5 incluses les unes dans les autres, ainsi que 4 familles de fonctions de complexités respectives 1, 3, 5 et 7 également incluses les unes dans les autres. COMPLEXITYRANGEMIN s'intéresse aux familles de fonctions de S_f de complexité inférieure à rangeMax (on exclut donc les familles représentées ici en rouge) et parmi celles-ci uniquement à celles qui ne sont incluses dans aucune autre (on exclut donc les familles représentées ici en bleu). La routine renvoie alors la complexité minimale parmi les familles restantes (représentées ici en vert), c'est à dire 3.

Nous supposerons également que nous avons accès à une routine GETFAMILIES(S_f, c_{min}, c_{max}), qui renvoie la liste des ($\mathscr{F}$, COMP($\mathscr{F}$)), triée par complexité croissante, où les $\mathscr{F}$ sont les familles de S_f dont la complexité est comprise entre c_{min} et c_{max} inclus.

Un pseudo-code simplifié de GETBESTFIT est donné en Pseudo-code 4.3, et une version plus détaillée est disponible annexe (Pseudo-code A.1).

Pseudo-code 4.3 Algorithme de recherche dichotomique pour la routine GETBESTFIT

```
function getBestFit(S_f, S_d, c_min = S_f,min, c_max = S_f,max)
    ▷ 'low' et 'high' sont les bornes sur la complexité pour la recherche
    ▷   dichotomique, et 'bestSol' contient la solution à renvoyer

    initialiser low, high, bestSol (avec low=c_min et high=c_max)
    while low ≤ high do
        déterminer rangeMax, le milieu de [low, high]
        en déduire rangeMin à l'aide de complexityRangeMin
        utiliser getFamilies pour obtenir la liste des familles
          de fonctions de complexité dans [rangeMin, rangeMax]
        utiliser getFittingFunction pour chercher la première solution
          dans cette liste
        if solution trouvée then
            mettre à jour bestSol
            high ← rangeMin - 1
        else
            low ← rangeMax + 1
        end if
    end while

    return bestSol

end function
```

4.3.3 Fusion d'experts

Nous arrivons maintenant à un sous-problème central pour notre algorithme itératif : étant donnés deux jeux de données disjoints $S_{d,0}$ et $S_{d,1}$, ainsi que deux couples $(c_0, f_0) = \text{GETBESTFIT}(S_f, S_{d,0})$ et $(c_1, f_1) = \text{GETBESTFIT}(S_f, S_{d,1})$, peut-on réduire la complexité globale du mélange d'expert en remplaçant les approximations f_0 et f_1 (qui sont locales aux sous-espaces couvrant respectivement $S_{d,0}$ et $S_{d,1}$) par une unique approximation f_{0+1} sur $S_{d,0+1} = S_{d,0} \cup S_{d,1}$ dont on notera la complexité c_{0+1}.

Nous avons défini plus haut la complexité d'un mélange d'experts comme la somme des complexités des experts qui le composent, ce qui nous permet donc de formuler une réponse à cette question : nous préférerons les approximations locales si $c_{0+1} > c_0 + c_1$ (saut de complexité), et l'approximation unique si $c_{0+1} < c_0 + c_1$. En cas d'égalité les deux alternatives sont valides et nous choisirons dans ce cas l'approximation unique (une raison à cela est que lorsque l'on fusionne des experts deux à deux, comme nous le ferons, le choix contraire peut empêcher toute émergence d'un expert de complexité supérieure à la complexité minimale).

On peut donc écrire une routine MERGEEXPERTS(S_f, expert0, expert1), qui prend en argument S_f ainsi que les deux triplets $(S_{d,0}, c_0, f_0)$ et $(S_{d,1}, c_1, f_1)$, et renvoie le triplet $(S_{d,0+1}, c_{0+1}, f_{0+1})$ si l'approximation unique est préférable et NONE dans le cas contraire. Le Pseudocode 4.4 donne le détail de MERGEEXPERTS.

Pseudo-code 4.4 La routine MERGEEXPERTS

```
function MERGEEXPERTS(S_f, expert0, expert1)
    (S_{d,0}, c_0, f_0) ← expert0
    (S_{d,1}, c_1, f_1) ← expert1
    S_{d,0+1} ← S_{d,0} ∪ S_{d,1},
    ▷ Il faut au moins une complexité c_0 pour rendre compte de S_{d,0} ⊆ S_{d,0+1},
    ▷ et une complexité c_1 pour rendre compte de S_{d,1} ⊆ S_{d,0+1}
    c_min = max(c_0, c_1)
    ▷ Complexité max pour laquelle on préfère l'approximation unique
    c_max = c_0 + c_1
    ▷ Recherche de f_{0+1}
    newExpert ← GETBESTFIT(S_f, S_{d,0+1}, c_min, c_max)
    if newExpert ≠ NONE then
        (c_{0+1}, f_{0+1}) ← getBestFit
        return (S_{d,0+1}, c_{0+1}, f_{0+1})
    else
        return NONE
    end if
end function
```

4.3.4 COCOTTE

À l'aide des fonctions décrites plus haut, nous pouvons désormais écrire notre algorithme, appelé COCOTTE (pour COnstrained Complexity Optimisation Through iTerative merging of Experts). Il s'agit d'un algorithme itératif en quatre étapes, dont les étapes sont illustrées en Figure 4.6 dans le cas de l'approximation d'une fonction créneau.

Passons maintenant en revue chacune des étapes de COCOTTE (nota : une version détaillée de l'algorithme est donnée en annexe, en Pseudo-code A.2) :

1. Dans un premier temps, on associe à chaque point de données un expert de complexité $S_{f,min}$, similairement à ce que l'on pourrait obtenir via l'algorithme du plus proche voisin. On a donc dès cette étape un mélange d'experts rendant compte des points de données.
2. Dans un second temps, on suit une approche gloutonne : on choisit la paire d'experts les plus proches (pour une certaine notion de distance), et on essaie de les fusionner à l'aide de MERGEEXPERTS. Si la fusion réussit, le nouvel expert les remplace. Dans le cas contraire, on marque la paire comme non valide et on passe à la suivante. Cette étape repose sur l'idée que — à supposer des points de données suffisamment denses et une mesure de distance adaptée — les fusions d'experts se feront d'abord localement avant de se faire à plus grande échelle. Ceci donne la possibilité à des approximations reflétant la structure locale des données d'émerger (ce qui correspond à des fusions d'experts avec $c_{0+1} < c_0 + c_1$), ce qui rend plus difficile les fusions d'experts suivantes (complexité maximum autorisée plus faible pour l'approximation unique) et en particulier celles ne reflétant pas la structure de données.
3. Dans un troisième temps, on procède à une étape d'élimination des artefacts. Lors des premières fusions d'experts, il arrive en effet que certains experts soient fusionnés à tort avant que l'émergence locale de structure puisse empêcher de telles fusions ; ceci se produit notamment au niveau des frontières entre les sous-espaces sur lesquels la fonction à approximer est $\mathscr{C}^\infty$. Pour les éliminer, on passe en revue chaque expert un par un, et tout expert superflu (c'est-à-dire dont les points peuvent être répartis parmi les autres experts) est considéré comme un artefact et est éliminé. Plus précisément, pour déterminer si un expert donné E est un artefact, on crée une copie de notre mélange d'experts dans laquelle E est remplacé par un ensemble d'experts obtenu en associant à chaque point de E un expert de complexité $S_{f,min}$ comme à l'étape 1. On procède ensuite à une fusion gloutonne similaire à celle de l'étape 2, mais dans laquelle ces experts individuels ne peuvent être fusionnés entre eux. À l'issue de celle-ci, si tous ces experts individuels ont pu être fusionnés avec d'autres experts, on considère que E est un artefact et le nouveau mélange d'experts remplace l'ancien. Dans le cas contraire, on considère que E n'est pas un artefact et on passe à l'expert suivant.
4. Dans un dernier temps, on entraîne un classifieur pour associer des régions de l'espace à chaque expert à partir de l'ensemble des points dont celui-ci rend compte.

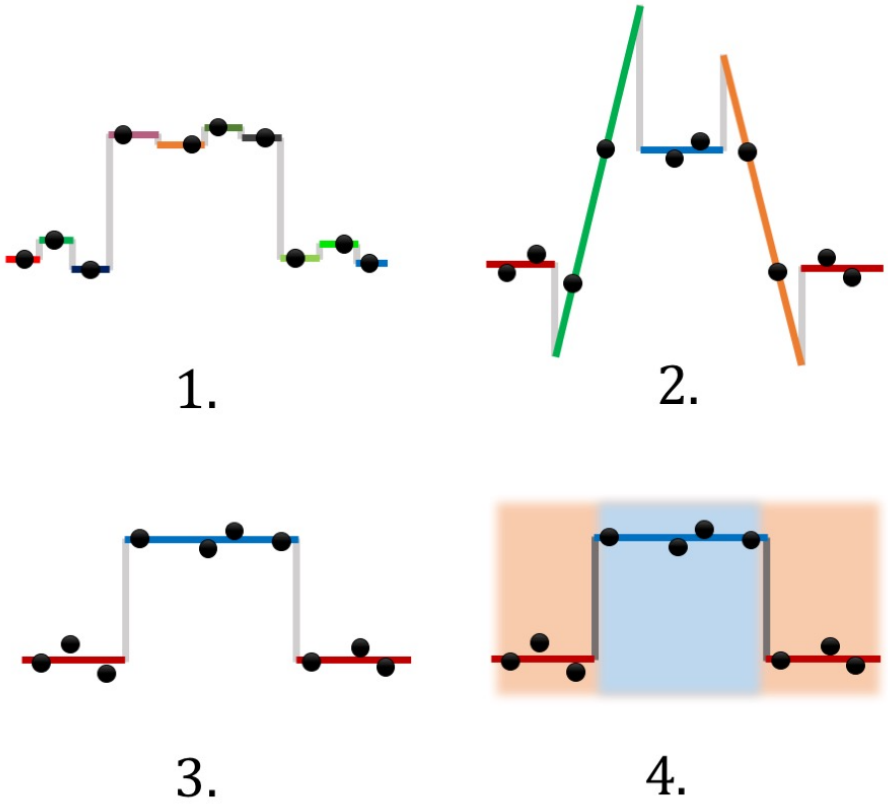


FIGURE 4.6 – Application de l'algorithme COCOTTE à une fonction créneau.
(1.) Lors de l'initialisation, chaque point est expliqué par un expert dédié.
(2.) Lors de la phase de fusion gloutonne, les experts sont fusionnés localement puis progressivement à des échelles plus grandes (*cf.* expert en rouge).
(3.) On élimine ensuite les artefacts, c'est-à-dire les experts dont les points peuvent être expliqués par d'autres experts, et que l'on trouve généralement au niveau des frontières entre les sous-espaces du domaine de définition où la fonction à approximer est régulière (ici les experts vert et orange).
(4.) On finit en entraînant un classifieur pour segmenter l'espace en régions assignées aux experts.

4.3.5 Extension à plusieurs dimensions

L'algorithme décrit ci-dessus s'applique avant tout au cas où la fonction approximée $f : x \mapsto t$ est unidimensionnelle, et la question se pose donc de comment celui-ci doit être étendu aux cas où t comporte plus d'une dimension.

Deux approches sont alors possibles : traiter ces dimensions en bloc, ou séparément. Traiter les dimensions séparément a souvent du sens et pourrait par exemple, dans le cas où t contient la position finale d'un objet et celle du robot, permettre d'apprendre une compétence $\lceil$*déplacer l'objet*$\rfloor$ sans se préoccuper de la position dans laquelle se retrouve le robot à l'issue de l'exécution de celle-ci. Cette approche ne requiert pas de modifier l'algorithme, car elle revient simplement à exécuter celui-ci indépendamment sur chaque dimension.

Il est cependant souvent avantageux de traiter des dimensions en bloc, comme par exemple de traiter les différentes coordonnées d'un objet comme constituant collectivement sa position. Dans cette approche, les experts approximant une fonction à n dimensions seront des n-uplets de $\hat{S_f}^n$, dont chaque élément approxime localement une des dimensions de f. Pour les fusions d'experts, on pourra alors soit associer à chaque expert une complexité donnée par une fonction de complexité sur S_f^n (*e.g.* , la somme des complexités des différentes fonctions du n-uplet), soit associer à chaque expert le vecteur des n complexités et ne réaliser la fusion de deux experts que si on a $c_{0+1} \leq c_0 + c_1$ sur chacune de ces dimensions. Nous avons choisi la seconde option.

4.3.6 Implémentation

Nous avons implémenté COCOTTE en C++ (le code est disponible sur https ://github.com/AdrienMatricon/cocotte), et avons fait le choix d'en faire une implémentation modulaire, qui permet de changer aisément S_f et COMP.

L'ensemble de familles de fonctions que nous avons utilisé pour nos expériences est $S_f = \{P_{k,d} \mid k \in \mathbb{N}, d \subseteq [\![0, D[\![\}$, où D est le nombre de dimensions de x et où $P_{k,d}$ est la famille des fonctions polynomiales de degré au plus k à $|d|$ variables appliquées aux dimensions de x dont les indices sont dans d. Les raisons de ce choix pour l'ensemble S_f sont avant tout pratiques : les $P_{k,d}$ sont des familles de la forme $t = \sum_j w_j \cdot f_j(x)$, et permettent donc d'exprimer GETFITTINGFUNCTION à l'aide du Pseudo-code 4.1. Les fonctions polynomiales sont par ailleurs réputées mener facilement à des situations de sur-entraînement, et il nous a donc paru pertinent de s'intéresser à celles-ci dans le cadre d'une minimisation de complexité qui devrait par construction éviter de telles situations. Le programme linéaire est quant à lui résolu à l'aide de l'implémentation de l'algorithme du simplexe fournie par la bibliothèque *SoPlex* (Wunderling [1997]).

La fonction de complexité sur S_f que nous avons choisie est la fonction qui associe à chaque famille $P_{k,d} \in S_f$ la complexité $k \times |d| + 1$. Il s'agit là d'un

choix relativement arbitraire de notre part, mais qui marche raisonnablement bien en pratique. Pour $|d| = 1$, elle correspond au nombre de points qui peuvent être interpolés par des polynômes de cette famille (qui est aussi le nombre de paramètres). Il s'agit également d'une fonction croissante avec le nombre de variables, ce qui favorise les fonctions qui sélectionnent les variables pertinentes pour le problème. (on notera que le nombre de paramètres de la famille $P_{k,d}$ est $\binom{k+|d|}{k}$, qui est minimal pour $k = 0$ ou $|d| = 0$ et augmente exponentiellement à mesure qu'on se rapproche de $k = |d|$; utiliser le nombre de paramètres comme fonction de complexité mènerait donc à préférer les fonctions polynomiales avec $k = 1$ et $|d|$ élevé (ou l'inverse) à celles offrant un compromis entre degré et nombre de variables).

Pour la fusion gloutonne, nous avons utilisé comme distance entre experts la distance en norme L_1 minimale entre les points de données rattachés à ces experts. Celle-ci marche raisonnablement bien en pratique pour les cas que nous avons considérés, qui comportent un nombre relativement restreint de dimensions. Elle serait en revanche très insuffisante pour des problèmes comportant un grand nombre de dimensions.

En ce qui concerne la détermination des régions, nous avons utilisé l'implémentation venant de la bibliothèque *OpenCV* (Bradski [2000]) d'une forêt d'arbres décisionnels, un algorithme de classification connu (Breiman [2001]). Cet algorithme découpe l'espace en régions le long d'hyperplans correspondant à des valeurs fixes de certaines coordonnées, ce qui lui permet d'ignorer les dimensions qui ne sont pas pertinentes pour le problème, et il nous a donc paru approprié pour déterminer les régions. Nous n'avons cependant pas cherché à obtenir les meilleures régions en jouant sur le choix du classifieur ou de ses hyper-paramètres, et d'autres devraient a priori fonctionner tout aussi bien.

4.4 Limites et améliorations possibles

La principale limite de COCOTTE est son coût en calculs. On procède ainsi dans le pire des cas à N^2 tentatives de fusions d'experts, (où N est le nombre de points de données), chacune nécessitant de tester si des familles de fonctions ayant de l'ordre $\log(c_{max})$ complexités différentes contiennent une solution (où c_{max} est la plus grande complexité qu'un expert se voit attribuer durant l'exécution de l'algorithme). Si dans nos expériences N et c_{max} restent assez faibles (de l'ordre de quelques centaines de points, pour des experts de complexités inférieures à 10), tester une complexité requiert d'appeler de nombreuses fois GETFITTINGFUNCTION($\mathscr{F}, S_d$), une routine coûteuse en calculs car s'appuyant dans notre implémentation sur un solveur linéaire : l'algorithme du simplexe. La plupart des appels à GETFITTINGFUNCTION($\mathscr{F}, S_d$) ne s'intéressant pas réellement à la détermination d'une fonction rendant compte des points mais seulement à l'existence ou non d'une telle solution, des travaux fu-

turs permettraient probablement de remplacer le recours à un solveur linéaire par une alternative plus légère.

Au-delà de ces considérations, le nombre de familles de fonctions sur lequel la routine GETFITTINGFUNCTION($\mathscr{F}, S_d$) est appelée, c'est-à-dire le nombre de familles de fonctions renvoyées par GETFAMILIES(S_f, c_{min}, c_{max}), peut devenir élevé lorsque pour k et $|d|$ fixés il existe de nombreuses familles $P_{k,d}$. En effet, le nombre de combinaisons de $|d|$ dimensions parmi les D dimensions de x est $\binom{D}{|d|}$ (le nombre de combinaisons augmente donc exponentiellement avec le nombre de dimensions non pertinentes au problème).

Pour tenter de remédier à ce problème, nous avons essayé d'adopter une approche gloutonne. Dans celle-ci, lors de la fusion d'un expert appartenant à P_{k_0,d_0} et d'un expert appartenant à P_{k_1,d_1}, on ne cherche le nouvel expert que dans les familles de fonctions $P_{k,d}$ qui sont telles que $|d \setminus (d_0 \cup d_1)| \leq 1$, c'est-à-dire dans les familles de fonctions comprenant au plus une unique dimension hors de $d_0 \cup d_1$. Ceci permettrait de limiter l'exploration à un nombre de combinaisons n'augmentant que linéairement avec le nombre de dimensions non pertinentes au problème. Cette approche s'appuie sur l'idée que les dimensions pertinentes au problème permettent de mieux rendre compte des données que celles qui ne le sont pas. Cette idée ne résiste cependant pas à l'expérience : si les dimensions pertinentes au problème permettent en effet de rendre mieux compte des données que les autres lorsqu'elles sont prises collectivement, il est fréquent qu'un sous-ensemble de ces dimensions pertinentes ne permette pas de rendre aussi bien compte des données qu'un sous-ensemble d'autant de dimensions non-pertinente. Nous avons donc dû renoncer à suivre cette piste.

Une autre faiblesse de COCOTTE est que sa phase de fusion gloutonne s'appuie, comme nous l'avons vu plus haut, sur l'hypothèse que les points sont assez denses pour que les fusions d'experts se fassent d'abord localement puis à plus grande échelle, et permettent ainsi à des structures locales d'émerger et d'empêcher de fusionner des experts à tort. Une alternative qui a été brièvement envisagée serait de procéder pour la fusion des modèles non pas à une approche gloutonne suivie d'une élimination des artefacts mais à une approche de séparation et évaluation (*branch and bound* en anglais) : il s'agirait alors de considérer le choix des paires de modèles à fusionner comme un arbre de décision dont chaque feuille correspond à un mélange d'experts, et d'utiliser les propriétés du problème pour éliminer les branches qui mèneraient à des mélanges d'experts non optimaux avant d'avoir à les explorer. Cette approche permettrait ainsi de se passer de l'hypothèse mentionnée plus haut, au prix d'une complexité algorithmique fortement accrue. Nous avons cependant jugé que le coût en calculs serait alors prohibitif et avons renoncé à nous engager dans cette voie.

4.5 Un début d'amélioration : COCOTTE+

Bien que nous ayons rapidement abandonné l'idée d'une approche de séparation et évaluation en raison de son coût prohibitif en calculs, nous nous sommes penchés sur un autre moyen de limiter la dépendance de l'algorithme à la qualité des premières fusions d'experts. L'idée générale de cette nouvelle approche est d'autoriser les experts à se "voler" des points entre eux si cela permet de réduire la complexité.

4.5.1 Structure de données

Ce que l'on entend par autoriser les experts à se "voler" des points entre eux est de se donner la possibilité d'annuler certaines fusions d'experts pour permettre à l'un d'entre eux de fusionner avec un expert différent. Pour cela, nous allons garder la trace de l'historique des fusions à l'aide d'une collection d'arbres dont les racines contiennent les meilleurs experts trouvés par l'algorithme.

Lors de la phase d'initialisation, chaque expert unitaire (*i.e.* expert de complexité $S_{f,min}$ rendant compte d'un unique point) est donc placé dans un arbre réduit à une unique feuille, qui est ajoutée à la collection. Par la suite, à chaque fois que l'on procède à une fusion d'experts, on crée un nœud qui contient cet expert et dont les fils sont les nœuds qui contiennent les experts fusionnés. Si l'expert résultant de la fusion est jugé préférable aux experts fusionnés, son nœud vient remplacer dans la collection d'arbres ceux des experts fusionnés.

Cette structure nous permet donc d'annuler simplement une fusion d'experts, en remplaçant un nœud par ses fils dans la collection d'arbres.

4.5.2 Fusion d'experts et vol de points

La structure décrite dans la section précédente nous permet désormais d'écrire une routine POINTSTEALING(S_f, leaf0, tree0, leaf1, tree1), qui prend en argument S_f, une feuille leaf0 appartenant à l'arbre tree0, ainsi qu'une feuille leaf1 appartenant à l'arbre tree1, et pourra selon la situation procéder à la fusion des experts de tree0 et tree1 ou permettre à l'expert de tree0 de "voler" le point correspondant à leaf1 (ou l'inverse).

Plus précisément, POINTSTEALING détermine d'abord si une approximation commune est préférable à deux experts distincts pour les points de tree0 et tree1 (*cf.* MERGEEXPERTS). Pour chacune des deux feuilles leaf0 et leaf1, la routine considère ensuite la branche la reliant à la racine de son arbre pour déterminer si le "vol" d'un des nœuds qui la compose (donc des points correspondants à ce sous-arbre) permet de réduire la complexité totale. Pour chaque "vol" possible, deux fusions sont à considérer.

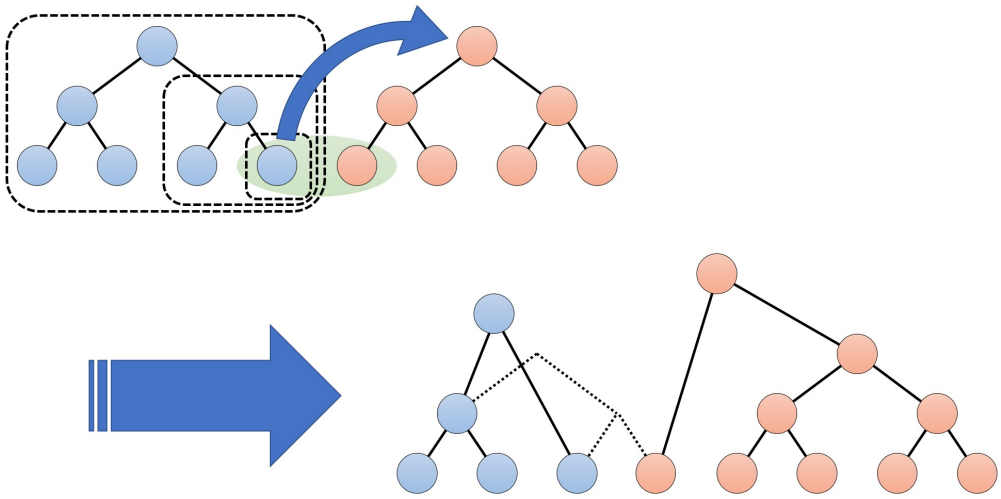


FIGURE 4.7 – Exemple de vol de points entre experts dans COCOTTE+. Plutôt que de s'intéresser à une paire d'experts qu'on ne peut que fusionner ou garder séparés comme dans COCOTTE, on s'intéresse ici à une paire de points que les experts associés peuvent essayer de se "voler" l'un à l'autre. Un tel vol consiste alors à réaliser des fusions d'experts de façon à remanier les deux arbres et à placer la racine de l'un et la feuille contenant l'expert unitaire associée au point de l'autre sous un même nœud. Ainsi, pour que l'expert orange vole le point bleu, il faut le fusionner à un des nœud de la branche allant de la racine de l'arbre bleu à la feuille correspondante. Le nœud en question quitte donc l'arbre bleu avec tous ses descendant, ses parents dans l'arbre bleu sont supprimés (car ils ne correspondent plus à des fusions d'experts), et les nœuds bleus orphelins sont alors réunis en un seul arbre en fusionnant tous les experts qu'ils contiennent.

D'une part, on s'intéresse à la fusion entre les experts correspondant au nœud "volé" et à l'arbre qui le vole, qu'on ne peut fusionner que si cela est pertinent (*cf.* MERGEEXPERTS), et qui résulte en un expert e_0 de complexité c_0. D'autre part, procéder au "vol" de ce nœud implique d'annuler la fusion qu'il représente, ainsi que toutes celles représentées par ses parents dans la branche, et il nous faut donc réunir tous les experts correspondant aux nœuds restants (on suppose disposer d'une routine GETREMAININGNODES donnant accès à ceux-ci) en un seul, ce qui mène à un expert e_1 de complexité c_1 (complexité qui est par construction au plus égale à celle de l'expert associé au nœud racine avant le "vol"). Ceci est illustré en Figure 4.7.

Étant données ces deux fusions, la routine ne peut donc procéder au "vol" que si celui-ci permet de réduire la complexité totale, c'est-à-dire quand $c_0 + c_1$ est strictement inférieur à la complexité totale avant celui-ci. Dans le cas où

Pseudo-code 4.5 La routine POINTSTEALING

```
function POINTSTEALING(S_f, leaf0, tree0, leaf1, tree1)
    ▷ On considère d'abord la fusion des experts aux racines
    e_tree0 ← tree0.expert
    e_tree1 ← tree1.expert
    e_0 ← MERGEEXPERTS(S_f, e_tree0, e_tree1)
    if e_0 ≠ NONE then
        bestSolution ← createNode(newExpert, children={tree0, tree1})
        bestComplexity ← e_0.complexity
    else
        bestSolution ← NONE
        bestComplexity ← e_tree0.complexity + e_tree1.complexity
    end if

    ▷ On cherche ensuite à réduire la complexité en autorisant un arbre
    ▷ à voler des points (i.e. un nœud et ses fils) à l'autre
    for triplet ∈ {(leaf0, tree0, tree1), (leaf1, tree1, tree0)} do
        (node, parentTree, otherTree) ← triplet
        (e_parent, e_other) ← (parentTree.expert, otherTree.expert)

        ▷ On remonte la branche contenant node à la recherche
        ▷ d'une complexité minimale
        while node ≠ parentTree do
            e_0 ← MERGEEXPERTS(S_f, e_other, node.expert)
            if e_0 ≠ NONE then
                remainingNodes ← GETREMAININGNODES(parentTree, node)
                S_d ← ⋃_{n ∈ remainingNodes} n.expert.points
                (c_1, f_1) ← GETBESTFIT(S_f, S_d, S_f,min, e_parent.complexity)
                totalComplexity ← e_0.complexity + c_1
                if totalComplexity < bestComplexity then
                    newNode0 ← createNode(e_0, children={node, otherTree})
                    e_1 ← (S_d, c_1, f_1)
                    newNode1 ← createNode(e_1, children=orphanNodes)
                    bestSolution ← {newNode0, newNode1}
                    bestComplexity ← totalComplexity
                end if
            end if
            node ← parentTree.getParent(node)
        end while

    end for
    return bestSolution
end function
```

plusieurs nœuds de la branche sont éligibles pour le “vol”, la routine choisit celui qui mène à la plus faible complexité totale. Le Pseudocode 4.5 donne le détail de POINTSTEALING.

4.5.3 Mise à jour de la collection d’arbres

À l’issue de la phase d’initialisation, COCOTTE+ met à jour la collection d’arbres en appelant successivement POINTSTEALING pour chaque paire de feuilles (une unique fois par paire) en passant également en argument les arbres qui les contiennent. À chaque fois que la routine ne renvoie pas NONE, ces arbres sont remplacés dans la collection par ceux retournés par POINTSTEALING. Les paires de feuilles sont considérées en commençant par celles dont les points correspondant sont les plus proches pour une mesure de distance donnée (la distance en norme L_1 dans notre implémentation). Ce processus vient remplacer la phase de fusion gloutonne de COCOTTE, et permet de se passer de la phase d’élimination des artefacts (voir chapitre suivant). Une version détaillée de l’algorithme est donnée en annexe, en Pseudo-code A.3.

COCOTTE+ permet par construction d’éliminer des artefacts au cours de la phase de fusion, et donne par conséquent la possibilité à des structures locales d’émerger même lorsque les premières fusions d’experts mènent à des artefacts (ce qui peut avoir lieu par exemple si les points ne sont pas assez denses ou si la fonction de distance est peu adaptée). COCOTTE+ est donc bien une amélioration de COCOTTE qui réduit la dépendance de l’algorithme à la qualité des toutes premières fusions d’experts.

4.6 Conclusion

Dans ce chapitre, après avoir brièvement rappelé le problème du compromis entre nombre et versatilité des experts, nous nous sommes intéressés au principe du rasoir d’Ockham pour apporter une réponse à cette question : choisir le mélange d’experts de complexité minimale parmi ceux qui rendent compte des données d’entraînement.

Nous avons d’abord posé des définitions permettant d’établir formellement ce qu’est la complexité d’un mélange d’experts et ce que signifie le fait de rendre compte des données d’entraînement, puis nous nous sommes ensuite appuyés sur ces définitions pour introduire COCOTTE, l’algorithme que nous proposons pour rechercher ce mélange d’expert, et détailler les principales routines sur lesquelles il s’appuie. Nous avons également introduit COCOTTE+, une amélioration de COCOTTE permettant de limiter la dépendance à une des hypothèses sur lesquelles cet algorithme s’appuie.

Dans le prochain chapitre, nous appliquerons ces algorithmes à l’approximation de diverses fonctions et nous verrons si cet algorithme est à la hauteur

de nos attentes.

Chapitre 5

Expériences

Dans les chapitres précédents nous avons vu que notre problème d'apprentissage de compétences paramétrées pouvait être vu comme un problème de régression vers une fonction localement $\mathscr{C}^\infty$ (chapitre 2), puis avons passé en revue divers algorithmes de régression et déterminé qu'une approche de type mélange d'experts serait appropriée (chapitre 3).

Nous avons ensuite montré que pour une telle approche, la question du compromis entre le nombre d'experts et la versatilité de ceux-ci était critique, puis nous avons introduit COCOTTE, l'algorithme de mélange d'experts que nous avons développé pour répondre à cette question et qui se fonde sur la recherche du mélange d'experts de complexité minimale, ainsi qu'une version améliorée de celui-ci appelée COCOTTE+.

Dans ce chapitre, nous allons maintenant appliquer ces algorithmes à l'apprentissage de diverses fonctions ainsi qu'à l'apprentissage de compétences paramétrées. Dans toutes les expériences que nous allons voir, l'implémentation qui est utilisée est celle présentée à la fin du chapitre précédent : S_f est l'ensemble des familles $P_{k,d}$ (fonctions polynomiales de degré au plus k à $|d|$ variables, appliquées aux dimensions de x dont les indices sont dans d), chacune de ces familles est associée à une complexité $k \times |d| + 1$, la distance entre deux points de données est la distance euclidienne entre ceux-ci, la distance entre deux experts est la distance minimale entre les points de données qui leur sont rattachés, et les régions sont déterminées à l'aide d'une forêt d'arbres décisionnels.

5.1 Expérience : application à des cas simples

Avant d'appliquer COCOTTE à la recherche de compétences paramétrées directes, commençons par vérifier que cet algorithme de mélange d'experts donne les résultats attendus sur des cas simples, c'est-à-dire sur des données synthétiques ou les fonctions à approximer et les imprécisions du jeu d'entraînement sont connues.

5.1.1 Comportement lié à l'imprécision

Considérons tout d'abord une fonction sinus d'amplitude 1, légèrement bruitée au moyen d'un bruit uniforme d'amplitude 0.1, et intéressons-nous à l'influence de l'imprécision attribuée aux données d'entraînement sur le comportement de COCOTTE.

Si les imprécisions fournies sur ces données sont égales (ou très légèrement supérieures) au bruit, COCOTTE fournit une assez bonne approximation de notre fonction comme on peut le voir en Figure 5.1.

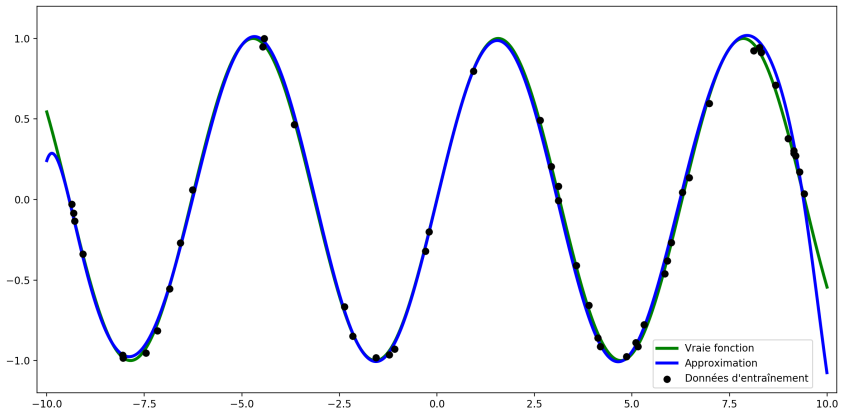


FIGURE 5.1 – Approximation par COCOTTE d'une fonction sinus d'amplitude 1 légèrement bruitée (bruit uniforme d'amplitude 0.1), quand les imprécisions fournies sur les 50 points d'entraînement valent $\varepsilon = 0.1$. L'approximation est composée d'un unique expert, appartenant à $P_{13,\{0\}}$.

Si on augmente l'imprécision, les variations de la fonction sinus sont progressivement assimilées à du bruit, et la complexité de l'approximation diminue jusqu'à devenir une fonction constante, comme on peut le voir en Figure 5.2.

Nous avons donc vu sur cet exemple que comme on pouvait s'y attendre, la capacité à rendre compte du jeu de données d'entraînement est fondamentale pour COCOTTE et, celle-ci étant intimement liée à la précision avec laquelle nous connaissons les points d'entraînement, des imprécisions plus élevées mènent à des approximations plus simples. En cela, les imprécisions sur le jeu d'entraînement jouent en pratique un rôle similaire aux hyper-paramètres réglant la complexité dont nous avions parlé précédemment. Ces imprécisions ne sont cependant pas des hyper-paramètres : elles ont ceci de différent qu'elles correspondent à quelques chose de concret qui peut être mesuré en amont de l'apprentissage (précision sur les flottants, précision des appareils de mesures, *etc*).

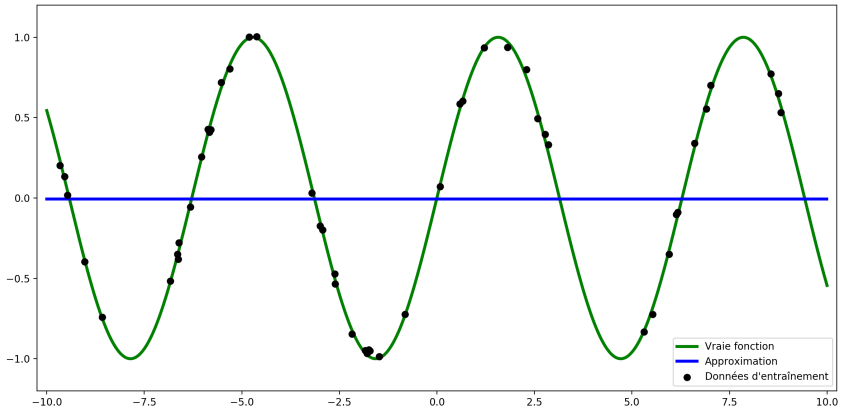


FIGURE 5.2 – Approximation par COCOTTE d'une fonction sinus d'amplitude 1 légèrement bruitée (bruit uniforme d'amplitude 0.1), quand les imprécisions fournies sur les 50 points d'entraînement valent $\varepsilon = 1.1$. L'approximation est composée d'un unique expert, appartenant à $P_{0,\emptyset}$.

5.1.2 Comportement lié aux ruptures de régularité

Intéressons-nous maintenant aux ruptures de régularité : COCOTTE est-il bien capable d'approximer une fonction qui n'est que localement $\mathscr{C}^\infty$ par plusieurs experts localement $\mathscr{C}^\infty$?

On peut dans un premier temps s'intéresser au cas d'une fonction discontinue, comme par exemple la fonction échelon dont nous avons parlé précédemment, qui vaut 1 sur $\mathbb{R}^+$ et 0 ailleurs. La Figure 5.3 montre le résultat de l'exécution de COCOTTE sur cette fonction : deux experts on été appris, l'un approximant la fonction créneau sur $\mathbb{R}^+$ par une constante proche de 1, et l'autre approximant la fonction sur $\mathbb{R}^-$ par une fonction constante proche de 0. L'approximation est très proche de la vraie fonction, bien que l'ajout de nouveaux points d'entraînement puisse permettre d'améliorer les régions.

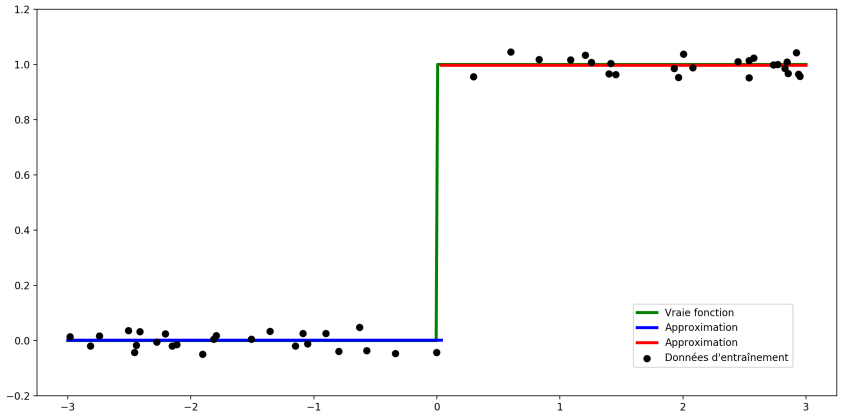


FIGURE 5.3 – Approximation par COCOTTE de la fonction échelon, valant 1 sur $\mathbb{R}^+$ et 0 ailleurs, légèrement bruitée (bruit uniforme d'amplitude 0.1). Les imprécisions fournies sur les 50 points d'entraînement valent $\varepsilon = 0.1$. L'approximation est composée de deux experts, en bleu et en rouge respectivement, appartenant tous les deux à $P_{0,\emptyset}$.

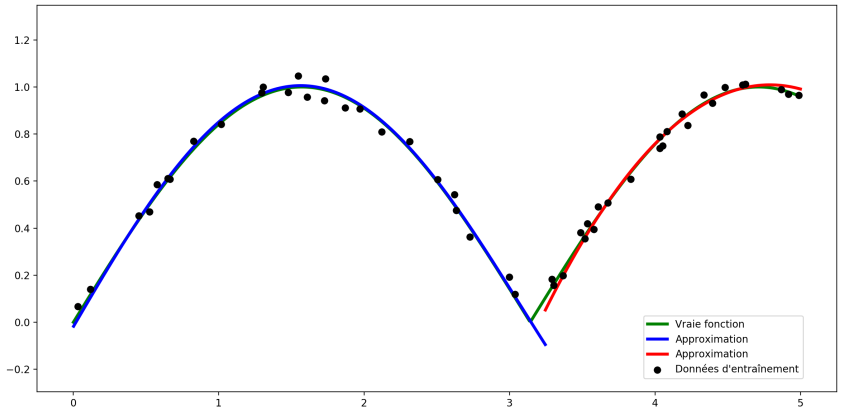


FIGURE 5.4 – Approximation par COCOTTE fonction $x \mapsto |sin(x)|$, légèrement bruitée (bruit uniforme d'amplitude 0.1). Les imprécisions fournies sur les 50 points d'entraînement valent $\varepsilon = 0.1$. L'approximation est composée de deux experts, l'un appartenant à $P_{4,\{0\}}$ (en bleu). et l'autre à $P_{2,\{0\}}$ (en rouge).

Dans un second temps, on peut s'intéresser au cas de fonctions qui sont continues mais comportent des ruptures de régularité, comme par exemple la fonction $x \mapsto |sin(x)|$. La Figure 5.4 montre le résultat de l'exécution de COCOTTE sur des données produites par cette fonction sur $[0, 5]$: deux experts on été appris, chacun approximant la fonction de part et d'autre de la rupture de régularité en π. Là encore, l'approximation est très proche de la vraie fonction, bien que l'ajout de nouveaux points d'entraînement puisse permettre d'améliorer les régions.

Nous avons donc vu sur ces exemples que COCOTTE est capable d'approximer des fonctions qui sont $\mathscr{C}^\infty$ localement mais pas globalement par des experts $\mathscr{C}^\infty$ qui en fournissent de bonnes approximations locales là où elle est régulière, les ruptures de régularité correspondant à des frontières entre les différents experts.

5.1.3 Et COCOTTE+ ?

L'exécution de COCOTTE+ sur les mêmes données que COCOTTE révèle que cet algorithme mène à des mélanges d'experts similaires à ceux de son prédécesseur. Il s'agit en effet de combinaisons d'experts de même nombre et de même complexités, et on peut constater visuellement sur les Figures 5.5, 5.6, 5.7 et 5.8 que les mélanges d'experts ainsi trouvés sont sensiblement identiques à ceux renvoyés par COCOTTE (la principale différence visuelle concernant l'emplacement des frontières entre les différents experts, qui vient en réalité non pas de la différence entre les algorithmes mais du caractère stochastique de l'algorithme de classification utilisé).

Cette similarité entre les mélanges d'experts renvoyés par COCOTTE et COCOTTE+ est observable bien que COCOTTE+ ne comprenne pas de phase d'élimination des artefacts. C'est en effet ici le "vol de points" qui joue ce rôle, ce qui est mis en évidence sur la Figure 5.9, qui représente les mélanges d'experts renvoyés sur les mêmes données par COCOTTE et COCOTTE+, en sautant pour COCOTTE la phase d'élimination des artefacts.

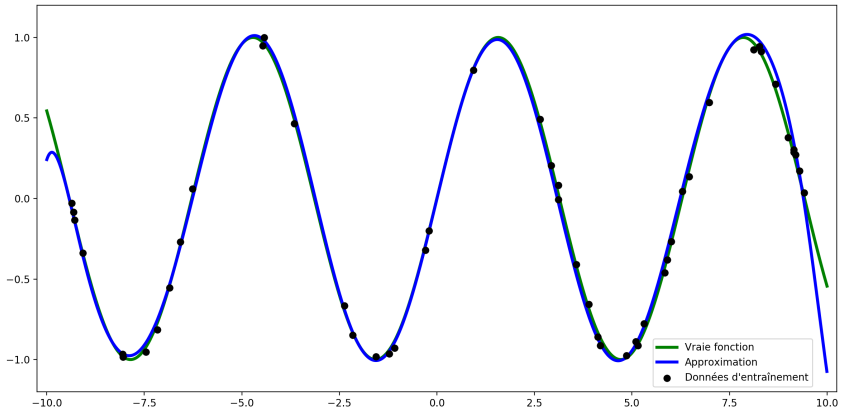


FIGURE 5.5 – Approximation par COCOTTE+ d'une fonction sinus d'amplitude 1 légèrement bruitée (bruit uniforme d'amplitude 0.1), quand les imprécisions fournies sur les 50 points d'entraînement valent $\varepsilon = 0.1$. L'approximation est composée d'un unique expert, appartenant à $P_{13,\{0\}}$.

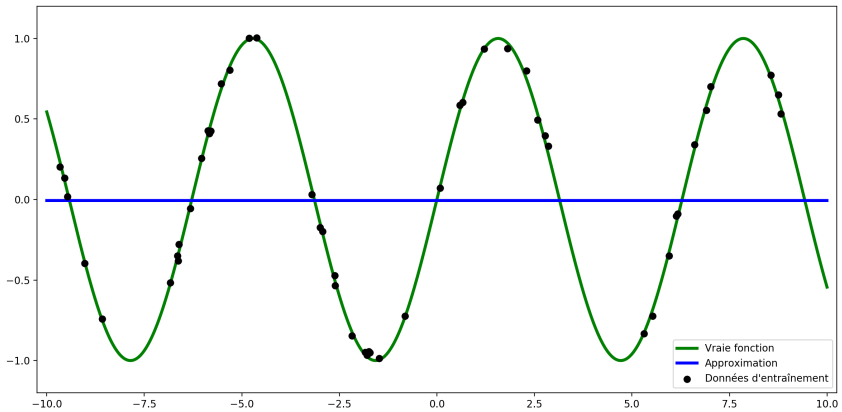


FIGURE 5.6 – Approximation par COCOTTE+ d'une fonction sinus d'amplitude 1 légèrement bruitée (bruit uniforme d'amplitude 0.1), quand les imprécisions fournies sur les 50 points d'entraînement valent $\varepsilon = 1.1$. L'approximation est composée d'un unique expert, appartenant à $P_{0,\emptyset}$.

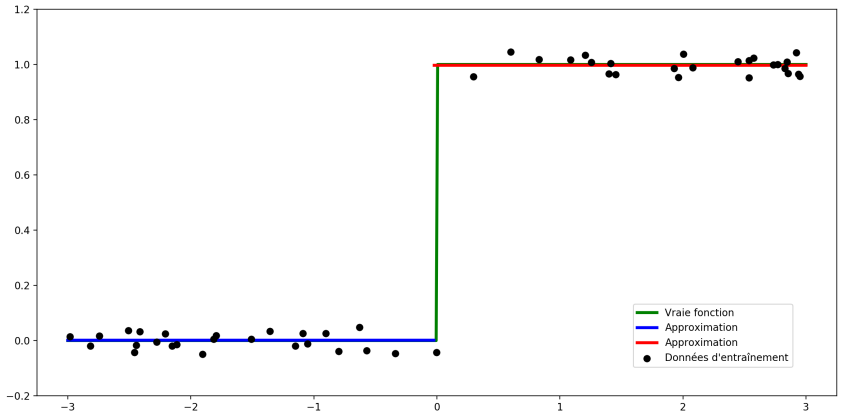


FIGURE 5.7 – Approximation par COCOTTE+ de la fonction échelon, valant 1 sur $\mathbb{R}^+$ et 0 ailleurs, légèrement bruitée (bruit uniforme d'amplitude 0.1). Les imprécisions fournies sur les 50 points d'entraînement valent $\varepsilon = 0.1$. L'approximation est composée de deux experts, en bleu et en rouge respectivement, appartenant tous les deux à $P_{0,\emptyset}$.

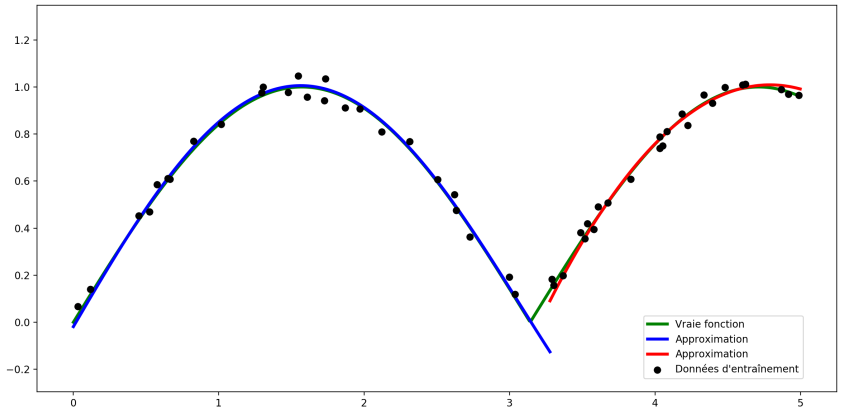
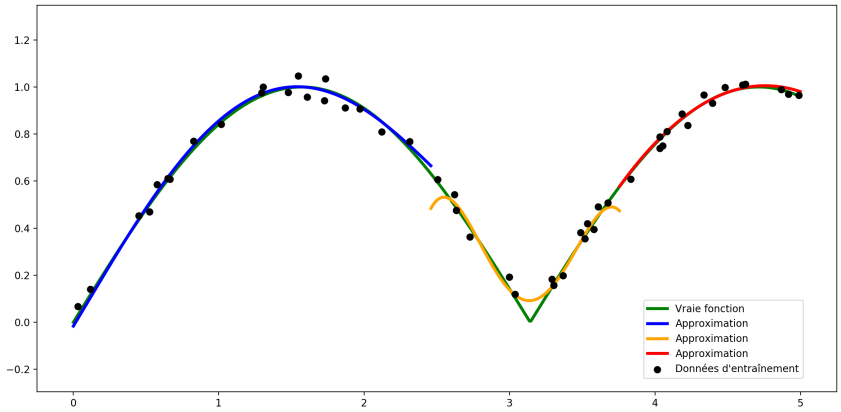
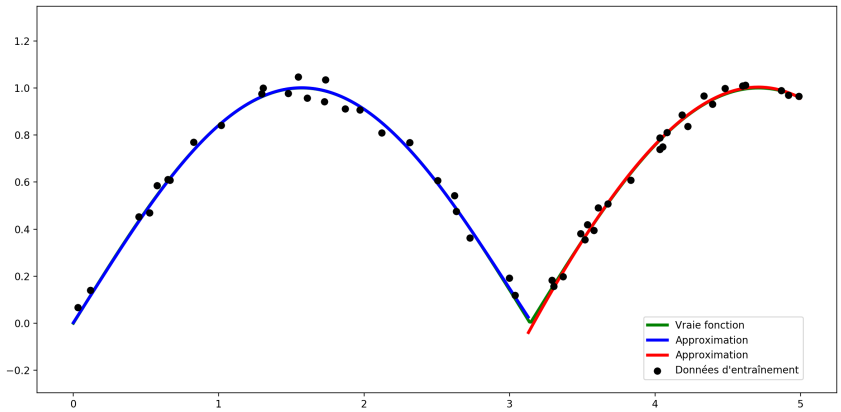


FIGURE 5.8 – Approximation par COCOTTE+ fonction $x \mapsto |sin(x)|$, légèrement bruitée (bruit uniforme d'amplitude 0.1). Les imprécisions fournies sur les 50 points d'entraînement valent $\varepsilon = 0.1$. L'approximation est composée de deux experts, l'un appartenant à $P_{4,\{0\}}$ (en bleu). et l'autre à $P_{2,\{0\}}$ (en rouge).



(a)



(b)

FIGURE 5.9 – Approximations de la fonction $x \mapsto |sin(x)|$, légèrement bruitée (bruit uniforme d'amplitude 0.1), en fournissant des imprécisions $\varepsilon = 0.1$ sur les 100 points d'entraînement. Ces approximations correspondent aux mélanges d'experts renvoyés (a) par COCOTTE en sautant la phase d'élimination des artefacts, et (b) par COCOTTE+

5.2 Expérience : simulation d'interaction d'un bras robotique avec un cube

Maintenant que nous avons vérifié que COCOTTE donne les résultats attendus sur des fonctions simples connues, il est temps de l'appliquer au problème pour lequel nous l'avons développé : la recherche de compétences paramétrées directes.

5.2.1 Environnement de simulation

Nous utilisons l'environnement de simulation de Ecarlat *et al.* [2015], qui met en jeu un bras robotique fixé à une table sur laquelle se trouve un objet cubique (voir Figure 5.10). La configuration du robot est entièrement déterminée par les angles des 6 articulations et l'état d'ouverture de la pince, représentés collectivement par un vecteur $q \in [0, 1]^7$. La position du cube sera quant à elle représentée par un vecteur $p = (x, y, z) \in \mathbb{R}^3$, les coordonnées x, y et z correspondant respectivement à l'axe avant-arrière du robot (*i.e.* axe de la longueur de la table), à l'axe droite-gauche (*i.e.* axe de la largeur de la table), et à l'axe vertical (*i.e.* axe de l'épaisseur de la table).

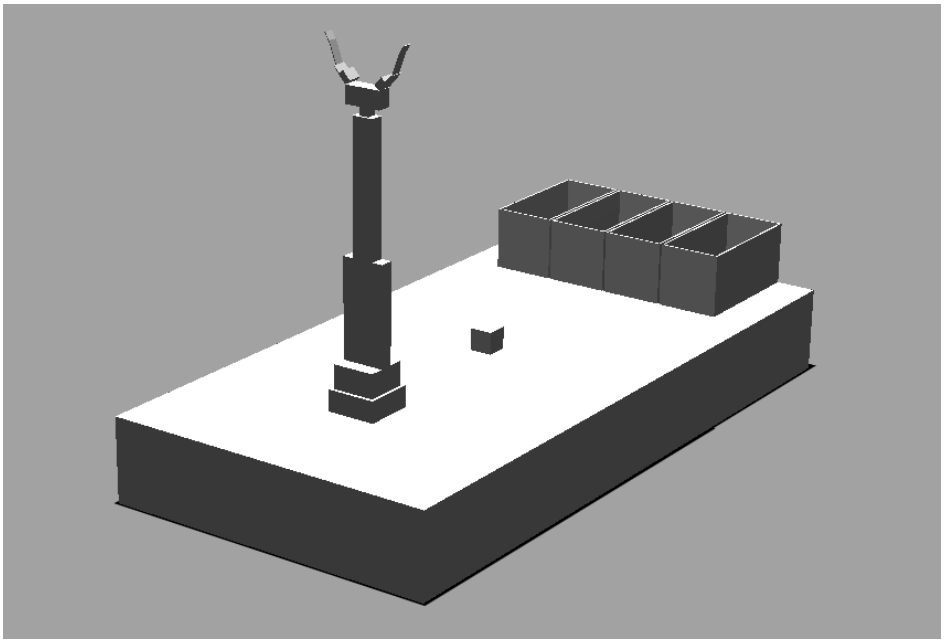


FIGURE 5.10 – L'environnement de simulation. Le robot est ici dans la configuration $(0, 0, 0, 0, 0, 0, 0)$ et le cube à la position $(0, 0, 0)$.

5.2.2 Protocole expérimental

Domaines, politiques, effets

Le bras robotique est initialement placé dans une configuration q_0 variable, et le cube en une position fixe $(0, 0, 0)$ située devant le robot (*cf.* Figure 5.10). On choisit alors une nouvelle configuration q_1 que l'on fait prendre au robot, puis on observe la position finale p_{fin} du cube (à laquelle on ajoute un bruit uniforme d'amplitude 1cm et centré sur 0).

Dans le formalisme que nous avons présenté aux chapitre 1, nous avons donc ici un domaine paramétré par q_0, une politique paramétrée par q_1, et un effet paramétré par p_{fin}. La précision que nous avons sur les valeurs de p_{fin} sont prises comme la somme de l'amplitude du bruit que nous avons ajouté et de celle du bruit de simulation (cette dernière est évaluée empiriquement et est composée d'un terme constant et d'un terme proportionnel au déplacement du cube).

Génération des données

Afin de permettre une meilleure visualisation du comportement de COCOTTE, nous nous limitons à des interactions mettant en jeu un nombre restreint de dimensions. Pour cela, nous faisons exécuter deux types de comportements A et B au robot.

Le comportement A consiste à tirer aléatoirement deux réels $t_0 \in [-1, 0]$ et $t_1 \in [t_0, 1.5]$, puis à se placer dans le domaine défini par le vecteur de paramètres $q_0 = q_0^A + t_0 \cdot \overrightarrow{q_0^A q_1^A}$ et à exécuter la politique définie par le vecteur de paramètres $q_1 = q_0^A + t_1 \cdot \overrightarrow{q_0^A q_1^A}$, où q_O^A et q_1^A sont deux configurations fixes du robot dans lesquelles la pince est fermée et située respectivement à gauche et à droite du cube (axe y).

Le comportement B est identique au comportement A, à ceci près qu'il s'appuie sur deux autres configurations fixes du robot q_0^B et q_1^B, dans lesquelles la pince est également fermée mais cette fois située respectivement à droite et à gauche du cube. Le comportement A mène donc à pousser le cube vers la droite et le comportement B à le pousser vers la gauche.

On exécute aléatoirement les comportements A et B pour générer d'une part 300 points de données d'interaction qui constitueront le jeu de données d'entraînement que nous fournirons à COCOTTE, et d'autre part 10 000 autres points de données qui constitueront un jeu de test qui nous permettra ensuite d'évaluer la qualité de ce qui a été appris.

5.2.3 Résultats

Afin de mesurer quantitativement la qualité des experts appris par COCOTTE, on exécute l'algorithme successivement sur des sous-ensemble du jeu

d'entraînement de tailles croissantes (les 10 premiers points du jeu d'entraînement, puis les 20 premiers points, puis les 30 premiers points, *etc*). On recommence ensuite après avoir réalisé une permutation aléatoire des points, et ce jusqu'à obtenir les résultats de l'apprentissage pour 30 permutations distinctes (ce que nous avons jugé suffisant pour calculer moyenne et écart-type).

La Figure 5.11 montre l'évolution de la complexité totale (somme des complexités des fonctions prédisant chacune des coordonnées de p_{fin}) à mesure que la taille du jeu d'entraînement augmente. Lorsque le nombre de points est très faible, la complexité l'est également : le nombre de points limite la complexité maximum et ne permet pas d'apprendre une fonction de complexité satisfaisante pour refléter la structure réelle des données, et on est dans une situation de sous-entraînement. Dit autrement, il existe des fonctions très simples permettant de rendre compte des points d'entraînement, et plus de données sont nécessaires pour pouvoir les rejeter au profit de fonctions plus complexes. Lorsque la taille du jeu d'entraînement augmente, la complexité totale commence donc d'abord par augmenter (on sort du sous-entraînement), avant de redescendre et de se stabiliser.

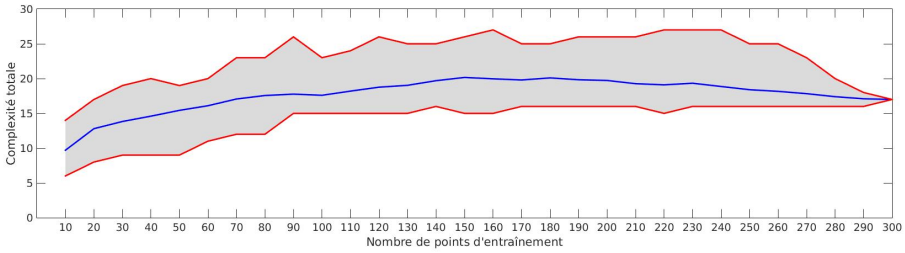


FIGURE 5.11 – Évolution de la complexité totale du mélange d'experts renvoyé par COCOTTE à mesure que l'on ajoute les points du jeu d'entraînement. Sont représentés la moyenne et les valeurs extrémales sur 30 permutations aléatoires du jeu d'entraînement (voir texte pour plus d'informations). Celles-ci sont égales pour 300 points car tout le jeu d'entraînement est donné à COCOTTE quelle que soit la permutation.

On peut cependant s'étonner du fait que la complexité augmente puis redescende. En effet, si COCOTTE apprend le mélange d'experts le plus simple rendant compte des données d'entraînement, alors la complexité de ce mélange ne devrait qu'augmenter ou rester constante lorsque l'on ajoute des points d'entraînement. Nous pouvons cependant proposer une explication assez simple à cela. Comme nous l'avons vu en section 4.3.4, l'étape de fusion gloutonne repose sur l'idée que les points sont assez denses pour que les fusions d'experts se fassent d'abord localement puis à plus grande échelle, et permettent ainsi à des structures locales d'émerger et d'empêcher de fusionner des experts à tort. Si les points d'entraînement ne sont initialement pas assez denses, ces structures locales ne peuvent pas émerger et les artefacts (experts résultant de

mauvaises fusions) sont difficiles à différencier des autres experts. Il est donc normal que la complexité totale soit trop élevée lorsque les points de données d'entraînement sont encore peu nombreux et épars, et qu'elle diminue lorsque de nouveaux points sont ajoutés et permettent de gagner en densité.

La Figure 5.12 montre quant à elle l'évolution du taux d'erreur de prédiction sur les diverses coordonnées (pourcentage de points du jeu de test prédits avec des erreurs supérieures aux précisions sur ces points) à mesure que l'on ajoute des points de données d'entraînement. On constate ainsi que le taux d'erreur diminue très vite lorsque l'on ajoute de nouveaux points, et converge vers un taux d'erreur très bas avoisinant les 2 ou 3% selon la coordonnée. Le mélange d'experts appris par COCOTTE semble donc avoir de bonnes capacités de généralisation.

Lorsque la totalité des 300 points du jeu de données d'entraînement est donné à COCOTTE, l'algorithme renvoie un mélange de 4 experts. Pour visualiser ce à quoi correspondent ces experts, nous allons nous intéresser spécifiquement à la coordonnée y de p_{fin}, qui correspond à l'axe allant de la droite (valeurs faibles de y) à la gauche (valeurs élevées de y) du robot. En effet, comme nous l'avons vu précédemment, les comportements A et B que nous avons fait adopter au robot font principalement bouger la pince (et donc le cube) le long de l'axe gauche-droite du robot ; les variations de la coordonnée x (axe avant-arrière) sont donc anecdotiques, et la coordonnée z ne prend que deux valeurs selon que le cube est resté sur la table (ce qui correspond à 3 des 4 experts) ou qu'il en est tombé (dernier expert). La Figure 5.13 montre les prédictions de y fournies par les 4 experts appris par COCOTTE pour les points du jeu de test appartenant à leur région.

Au-delà des performances quantitatives (complexité, taux d'erreurs) vues plus haut, c'est d'un point de vue qualitatif que les choses sont les plus intéressantes. D'une part, on peut constater visuellement sur la Figure 5.13 que le mélange d'experts appris par COCOTTE est le mélange d'experts que l'on cherche : il est proche des données, les zones où la fonction sous-jacente est régulière ne sont pas divisées entre plusieurs experts, et les ruptures de régularité correspondent bien à des frontières entre modèles.

D'autre part, les experts ainsi appris semblent bel et bien correspondre aux compétences paramétrées directes que nous cherchons à apprendre : on peut ainsi les interpréter sémantiquement comme étant les compétences paramétrées directes $\lceil$*ne pas toucher le cube*$\rfloor$ (en rouge), $\lceil$*pousser le cube vers la gauche de la table*$\rfloor$ (en vert), $\lceil$*pousser le cube vers la droite de la table*$\rfloor$ (en orange), et $\lceil$*pousser le cube vers la droite jusqu'à le faire tomber*$\rfloor$ (en bleu).

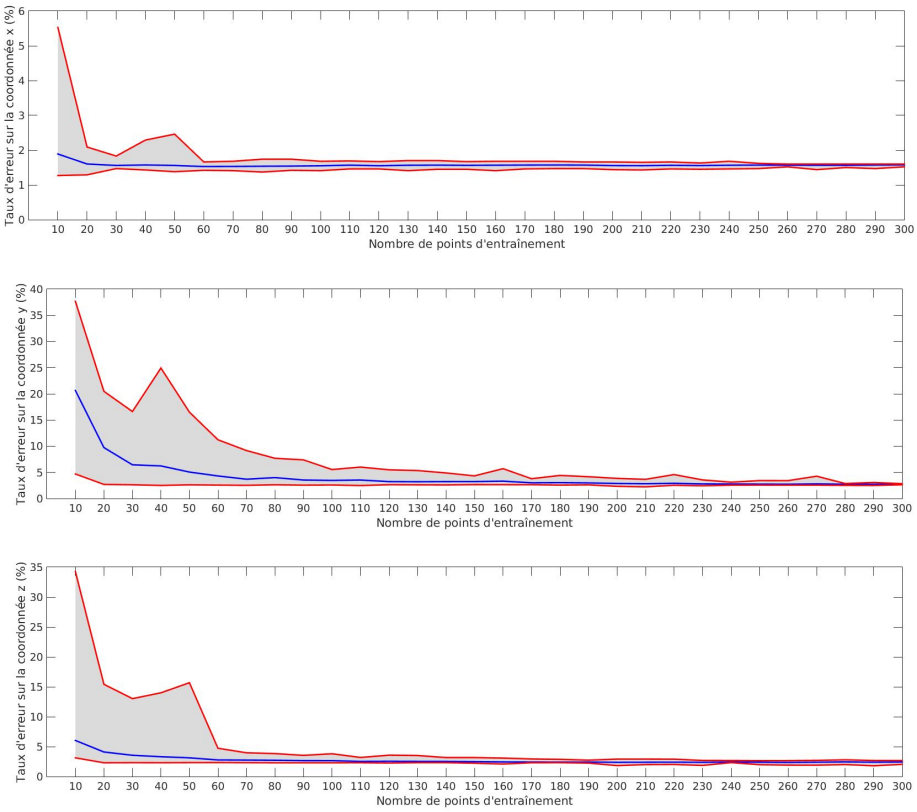


FIGURE 5.12 – Évolution des taux d'erreur de prédiction du mélange d'expert renvoyé par COCOTTE sur les diverses coordonnée de p_{fin} (pourcentage de points du jeu de test prédits avec des erreurs supérieures aux précisions sur ces points) à mesure que l'on ajoute les points du jeu d'entraînement. Sont représentés la moyenne et les valeurs extrémales sur 30 permutations aléatoires du jeu d'entraînement (voir texte pour plus d'informations). Celles-ci sont égales pour 300 points car tout le jeu d'entraînement est donné à COCOTTE quelle que soit la permutation.

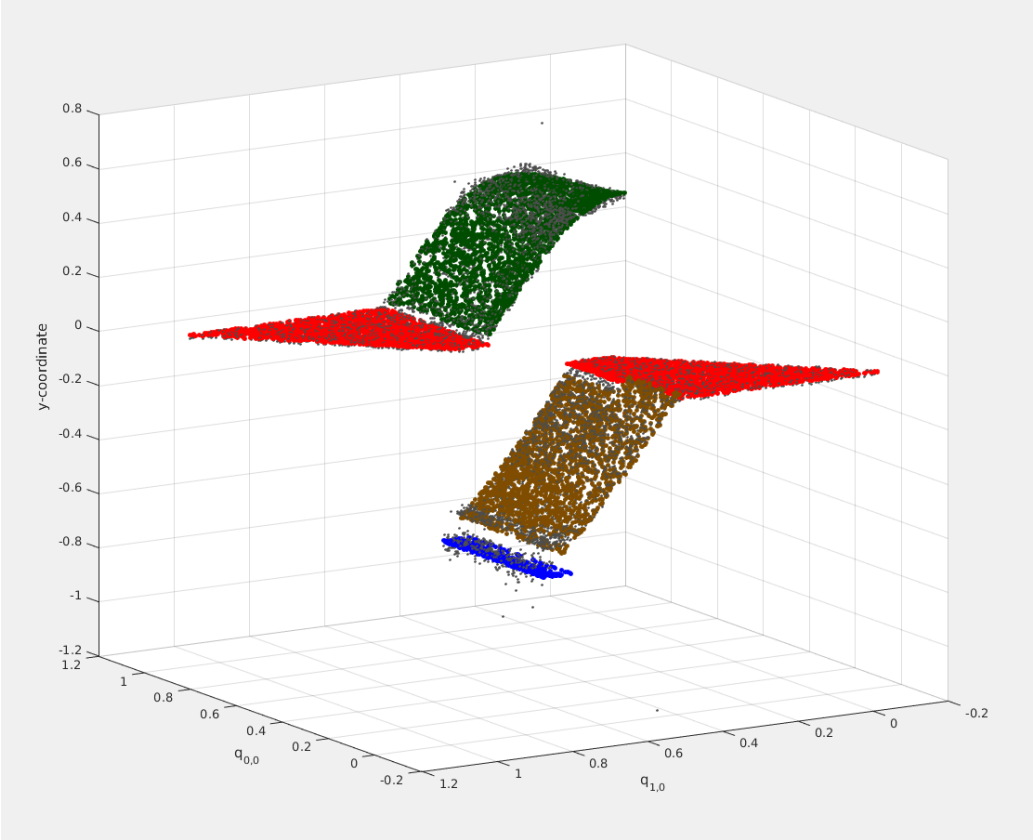


FIGURE 5.13 – À l'issue de l'apprentissage, COCOTTE rend compte du jeu d'entraînement à l'aide de 4 experts. Sont représentés ici d'une part les points du jeu de test (en gris), et d'autre part les prédictions des différents experts (en couleur). Pour permettre la visualisation, on s'intéresse à la coordonnée y de p_{fin} en fonction de $q_{0,0}$ (la première coordonnée de q_0) et $q_{1,0}$ (la première coordonnée de q_1). Les points verts sont prédits par un expert associé sur y à une fonction de $P_{3,\{q_{1,0}\}}$ (complexité 4), les points oranges par un expert associé sur y à une fonction de $P_{2,\{q_{1,0}\}}$ (complexité 3), les points rouges par un expert associé sur y à une fonction de $P_{0,\emptyset}$ (complexité 1), et les points bleus par un expert associé sur y à une autre fonction de $P_{0,\emptyset}$ (complexité 1 également). Voir le texte pour plus d'information.

On peut par ailleurs constater que les compétences ⌈*ne pas toucher le cube*⌋ et ⌈*pousser le cube vers la droite jusqu'à le faire tomber*⌋ correspondent à des fonctions constantes et ne s'appuient sur q_0 et q_1 que pour définir les régions qu'elles occupent. Les compétences⌈*pousser le cube vers la gauche de la table*⌋ et ⌈*pousser le cube vers la droite de la table*⌋ correspondent quant à elles à des fonctions qui ne dépendent que de $q_{1,1}$ (la première coordonnée de q_1) ; on retrouve bien là le fait que les comportements que nous avons fait exécuter au robot sont entièrement déterminés par (t_0, t_1), qui peuvent être déduits de $q_{0,0}$ (la première coordonnée de q_0) et $q_{1,1}$, et que la position finale du cube est indépendante de t_0. Le fait que la complexité des familles de fonctions soit lié aux nombres de dimensions mises en jeu a donc bien permis à COCOTTE de sélectionner les dimensions pertinentes pour chacune des compétences paramétrées directes qui ont été apprises.

5.2.4 Et COCOTTE+ ?

Ici aussi, malgré l'absence de phase d'élimination des artefacts, l'exécution de COCOTTE+ sur les mêmes données que son prédécesseur mène à un mélange d'experts sensiblement identique à celui trouvé par COCOTTE.

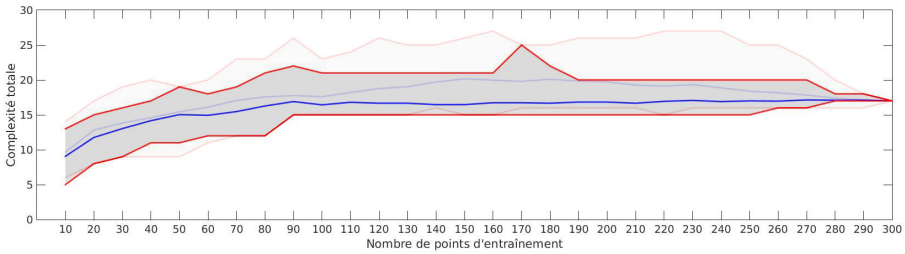


FIGURE 5.14 – Évolution de la complexité totale du mélange d'experts renvoyé par COCOTTE+ à mesure que l'on ajoute les points du jeu d'entraînement. Sont représentés la moyenne et les valeurs extrémales sur 30 permutations aléatoires du jeu d'entraînement (voir texte pour plus d'informations). Celles-ci sont égales pour 300 points car tout le jeu d'entraînement est donné à COCOTTE+ quelle que soit la permutation. Pour faciliter la comparaison, l'évolution de la complexité totale pour COCOTTE (Figure 5.11) est superposée par transparence.

Lorsque l'on exécute l'algorithme successivement sur des sous-ensemble du jeu d'entraînement de tailles croissantes comme précédemment, on constate que la complexité totale du mélange d'experts renvoyé évolue différemment pour les deux algorithmes. Dans le cas de COCOTTE (voir Figure 5.11), lorsque la taille du jeu d'entraînement augmente, la complexité totale augmente d'abord lorsque l'on sort du sous-entraînement, puis redescend lorsque

les structures locales permettre de limiter la formation d'artefacts. Dans le cas de COCOTTE+ (voir Figure 5.14), la complexité totale augmente également lorsque l'on sort du sous-entraînement, mais converge directement vers la complexité finale. On retrouve donc bien que COCOTTE+ est moins dépendant que COCOTTE à la qualité des toutes premières fusions d'experts, et qu'il permet de limiter la formation d'artefacts même lorsque les points de données sont peu denses. Ceci se ressent également sur les taux d'erreur de prédiction (voir Figure 5.15), qui convergent plus rapidement vers leurs valeurs finales avec COCOTTE+ qu'avec COCOTTE.

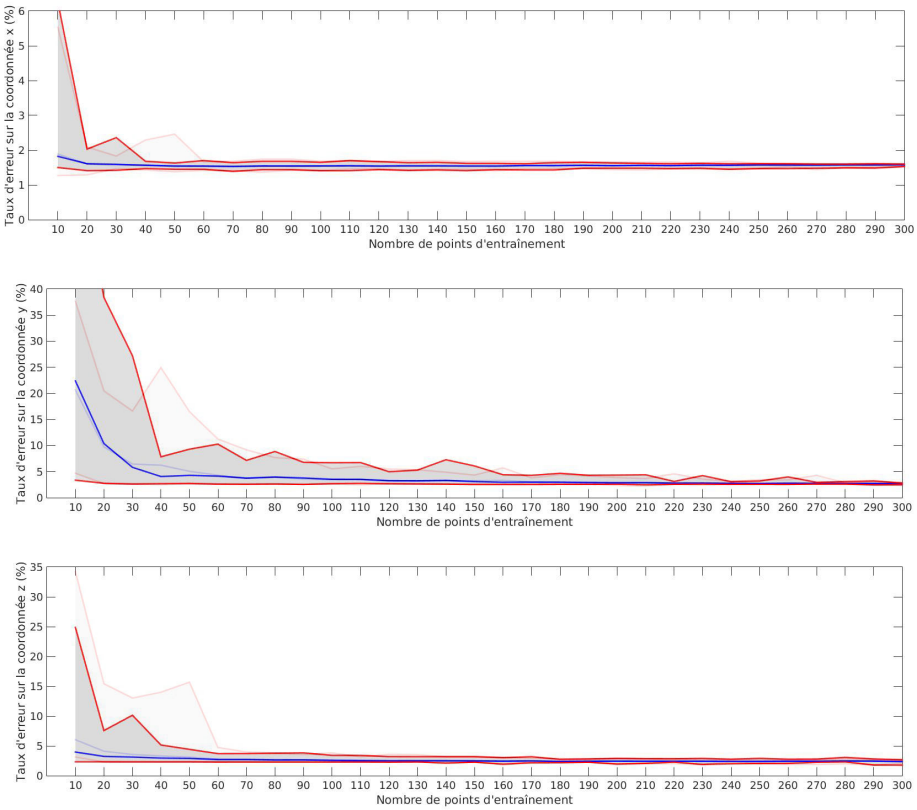


FIGURE 5.15 – Évolution des taux d'erreur de prédiction du mélange d'expert renvoyé par COCOTTE+ sur les diverses coordonnée de p_{fin} (pourcentage de points du jeu de test prédits avec des erreurs supérieures aux précisions sur ces points) à mesure que l'on ajoute les points du jeu d'entraînement. Sont représentés la moyenne et les valeurs extrémales sur 30 permutations aléatoires du jeu d'entraînement (voir texte pour plus d'informations). Celles-ci sont égales pour 300 points car tout le jeu d'entraînement est donné à COCOTTE+ quelle que soit la permutation. Pour faciliter la comparaison, l'évolution des taux d'erreurs pour COCOTTE (Figure 5.12) sont superposées par transparence.

5.3 Conclusion

Dans ce chapitre, nous avons d'abord appliqué COCOTTE et COCOTTE+ à des fonctions simples connues afin de vérifier que ces algorithmes ont le comportement attendu et que les mélanges d'experts qu'ils renvoient fournissent de bonnes approximations de fonctions localement régulières.

Nous avons ensuite appliqué ces algorithmes à des données d'interaction entre un robot et un objet, les mélanges d'experts ainsi appris permettant de structurer les contingences sensorimotrices (soit ici l'influence de la politique exécutée sur la position finale de l'objet observée par le robot) en compétences paramétrées directes.

À l'issue de l'apprentissage, les mélanges d'experts ainsi appris semblent refléter la structure des données d'interaction et fournissent des compétences distinctes auxquelles il est possible d'attribuer une réelle interprétation sémantique. De plus, la comparaison de COCOTTE et COCOTTE+ confirme l'avantage du second sur le premier, COCOTTE+ permettant ainsi de converger vers les mélanges d'experts souhaités de façon plus robuste, et fournissant des mélanges d'experts de meilleure qualité lorsque moins de points de données d'entraînement sont disponibles.

Conclusion et perspectives

La découverte de contingences sensorimotrices et leur structuration en compétences sont des enjeux importants en robotique. En particulier, on souhaite pouvoir apprendre ainsi des compétences qui soient à la fois riches sémantiquement et aussi générales que possible.

Cette thèse s'est intéressée à la question du regroupement de compétences en compétences plus générales, et a entrepris de déterminer quand il est ou non pertinent de réaliser de tels regroupement ainsi que de proposer des algorithmes s'appuyant sur ces critères pour les effectuer.

Ce chapitre résume les contributions de cette thèse et présente quelques pistes pour de futures recherches.

Résumé

Dans ce document, nous avons dans un premier chapitre présenté le formalisme des politiques et des options et donné une définition de ce qu'est une tâche. Nous avons alors discuté des diverses façons dont le terme de "compétence" est utilisé dans la littérature et posé la définition que nous utiliserions par la suite. Suite à cela, nous avons vu comment ces définitions pouvaient être étendues à des politiques paramétrées, tâches paramétrées et compétences paramétrées.

Nous avons ensuite inscrit l'apprentissage de compétences paramétrées dans le cadre plus large du transfert entre divers apprentissages, dont nous avons présenté les différentes problématiques ainsi que modalités. Ceci nous a permis de reformuler l'apprentissage de compétences paramétrées comme une technique permettant un transfert de paramètres dans le cadre d'une problématique d'apprentissage multi-tâches et multi-domaines de politiques. Nous avons alors utilisé ce formalisme pour étendre les définitions de "compétence" et "compétence paramétrée" à un cadre un peu plus général.

Dans le chapitre suivant, nous avons adapté ces définitions au cas d'un robot qui ne reçoit pas de tâches d'un utilisateur extérieur mais se fixe ses propres buts durant la phase d'apprentissage. Nous avons ensuite connecté la notion de compétences paramétrées au formalisme des modèles inverses, ce qui nous a poussé à les qualifier par la suite de "compétences paramétrées inverses".

En nous fondant sur la dualité entre modèles inverses et modèles directs, nous avons alors élaboré une nouvelle forme de compétences paramétrées connectée cette fois-ci aux modèles directs : les compétences paramétrées directes.

Ce chapitre fut également l'occasion de montrer les limites des approches inverses pour le problème qui nous intéresse — le regroupement de compétences en compétences plus générales — et d'illustrer au moyen d'une expérience de pensée ce que pourrait apporter une approche directe pour le traitement de celui-ci. Au vu de ces considérations, nous avons alors proposé une approche directe qui consiste, dans une cadre plus général que celui de l'expérience de pensée, à ramener ce regroupement des compétences à un problème de régression vers une fonction localement $\mathscr{C}^\infty$ mais non nécessairement globalement (détermination des plus grands sous-espaces de l'espace des compétences sur lesquels des modèles directs $\mathscr{C}^\infty$ locaux peuvent être trouvés, et estimation de ces modèles).

Afin de traiter le problème auquel nous nous sommes ramenés, nous avons consacré notre troisième chapitre aux algorithmes de régression, à la recherche d'un algorithme qui y soit adapté. Nous nous sommes dans un premier temps intéressés aux algorithmes de régression linéaire, qui bien qu'étant insuffisants pour traiter notre problème sont à la base d'autres algorithmes plus élaborés et nous ont également permis d'aborder certains enjeux des problèmes de régression. Ces algorithmes, même étendus à des cas non-linéaires, n'étant pas assez expressifs pour approximer de façon satisfaisante des fonctions qui sont localement $\mathscr{C}^\infty$ mais pas globalement, nous avons ensuite considéré les algorithmes de régression non paramétrique. Nous avons alors vu que ces derniers, bien que particulièrement adaptés à l'approximation de fonctions qui ne sont pas globalement $\mathscr{C}^\infty$, sont en revanche trop locaux pour tirer parti efficacement des données et ne permettent pas non plus de réellement partitionner le domaine de définition en sous-espaces distincts. Nous nous sommes alors tournés vers des méthodes qui abordent différemment la localité : les algorithmes de mélange d'experts. Si cette classe d'algorithmes nous a semblé adaptée à notre problème, nous avons toutefois constaté que les algorithmes existants s'intéressaient peu à la question du compromis entre le nombre des experts et la versatilités de ceux-ci, pourtant cruciale à notre cas d'application.

Dans un quatrième chapitre, après avoir brièvement rappelé le problème du compromis entre nombre et versatilité des experts, nous nous sommes intéressés au principe du rasoir d'Ockham pour apporter une réponse à cette question : choisir le mélange d'experts de complexité minimale parmi ceux qui rendent compte des données d'entraînement. Nous avons alors posé des définitions permettant d'établir formellement ce qu'est la complexité d'un mélange d'experts et ce que signifie le fait de rendre compte des données d'entraînement, puis nous nous sommes ensuite appuyés sur ces définitions pour introduire COCOTTE, l'algorithme que nous proposons pour rechercher ce mélange d'expert, et détailler les principales routines sur lesquelles il s'appuie. Nous avons également

introduit COCOTTE+, une amélioration de COCOTTE permettant de limiter la dépendance à une des hypothèses sur lesquelles cet algorithme s'appuie.

Nous avons enfin appliqué ces algorithmes dans un cinquième et dernier chapitre, d'une part à des fonctions simples connues — afin de vérifier que ces algorithmes ont le comportement attendu et que les mélanges d'experts qu'ils renvoient fournissent de bonnes approximations de fonctions localement régulières — et d'autre part à des données d'interaction entre un robot et un objet. Sur ces données d'interaction, nous avons constaté que le mélange d'experts trouvé par COCOTTE et COCOTTE+ semble refléter la structure des données d'interaction, et fournit des compétences distinctes auxquelles il est possible d'attribuer une réelle interprétation sémantique. Par ailleurs, l'application de COCOTTE et COCOTTE+ à ces données permet de confirmer que le second est bien une amélioration du premier, celui-ci convergeant vers les mélanges d'experts souhaités de façon plus robuste, et fournissant des mélanges d'experts intermédiaires de meilleure qualité lorsque le nombre de points de données d'entraînement est insuffisant.

Pistes pour de futures recherches

Extensions directes des travaux présentés

De futurs travaux pourraient s'intéresser à l'apprentissage de compétences dans des scénarios plus complexes que ceux présentés ici. De tels scénarios peuvent en effet s'avérer difficiles à traiter pour nos algorithmes, car la complexité des compétences recherchées peut se manifester par des structures locales également complexes. Celles-ci nécessitent alors un nombre important de fusions locales pour émerger, ce qui amplifie les problèmes liés à de mauvaises fusions initiales. De tels scénarios s'accompagnent par ailleurs le plus souvent d'un nombre de dimensions plus élevé, ce qui rend la notion de distance moins utile et brouille donc l'information de localité (et ce d'autant plus que certaines dimensions entièrement non pertinentes pour le problème peuvent éloigner arbitrairement des points les uns des autres pour certaines fonctions de distance).

Pour remédier à ces difficultés, une piste qui s'offre à nous est alors de chercher une variante de nos algorithmes qui permette de s'affranchir complètement de toute notion de distance. C'est déjà dans cette direction que vont les améliorations que nous avons apportées à COCOTTE dans COCOTTE+, qui affaiblissent la dépendance aux premières fusions d'experts (déterminées par les plus proches voisins). Pour continuer dans cette direction, on pourrait par exemple suivre une approche de type séparation et évaluation telle que celle mentionnée au chapitre 4 (mais dont les coûts en calculs restent pour l'instant prohibitifs), ou encore attribuer une complexité à la segmentation de l'espace en régions associées aux divers experts et coupler la minimisation de la complexité des experts à celle de la complexité des régions.

Dans un autre registre, les travaux présentés dans cette thèse se sont limités à un unique choix de S_f (ensemble de familles de fonctions) et de COMP (fonction de complexité sur cet ensemble), et se sont appuyés sur des solveurs de programmes linéaires pour exprimer la routine GETFITTINGFUNCTION (recherche au sein d'une famille de fonctions d'une fonction expliquant un jeu de données). De futurs travaux pourraient donc explorer d'autres choix de S_f et COMP, et pourraient proposer d'autres implémentations de GETFITTINGFUNCTION afin de ne plus se limiter à des combinaisons linéaires de fonctions primitives.

À cette occasion, GETFITTINGFUNCTION pourrait être scindée en deux routines distinctes, selon que les appels à celle-ci cherchent à déterminer si une fonction expliquant les données existe ou qu'ils cherchent à renvoyer une telle fonction. Ceci permettrait probablement de réduire considérablement le coût de nos algorithmes en calculs, et pourrait possiblement rendre viable des approches de séparation et évaluation.

De futurs travaux pourraient également prendre en compte l'imprécision sur les mesures de x en plus de celles de t dans la définition de l'explication d'un jeu de données par une fonction, et fournir des implémentations adaptées.

Extensions à plus long terme

À un horizon plus lointain, divers travaux complémentaires pourraient être menés. On peut par exemple penser à l'élimination des points aberrants (*outliers* en anglais). Ces points étant moins structurés que les autres points de données, on peut en effet s'attendre à ce que les propriétés des experts qui en rendent compte permettent de les identifier.

On peut également envisager l'utilisation des experts pour la résolution du problème inverse (*i.e.* quelles valeurs de x mènent à des valeurs de t souhaitées), par exemple par optimisation locale à partir du point le plus proche vu à l'entraînement.

Un travail pourrait également être réalisé sur les valeurs de x, par exemple en s'autorisant à créer de nouvelles dimensions composites (*e.g.* , une dimension $x_{1+5} = x_1 + x_5$), ce qui impliquerait de s'interroger sur les implications en termes de complexité (*e.g.* , dans l'exemple précédent, $P_{3,\{x_{1+5}\}}$ a une complexité plus faible que $P_{3,\{x_1,x_5\}}$, mais $(x_1, x_5) \mapsto x_{1+5}$ est en elle-même une fonction de $P_{1,\{x_1,x_5\}}$).

Si les résultats sont concluants, ceci pourrait permettre de partir de données brutes (par exemple des images), et de composer les dimensions de celles-ci en caractéristiques plus pertinentes. À défaut, on peut également envisager d'utiliser nos algorithmes après un prétraitement par des algorithmes d'apprentissage de représentations (par exemple ceux s'appuyant sur des réseaux de neurones) qui extrairaient les caractéristiques pertinentes en amont.

Perspectives

À terme, on peut espérer que les travaux entrepris dans cette thèse mènent à des algorithmes permettant de structurer efficacement les interactions entre tout robot et son environnement en compétences à la fois générales et sémantiquement riches.

De tels algorithmes pourraient alors être couplés aux stratégies d'exploration de ces interactions, que ce soit en choisissant où explorer dans l'espace des compétences pour que l'algorithme apprenne plus rapidement (il s'agirait alors d'apprentissage actif, ou *active learning* en anglais), ou en mettant à profit les compétences déjà trouvées au moyen d'algorithmes de planification pour placer le robot dans les domaines où l'on souhaite explorer. Ceci permettrait par exemple lorsque l'exploration amène le robot à saisir un objet d'explorer ensuite les nouvelles possibilité que cela lui offre (tirer l'objet saisi, le soulever, *etc*).

À l'issue de cet apprentissage des compétences, leur richesse sémantique pourrait permettre d'identifier des compétences similaires (*e.g.* , les compétences ⌈*saisir un crayon*⌋ et ⌈*saisir un feutre*⌋) et ainsi permettre au robot de passer sans aide extérieure de l'apprentissage des compétences à celui des affordances (*i.e.* celui de la corrélation entre caractéristiques visuelles locales et compétences). C'est ensuite à partir de ces affordances que pourrait être construit le concept d'objet pour des robots qui apprennent à interagir avec leur environnement de façon autonome.

Annexe A

Pseudo-codes

Pseudo-code A.1 La subroutine GETBESTFIT

```
function GETBESTFIT(S_f, S_d, c_min = S_{f,min}, c_max = S_{f,max})
    ▷ La recherche dichotomique est inutile s'il y a
    ▷  une solution pour c_min,
    candidates ← GETFAMILIES(S_f, c_min, c_min)
    for (𝓕, c) in candidates do
        f ← GETFITTINGFUNCTION(𝓕, S_d)
        if f ≠ NONE then
            return (c, f)
        end if
    end for

    ▷ Dans le cas contraire, on procède à la recherche dichotomique
    ▷ On initialise 'bestSol', qui contiendra la solution à renvoyer
    bestSol ← NONE

    ▷ On initialise 'low'. Les variables 'low' et 'high' contiendront les bornes
    ▷   sur la complexité pour la recherche dichotomique
    low ← c_min + 1            ▷ On sait qu'il n'y a pas de solution pour c_min

    ▷ On vérifie l'existence d'une solution, et le cas échéant on initialise 'high'
    high ← 0
    if c_max < +∞ then
        ▷ Déterminer [rangeMin, rangeMax] à l'aide
        ▷   de COMPLEXITYRANGEMIN
        rangeMax ← c_max
        rangeMin ← max(low, COMPLEXITYRANGEMIN(S_f, rangeMax))
```

```
        ▷ Utiliser getFamilies pour obtenir la liste de familles
        ▷   de fonctions correspondantes
        candidates ← getFamilies(S_f, rangeMin, rangeMax)
        ▷ Utiliser getFittingFunction pour chercher la première
        ▷   solution dans cette liste
        for (𝓕, c) in candidates do
            f ← getFittingFunction(𝓕, S_d)
            if f ≠ None then
                bestSol ← (c, f)
                high ← rangeMin −1          ▷ ⟦ r-Min, r-Max ⟧ déjà exploré
                break
            else
                return None                 ▷ Pas de solution dans S_f
            end if
        end for
    else
        while true do
            ▷ Déterminer [rangeMin, rangeMax] à l'aide
            ▷   de complexityRangeMin
            rangeMax ← 2 * low
            rangeMin ← max(low, complexityRangeMin(S_f, rangeMax))
            ▷ Utiliser getFamilies pour obtenir la liste de familles
            ▷   de fonctions correspondantes
            candidates ← getFamilies(S_f, rangeMin, rangeMax)
            ▷ Utiliser getFittingFunction pour chercher la première
            ▷   solution dans cette liste
            for (𝓕, c) in candidates do
                f ← getFittingFunction(𝓕, S_d)
                if f ≠ None then
                    bestSol ← (c, f)
                    high ← rangeMin −1      ▷ ⟦ r-Min, r-Max ⟧ déjà exploré
                    break
                end if
            end for
            if high = 0 then
                low ← rangeMax +1
            else
                break
            end if
        end while
    end if
```

```
    ▷ On peut maintenant procéder à la recherche dichotomique
    ▷   entre 'low' et 'high'
    while low ≤ high do
        ▷ Déterminer rangeMax, le milieu de [low, high]
        rangeMax ← FLOOR((low + high + 1)/2)
        ▷ En déduire rangeMin à l'aide de COMPLEXITYRANGEMIN
        rangeMin ← max(low, COMPLEXITYRANGEMIN(S_f, rangeMax))
        ▷ Utiliser GETFAMILIES pour obtenir la liste de familles
        ▷   de fonctions correspondantes
        candidates ← GETFAMILIES(S_f, rangeMin, rangeMax)
        ▷ Utiliser GETFITTINGFUNCTION pour chercher la première
        ▷   solution dans cette liste
        success ← false
        for (𝓕, c) in candidates do
            f ← GETFITTINGFUNCTION(𝓕, S_d)
            if f ≠ NONE then
                success ← true
                bestSol ← (c, f)
                high ← rangeMin −1          ▷ ⟦ r-Min, r-Max ⟧ déjà exploré
                break
            end if
        end for
        if not success then
            low ← rangeMax +1
        end if
    end while

    return bestSol
end function
```

Pseudo-code A.2 L'algorithme COCOTTE

```
function COCOTTE(S_f, S_d)
  1. Initialisation :
     ▷ Un expert dédié par point de S_d
     experts ← ∅
     for datapoint ∈ S_d do
         unitSet ← {datapoint}
         (c,f) ← getBestFit (S_f, unitSet, S_f,min, S_f,min)
         newExpert ← (unitSet, c, f)
         experts.add(newExpert)
     end for
  2. Fusion gloutonne :
     ▷ On essaie toutes les paires d'experts
     ▷   en commençant par les plus proches
     distances ← ∅
     for (e0, e1) ∈ experts × experts do
         newItem ← (distance(e0, e1), e0, e1)
         distances.add(newItem)
     end for
     while distances ≠ ∅ do
         distances.sortByFirst()
         (dist, e0, e1) ← distances.pop()
         newExpert ← mergeExperts(S_f, e0, e1)
         if newExpert ≠ None then
             ▷ On retire de 'experts' les éléments e0 et e1
             experts.remove(e0)
             experts.remove(e1)
             ▷ On retire de 'distances' tous les triplets contenant e0 ou e1
             distances.removeAll(e0)
             distances.removeAll(e1)
             ▷ On met à jour experts et distances
             for e ∈ experts do
                 newItem ← (distance(newExpert, e), newExpert, e)
                 distances.add(newItem)
             end for
             experts.add(newExpert)
         end if
     end while
```

3. **Élimination des artefacts :**

```
toProcess ← ∅
for e ∈ experts do
    toProcess.add(e)
end for
while toProcess ≠ ∅ do
    ▷ On vérifie tous les experts un par un
    e_0 ← toProcess.pop()
    oldExperts ← experts                ▷ pour pouvoir revenir en arrière
    experts.remove(e_0)
    (S_{d,0}, c, f) ← e_0
    for datapoint ∈ S_{d,0} do
        unitSet ← {datapoint}
        (c,f) ← getBestFit (S_f, unitSet, S_{f,min}, S_{f,min})
        unitExpert ← (unitSet, c, f)   ▷ Expert individuel pour un point
        ▷ On essaie pour chaque point expliqué par e_0,
        ▷   c'est-à-dire pour chaque expert individuel composant celui-ci,
        ▷   de trouver un autre expert avec lequel il peut être fusionné
        success ← false
        for e_1 ∈ experts do
            newExpert ← mergeExperts(S_f, unitExpert, e_1)
            if newExpert ≠ None then
                success ← true
                experts.remove(e_1)
                experts.add(newExpert)
                toProcess.remove(e_1)
                toProcess.add(newExpert)
                break
            end if
        end for
        ▷ Si l'expert individuel d'un point ne peut être fusionné
        ▷   avec un des autres experts, on annule les changements
        ▷   et on examine l'expert suivant
        if not success then
            experts ← oldExperts
            break
        end if
    end for
end while
```

4. **Détermination des régions :**

▷ Le classifieur crée un label pour chaque expert, et découpe l'espace
▷ en régions correspondant à chaque expert
classifier = GENERATECLASSIFIER()
label ← 0
for e ∈ experts **do**
    ($S_{d,e}$, c, f) ← e
    classifier.add(label, $S_{d,e}$)
    label ← label + 1
end for
classifier.train()

return (experts, classifier)

end function

Pseudo-code A.3 L'algorithme COCOTTE+

```
function COCOTTE(S_f, S_d)
  1. Initialisation :
    ▷ On crée pour chaque point de S_d une feuille contenant
    ▷   un expert dédié, et on garde trace du nœud racine
    ▷   associé à chaque feuille au moyen d'un tableau associatif
    treeCollection ← ∅,   leafRootMap ← ∅
    for datapoint ∈ S_d do
        unitSet ← {datapoint}
        (c,f) ← getBestFit (S_f, unitSet, S_f,min, S_f,min)
        newExpert ← (unitSet, c, f)
        newLeaf ← createNode(newExpert, children=∅)
        treeCollection.add(newLeaf)
        leafRootMap[newLeaf] ← newLeaf
    end for
  2. Fusion d'experts et vol de points :
    ▷ On essaie toutes les paires de feuilles
    ▷   en commençant par les plus proches
    distances ← ∅
    for (leaf0, leaf1) ∈ treeCollection × treeCollection do
        point0 ← leaf0.point,   point1 ← leaf1.point
        newItem ← (distance(point0, point1), leaf0, leaf1)
        distances.add(newItem)
    end for
    distances.sortByFirst()
    while distances ≠ ∅ do
        (dist, leaf0, leaf1) ← distances.pop()
        root0 ← leafRootMap[l0],   root1 ← leafRootMap[l1]
        if root0 ≠ root1 then
            newTrees ← pointStealing(S_f, leaf0, root0, leaf1, root1)
            if newTrees ≠ None then
                treeCollection.remove({root0, root1})
                for tree ∈ newTrees do
                    treeCollection.add(tree)
                    for leaf ∈ tree.getLeaves() do
                        leafRootMap[leaf] ← tree
                    end for
                end for
            end if
        end if
    end while
```

3. **Détermination des régions :**

▷ Le classifieur crée un label pour chaque expert, et découpe l'espace
▷ en régions correspondant à chaque expert
classifier = GENERATECLASSIFIER()
label ← 0
for tree ∈ treeCollection **do**
    e ← tree.expert
    ($S_{d,e}$, c, f) ← e
    classifier.add(label, $S_{d,e}$)
    label ← label + 1
end for
classifier.train()

return (treeCollection, classifier)

end function

Bibliographie

ALTMAN, Naomi S, 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3) :175–185.

BANSAL, Trapit, PACHOCKI, Jakub, SIDOR, Szymon, SUTSKEVER, Ilya et MORDATCH, Igor, 2017. Emergent complexity via multi-agent competition. *CoRR*, abs/1710.03748.
URL http://arxiv.org/abs/1710.03748

BEIJBOM, Oscar, 2012. Domain adaptations for computer vision applications. *CoRR*, abs/1211.4860.
URL http://arxiv.org/abs/1211.4860

BRADSKI, Gary, 2000. The opencv library. *Dr. Dobb's Journal : Software Tools for the Professional Programmer*, 25(11) :120–123.

BREIMAN, Leo, 2001. Random forests. *Machine learning*, 45(1) :5–32.

CLEVELAND, William S et DEVLIN, Susan J, 1988. Locally weighted regression : an approach to regression analysis by local fitting. *Journal of the American statistical association*, 83(403) :596–610.

DA SILVA, Bruno Castro, KONIDARIS, George et BARTO, Andrew G., 2012. Learning parameterized skills. Dans *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*.
URL http://icml.cc/discuss/2012/826.html

DAI, Wenyuan, YANG, Qiang, XUE, Gui-Rong et YU, Yong, 2007. Boosting for transfer learning. Dans *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pages 193–200. doi :10.1145/1273496.1273521.
URL http://doi.acm.org/10.1145/1273496.1273521

DEVIN, Coline, GUPTA, Abhishek, DARRELL, Trevor, ABBEEL, Pieter et LEVINE, Sergey, 2016. Learning modular neural network policies for multi-task and multi-robot transfer. *CoRR*, abs/1609.07088.
URL http://arxiv.org/abs/1609.07088

DREDZE, Mark, KULESZA, Alex et CRAMMER, Koby, 2010. Multi-domain learning by confidence-weighted parameter combination. *Machine Learning*, 79(1-2) :123–149. doi :10.1007/s10994-009-5148-0.
URL https://doi.org/10.1007/s10994-009-5148-0

DRUCKER, Harris, BURGES, Christopher JC, KAUFMAN, Linda, SMOLA, Alex J et VAPNIK, Vladimir, 1997. Support vector regression machines. Dans *Advances in neural information processing systems*, pages 155–161.

ECARLAT, P., CULLY, A., MAESTRE, C. et DONCIEUX, S., 2015. Learning a high diversity of object manipulations though an evolutionary-based babbling. Dans *Proceedings of the workshop Learning Object Affordances, IROS 2015*, pages 1–2. Hambourg.

EFRON, Bradley, HASTIE, Trevor, JOHNSTONE, Iain et TIBSHIRANI, Robert, 2004. Least angle regression. *Ann. Statist.*, 32(2) :407–499. doi :10.1214/009053604000000067.
URL https://doi.org/10.1214/009053604000000067

EVGENIOU, Theodoros et PONTIL, Massimiliano, 2004. Regularized multi–task learning. Dans *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 109–117. doi :10.1145/1014052.1014067.
URL http://doi.acm.org/10.1145/1014052.1014067

FRENCH, Robert M, 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4) :128–135.

FREUND, Yoav et SCHAPIRE, Robert E., 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1) :119–139. doi :10.1006/jcss.1997.1504.
URL https://doi.org/10.1006/jcss.1997.1504

GELADI, Paul et KOWALSKI, Bruce R, 1986. Partial least-squares regression : a tutorial. *Analytica chimica acta*, 185 :1–17.

GERGONNE, JD, 1974. The application of the method of least squares to the interpolation of sequences. *Historia Mathematica*, 1(4) :439–447.

GHADIRZADEH, Ali, BÜTEPAGE, Judith, KRAGIC, Danica et BJÖRKMAN, Mårten, 2016. Self-learning and adaptation in a sensorimotor framework. Dans *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, pages 551–558. doi : 10.1109/ICRA.2016.7487178.
URL http://dx.doi.org/10.1109/ICRA.2016.7487178

GOODFELLOW, Ian, BENGIO, Yoshua et COURVILLE, Aaron, 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

HOFFMAN, Judy, DARRELL, Trevor et SAENKO, Kate, 2014. Continuous manifold based adaptation for evolving visual domains. Dans *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 867–874. doi :10.1109/CVPR.2014.116. URL https://doi.org/10.1109/CVPR.2014.116

JACOBS, Robert A., JORDAN, Michael I., NOWLAN, Steven J. et HINTON, Geoffrey E., 1991. Adaptive mixtures of local experts. *Neural Computation*, 3(1) :79–87. doi :10.1162/neco.1991.3.1.79. URL http://dx.doi.org/10.1162/neco.1991.3.1.79

KIM, Sanghong, KANO, Manabu, NAKAGAWA, Hiroshi et HASEBE, Shinji, 2011. Estimation of active pharmaceutical ingredients content using locally weighted partial least squares and statistical wavelength selection. *International journal of pharmaceutics*, 421(2) :269–274.

KIRKPATRICK, James, PASCANU, Razvan, RABINOWITZ, Neil, VENESS, Joel, DESJARDINS, Guillaume, RUSU, Andrei A, MILAN, Kieran, QUAN, John, RAMALHO, Tiago, GRABSKA-BARWINSKA, Agnieszka *et al.*, 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, page 201611835.

KOBER, Jens, WILHELM, Andreas, OZTOP, Erhan et PETERS, Jan, 2012. Reinforcement learning to adjust parametrized motor primitives to new situations. *Auton. Robots*, 33(4) :361–379. doi :10.1007/s10514-012-9290-3. URL http://dx.doi.org/10.1007/s10514-012-9290-3

KOMPELLA, Varun Raj, STOLLENGA, Marijn F., LUCIW, Matthew D. et SCHMIDHUBER, Jürgen, 2014. Explore to see, learn to perceive, get the actions for free : SKILLABILITY. Dans *2014 International Joint Conference on Neural Networks, IJCNN 2014, Beijing, China, July 6-11, 2014*, pages 2705–2712. doi :10.1109/IJCNN.2014.6889784. URL https://doi.org/10.1109/IJCNN.2014.6889784

KONIDARIS, G.D., KUINDERSMA, S.R., GRUPEN, R.A. et BARTO, A.G., 2011. Cst : Constructing skill trees by demonstration. Dans *Proceedings of the ICML Workshop on New Developments in Imitation Learning*. URL http://lis.csail.mit.edu/pubs/konidaris-icmlws11.pdf

KULIC, Dana, OTT, Christian, LEE, Dongheui, ISHIKAWA, Junichi et NAKAMURA, Yoshihiko, 2012. Incremental learning of full body motion primitives and their sequencing through human motion observation. *I. J. Robotics Res.*,

31(3) :330–345. doi :10.1177/0278364911426178.
URL https://doi.org/10.1177/0278364911426178

Kumar, Abhishek et III, Hal Daumé, 2012. Learning task grouping and overlap in multi-task learning. Dans *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*.
URL http://icml.cc/2012/papers/690.pdf

Lawrence, Neil D. et Platt, John C., 2004. Learning to learn with the informative vector machine. Dans *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*. doi :10.1145/1015330.1015382.
URL http://doi.acm.org/10.1145/1015330.1015382

Lázaro-Gredilla, Miguel, Van Vaerenbergh, Steven et Lawrence, Neil D, 2012. Overlapping mixtures of gaussian processes for the data association problem. *Pattern Recognition*, 45(4) :1386–1395.

Lemme, Andre, Reinhart, René Felix et Steil, Jochen Jakob, 2014. Self-supervised bootstrapping of a movement primitive library from complex trajectories. Dans *14th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2014, Madrid, Spain, November 18-20, 2014*, pages 726–732. doi :10.1109/HUMANOIDS.2014.7041443.
URL http://dx.doi.org/10.1109/HUMANOIDS.2014.7041443

Matricon, Adrien, Filliat, David et Oudeyer, Pierre-Yves, 2016. An iterative algorithm for forward-parameterized skill discovery. Dans *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics, ICDL-EpiRob 2016, Cergy-Pontoise, France, September 19-22, 2016*, pages 186–192. doi :10.1109/DEVLRN.2016.7846816.
URL https://doi.org/10.1109/DEVLRN.2016.7846816

Mihalkova, Lilyana, Huynh, Tuyen N. et Mooney, Raymond J., 2007. Mapping and revising markov logic networks for transfer learning. Dans *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 608–614.
URL http://www.aaai.org/Library/AAAI/2007/aaai07-096.php

Moulin-Frier, Clément et Oudeyer, Pierre-Yves, 2014. Learning how to reach various goals by autonomous interaction with the environment : unification and comparison of exploration strategies. Dans *1st Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM2013), Princeton University, New Jersey*. Princeton, United States.
URL https://hal.inria.fr/hal-00922537

OUDEYER, Pierre-Yves, KAPLAN, Frédéric et HAFNER, Verena Vanessa, 2007. Intrinsic motivation systems for autonomous mental development. *IEEE Trans. Evolutionary Computation*, 11(2) :265–286. doi :10.1109/TEVC.2006.890271.
URL https://doi.org/10.1109/TEVC.2006.890271

PAN, Sinno Jialin et YANG, Qiang, 2010. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.*, 22(10) :1345–1359. doi :10.1109/TKDE.2009.191.
URL https://doi.org/10.1109/TKDE.2009.191

PASTOR, Peter, HOFFMANN, Heiko, ASFOUR, Tamim et SCHAAL, Stefan, 2009. Learning and generalization of motor skills by learning from demonstration. Dans *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*, pages 763–768. doi :10.1109/ROBOT.2009.5152385.
URL https://doi.org/10.1109/ROBOT.2009.5152385

PETERS, Jan et SCHAAL, Stefan, 2008. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4) :682–697. doi :10.1016/j.neunet.2008.02.003.
URL https://doi.org/10.1016/j.neunet.2008.02.003

RAINA, Rajat, BATTLE, Alexis, LEE, Honglak, PACKER, Benjamin et NG, Andrew Y., 2007. Self-taught learning : transfer learning from unlabeled data. Dans *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pages 759–766. doi :10.1145/1273496.1273592.
URL http://doi.acm.org/10.1145/1273496.1273592

ROLF, Matthias et STEIL, Jochen J., 2012. Goal Babbling : a New Concept for Early Sensorimotor Exploration.

ROSENSTEIN, Michael T., MARX, Zvika, KAELBLING, Leslie Pack et DIETTERICH, Thomas G., 2005. To transfer or not to transfer. Dans *In NIPS'05 Workshop, Inductive Transfer : 10 Years Later*.

SCHAAL, Stefan et ATKESON, Christopher G, 1994. Robot juggling : implementation of memory-based learning. *IEEE Control Systems*, 14(1) :57–71.

SCHAAL, Stefan et ATKESON, Christopher G, 1997. Receptive field weighted regression. *ATR Human Information Processing Laboratories, Tech. Rep. TR-H-209*.

SHANNON, Claude E, 1959. Coding theorems for a discrete source with a fidelity criterion. *IRE Nat. Conv. Rec*, 4(142-163) :1.

SIGAUD, Olivier, SALAÜN, Camille et PADOIS, Vincent, 2011. On-line regression algorithms for learning mechanical models of robots : A survey. *Robotics and Autonomous Systems*, 59(12) :1115–1129. doi :10.1016/j.robot.2011.07.006.
URL https://doi.org/10.1016/j.robot.2011.07.006

SMOLA, Alexander J. et SCHÖLKOPF, Bernhard, 2004. A tutorial on support vector regression. *Statistics and Computing*, 14(3) :199–222. doi :10.1023/B:STCO.0000035301.49549.88.
URL https://doi.org/10.1023/B:STCO.0000035301.49549.88

STOLLE, Martin et PRECUP, Doina, 2002. Learning options in reinforcement learning. Dans *Abstraction, Reformulation and Approximation, 5th International Symposium, SARA 2002, Kananaskis, Alberta, Canada, August 2-4, 2002, Proceedings*, pages 212–223. doi :10.1007/3-540-45622-8_16.
URL https://doi.org/10.1007/3-540-45622-8_16

STULP, Freek, HERLANT, Laura, HOARAU, Antoine et RAIOLA, Gennaro, 2014. Simultaneous on-line discovery and improvement of robotic skill options. Dans *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*, pages 1408–1413. doi :10.1109/IROS.2014.6942741.
URL https://doi.org/10.1109/IROS.2014.6942741

STULP, Freek et SIGAUD, Olivier, 2015. Many regression algorithms, one unified model : A review. *Neural Networks*, 69 :60–79. doi :10.1016/j.neunet.2015.05.005.
URL http://dx.doi.org/10.1016/j.neunet.2015.05.005

SUN, Shiliang, SHI, Honglei et WU, Yuanbin, 2015. A survey of multi-source domain adaptation. *Information Fusion*, 24 :84–92. doi :10.1016/j.inffus.2014.12.003.
URL https://doi.org/10.1016/j.inffus.2014.12.003

SUTTON, Richard S., PRECUP, Doina et SINGH, Satinder P., 1999. Between MDPs and semi-MDPs : A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2) :181–211. doi :10.1016/S0004-3702(99)00052-1.
URL http://dx.doi.org/10.1016/S0004-3702(99)00052-1

TAYLOR, Matthew E., STONE, Peter et LIU, Yaxin, 2005. Value functions for rl-based behavior transfer : A comparative study. Dans *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 880–885.
URL http://www.aaai.org/Library/AAAI/2005/aaai05-139.php

TENENBAUM, Joshua B, DE SILVA, Vin et LANGFORD, John C, 2000. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500) :2319–2323.

VAPNIK, Vladimir N., 1999. The nature of statistical learning theory.

VIJAYAKUMAR, Sethu et SCHAAL, Stefan, 2000. Locally weighted projection regression : Incremental real time learning in high dimensional space. Dans *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 1079–1086.

WOLD, Svante, ESBENSEN, Kim et GELADI, Paul, 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3) :37–52.

WOLPERT, Daniel M. et KAWATO, Mitsuo, 1998. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7-8) :1317–1329. doi : 10.1016/S0893-6080(98)00066-5.
URL https://doi.org/10.1016/S0893-6080(98)00066-5

WUNDERLING, Roland, 1997. Soplex : The sequential object-oriented simplex class library.

YANG, Yongxin et HOSPEDALES, Timothy M., 2014. A unified perspective on multi-domain and multi-task learning. *CoRR*, abs/1412.7489.
URL http://arxiv.org/abs/1412.7489