

THÈSE DE DOCTORAT DE
L'UNIVERSITÉ DE LYON

École Doctorale ED 512
Informatique et Mathématique de Lyon
(INFOMATHS)
Spécialité Informatique

Présentée par

Quentin Debard

Sujet de la thèse :

Automatic Learning of Next Generation Human-Computer Interactions

Apprentissage Automatique des Interactions Homme-Machine de la Prochaine Génération

Devant le jury composé de :

Frédéric Precioso	Professeur, Université de Nice	Rapporteur
Elisa Fromont	Professeur, Université de Rennes	Rapportrice
Gilles Bailly	Chercheur CNRS, Sorbonne Université	Examinateur
Chloé Clavel	Professeur, Telecom-ParisTech	Examinatrice
Radu Horaud	Directeur de Recherche, INRIA Rhône-Alpes	Examinateur
Jilles Dibangoye	Maître de Conférences, INSA de Lyon	Invité
Stéphane Canu	Professeur, INSA de Rouen	Co-directeur de thèse
Christian Wolf	Maître de Conférences, HDR, INSA de Lyon	Directeur de thèse

ABSTRACT

Artificial Intelligence (AI) and Human-Computer Interactions (HCIs) are two research fields with relatively few common work. HCI specialists usually design the way we interact with devices directly from observations and measures of human feedback, manually optimizing the user interface to better fit users' expectations. This process is hard to optimize: ergonomy, intuitivity and ease of use are key features in a User Interface (UI) that are too complex to be simply modelled from interaction data. This drastically restrains the possible uses of Machine Learning (ML) in this design process. Currently, ML in HCI is mostly applied to gesture recognition and automatic display, e.g. advertisement or item suggestion. It is also used to fine tune an existing UI to better optimize it, but as of now it does not participate in designing new ways to interact with computers.

Our main focus in this thesis is to use ML to develop new design strategies for overall better UIs. We want to use ML to build intelligent – understand precise, intuitive and adaptive – user interfaces using minimal handcrafting. We propose a novel approach to UI design: instead of letting the user adapt to the interface, we want the interface and the user to adapt mutually to each other. The goal is to reduce human bias in protocol definition while building co-adaptive interfaces able to further fit individual preferences.

In order to do so, we will put to use the different mechanisms available in ML to automatically learn behaviors, build representations and take decisions. We will be experimenting on touch interfaces, as these interfaces are vastly used and can provide easily interpretable problems. The very first part of our work will focus on processing touch data and use supervised learning to build accurate classifiers of touch gestures. The second part will detail how Reinforcement Learning (RL) can be used to model and learn interaction protocols given user actions. Lastly, we will combine these RL models with unsupervised learning to build a setup allowing for the design of new interaction protocols without the need for real user data.

RÉSUMÉ

L'Intelligence Artificielle (IA) et les Interfaces Homme-Machine (IHM) sont deux champs de recherche avec relativement peu de travaux communs. Les spécialistes en IHM conçoivent habituellement les interfaces utilisateurs directement à partir d'observations et de mesures sur les interactions humaines, optimisant manuellement l'interface pour qu'elle corresponde au mieux aux attentes des utilisateurs. Ce processus est difficile à optimiser : l'ergonomie, l'intuitivité et la facilité d'utilisation sont autant de propriétés clé d'une interface utilisateur (IU) trop complexes pour être simplement modélisées à partir de données d'interaction. Ce constat restreint drastiquement les utilisations potentielles de l'apprentissage automatique dans ce processus de conception. A l'heure actuelle, l'apprentissage automatique dans les IHMs se cantonne majoritairement à la reconnaissance de gestes et à l'automatisation d'affichage, par exemple à des fins publicitaires ou pour suggérer une sélection. L'apprentissage automatique peut également être utilisé pour optimiser une interface utilisateur existante, mais il ne participe pour l'instant pas à concevoir de nouvelles façons d'intéragir.

Notre objectif avec cette thèse est de proposer grâce à l'apprentissage automatique de nouvelles stratégies pour améliorer le processus de conception et les propriétés des IUs. Notre but est de définir de nouvelles IUs intelligentes – comprendre précises, intuitives et adaptatives – requérant un minimum d'interventions manuelles. Nous proposons une nouvelle approche à la conception d'IU : plutôt que l'utilisateur s'adapte à l'interface, nous cherchons à ce que l'utilisateur et l'interface s'adaptent mutuellement l'un à l'autre. Le but est d'une part de réduire le biais humain dans la conception de protocoles d'interactions, et d'autre part de construire des interfaces co-adaptatives capables de correspondre d'avantage aux préférences individuelles des utilisateurs.

Pour ce faire, nous allons mettre à contribution les différents outils disponibles en apprentissage automatique afin d'apprendre automatiquement des comportements, des représentations et des prises de décision. Nous expérimenterons sur les interfaces tactiles pour deux raisons majeures : celles-ci sont largement utilisées et fournissent des problèmes facilement interprétables. La première partie de notre travail se focalisera sur le traitement des données tactiles et l'utilisation d'apprentissage supervisé pour la construction de classifieurs précis de gestes tactiles. La seconde partie détaillera comment l'apprentissage par renforcement peut être utilisé pour modéliser et apprendre des protocoles d'interaction en utilisant des gestes utilisateur. Enfin, nous combinerons ces modèles d'apprentissage par renforcement avec de l'apprentissage non supervisé pour définir une méthode de conception de nouveaux protocoles d'interaction ne nécessitant pas de données d'utilisation réelles.

CONTENTS

ABSTRACT iii

RÉSUMÉ v

CONTENTS vii

LIST OF FIGURES ix

LIST OF TABLES xi

ACRONYMS xiii

1 INTRODUCTION 1
- 1.1 Context . . . 1
  - 1.1.1 User Interfaces . . . 1
  - 1.1.2 Machine Learning . . . 3
- 1.2 Motivations . . . 4
- 1.3 About This Thesis . . . 7
  - 1.3.1 Contributions . . . 7
  - 1.3.2 Publications . . . 9

2 BACKGROUND AND RELATED WORKS 11
- 2.1 User Interfaces . . . 11
  - 2.1.1 A Short History . . . 11
  - 2.1.2 Discrete Interactions: Touch Gesture Recognition . . . 13
  - 2.1.3 Continuous Interactions: Interaction Protocols . . . 15
- 2.2 Supervised Deep Learning for Classification on Sequential Data . . . 16
  - 2.2.1 Fully Connected Networks and Backpropagation . . . 17
  - 2.2.2 Convolutional Neural Networks . . . 20
  - 2.2.3 Recurrent Neural Networks . . . 23
  - 2.2.4 Multi-Dimensional Recurrent Neural Networks . . . 25
- 2.3 Reinforcement Learning for Continuous Control . . . 27
  - 2.3.1 Policy gradient . . . 28
  - 2.3.2 Q-learning . . . 29
  - 2.3.3 Actor-Critic . . . 30
  - 2.3.4 Deep Deterministic Policy Gradient . . . 31
- 2.4 Artificial Intelligence and Human-Computer Interfaces . . . 32

3 TOUCH GESTURE RECOGNITION 37
- 3.1 Touch Gesture Data . . . 38
  - 3.1.1 Itekube-7 . . . 39
  - 3.1.2 Feature Preserving Sampling . . . 42
- 3.2 Neural Models . . . 46
  - 3.2.1 Convolutional MDGRU . . . 48
- 3.3 Experimental Results . . . 50
  - 3.3.1 Experimental setup . . . 52

3.3.2 Ablation Study . . . 54
3.3.3 Comparison with the state of the art . . . 55
3.4 Conclusions . . . 57
4 CONTINUOUS COUPLING OF GESTURES AND ACTIONS 59
4.1 Interaction Protocols as Reinforcement Learning Agents . . . 60
4.1.1 Formalizing the Interaction Protocol Design Problem . . . 61
4.1.2 Learning Signals and Environments . . . 65
4.2 Experiment: Learning a Touch Interaction Protocol . . . 67
4.2.1 Environment and known solution . . . 68
4.2.2 Handcrafted User Model . . . 69
4.2.3 Reward Function . . . 70
4.2.4 Experimental Setup . . . 71
4.2.5 Results . . . 73
4.3 Conclusions . . . 73
5 USER MODELLING 75
5.1 Natural Gesture Distribution Approximation . . . 76
5.1.1 Generative Models for Distribution Approximation . . . 76
5.2 Recurrent Variational Auto-Encoder . . . 78
5.3 Experiments . . . 80
5.3.1 Architectures . . . 80
5.3.2 Reconstruction and Generation Capabilities . . . 82
5.3.3 Walking the Manifold Encoded in the Latent Space . . . 84
5.4 Conclusions . . . 85
6 LEARNING COMPLEX CO-ADAPTIVE USER INTERFACES 87
6.1 Multi-Agent Reinforcement Learning Setup . . . 88
6.1.1 User Model . . . 88
6.1.2 Formalization . . . 91
6.2 Experiment: Learning a 3D Navigation Interaction Protocol . . . 93
6.2.1 Environment . . . 94
6.2.2 Experimental Details . . . 95
6.2.3 Results . . . 96
6.3 Conclusions . . . 97
7 CONCLUSION: ACHIEVEMENTS, LIMITATIONS AND PERSPECTIVES 99
7.1 Contributions . . . 99
7.1.1 Supervised Learning for Touch Gesture Recognition . . . 100
7.1.2 Interaction Protocol Design . . . 100
7.2 Perspectives and Future Work . . . 102
7.2.1 Real-Time Gesture Recognizer . . . 102
7.2.2 More Human-Like User Models . . . 102
7.2.3 Memory-Based Interface Agents . . . 103
7.2.4 Adaptation During Real Use . . . 103
7.2.5 A Final Point . . . 103

BIBLIOGRAPHY 105

LIST OF FIGURES

Chapter 1: Introduction 1
Figure 1.1 Touch Interface Example . . . 2
Figure 1.2 Two Complementary Tasks . . . 6
Figure 1.3 Eva Table . . . 7

Chapter 2: Background and Related Works 11
Figure 2.1 Gesture Coder . . . 14
Figure 2.2 Fully Connected Network . . . 18
Figure 2.3 Convolutional Layer . . . 21
Figure 2.4 Automated Menu Display . . . 34

Chapter 3: Touch Gesture Recognition 37
Figure 3.1 Touch Acquisition . . . 39
Figure 3.2 Raw Data . . . 40
Figure 3.3 Itekube-7 Statistics . . . 41
Figure 3.4 Itekube-7 Classes . . . 42
Figure 3.5 Contact Matrix . . . 43
Figure 3.6 Dynamic Sampling . . . 46
Figure 3.7 Deep Learning Models . . . 48
Figure 3.8 CNN Architecture . . . 49
Figure 3.9 MMG Classes . . . 56

Chapter 4: Continuous Coupling of Gestures and Actions 59
Figure 4.1 3D Touch Interface Example . . . 61
Figure 4.2 RL Formalization of an Interaction Protocol . . . 63
Figure 4.3 RST interactions . . . 67
Figure 4.4 2D Environment . . . 68
Figure 4.5 2D Rollout Illustration . . . 73

Chapter 5: User Modelling 75
Figure 5.1 Sequential VAE . . . 79
Figure 5.2 VAE Reconstruction Examples . . . 83
Figure 5.3 VAE Latent Space . . . 84

Chapter 6: Learning Complex Co-Adaptive User Interfaces 87
Figure 6.1 Naive MARL Setup . . . 88

Figure 6.2 Complete MARL Setup 92
Figure 6.3 3D Navigation Environment 94
Figure 6.4 3D Rollout Illustration 97

Chapter 7: conclusion: achievements, limitations and perspectives 99

LIST OF TABLES

Chapter 1: introduction 1

Chapter 2: background and related works 11

Chapter 3: touch gesture recognition 37
Table 3.1 Itekube-7 Results 51
Table 3.2 Confusion Matrix 52
Table 3.3 Sampling Comparison 54
Table 3.4 Variation in Parameters Results 55
Table 3.5 MMG Results 57

Chapter 4: continuous coupling of gestures and actions 59

Chapter 5: user modelling 75
Table 5.1 VAE loss comparison 82

Chapter 6: learning complex co-adaptive user interfaces 87
Table 6.1 MARL Results 96

Chapter 7: conclusion: achievements, limitations and perspectives 99

ACRONYMS

AI	Artificial Intelligence
AUI	Adaptive User Interface
BPTT	BackPropagation Through Time
CLI	Command-Line Interface
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DL	Deep Learning
DNN	Deep Neural Network
FC	Fully Connected
FCN	Fully Connected Network
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HSMM	Hidden Semi-Markov Model
LSTM	Long Short-Term Memory
MARL	Multi-Agent Reinforcement Learning
MDGRU	Multi-Dimensional Gated Recurrent Unit
MDLSTM	Multi-Dimensional Long Short-Term Memory
MDP	Markov Decision Process
ML	Machine Learning
MDRNN	Multi-Dimensional Recurrent Neural Network
NN	Neural Network
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
RST	Rotate-Scale-Translate
RNN	Recurrent Neural Networks
SVM	Support Vector Machine
TD	Temporal Difference
UI	User Interface
VAE	Variational Auto-Encoder

CHAPTER 1

INTRODUCTION

Contents

1.1 Context 1
1.1.1 User Interfaces 1
1.1.2 Machine Learning 3
1.2 Motivations 4
1.3 About This Thesis 7
1.3.1 Contributions 7
1.3.2 Publications 9

1.1 Context

In both our personal and professional life, we are using more and more electronic devices to perform a growing range of tasks. Repetitive tasks and simple decision processes are automated, access to pertinent information and representation of said information is optimized, and new interactions of increasing abstraction are made possible. One of the many pieces required to propose such interactions are User Interfaces (UIs).

1.1.1 User Interfaces

Using any form of electronic device requires a UI to communicate with it. In one way, the UI translates the user intent to the device, in the other it gives the user an interpretable representation of the state and/or results computed by the device. In the user-to-computer direction, a UI usually has a physical part (button, keyboard, touchscreen...) and an interaction protocol, i.e. a translation from user actions on the physical interface to properties of the query to input. The computer-to-user direction involves the transmission of pertinent system information to the user, classically in the form of a display, audio signal or in few cases haptic feedback.

These UIs vastly evolved in the last 80 years along with computational capabilities, gradually gaining versatility, expressivity and intuitiveness. Nowadays, we can perform a wide range of actions through the use of electronic devices: from

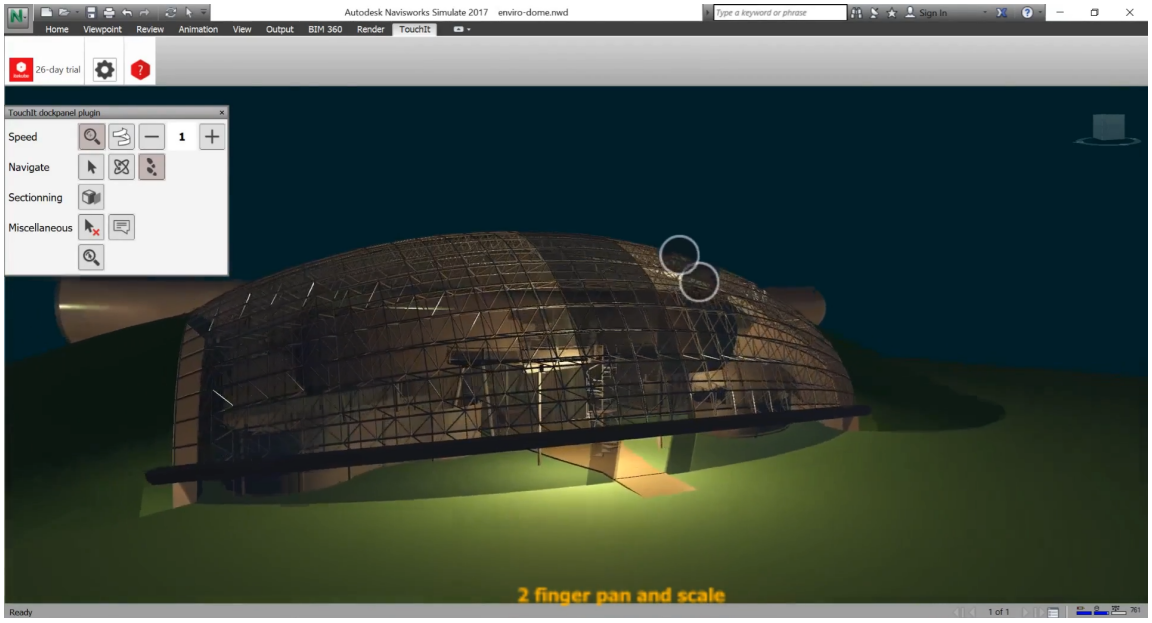

Figure 1.1 – Example of a touch UI for 3D navigation. This is the Autodesk's Navisworks software combined with Itekube's TouchIT, an overlay used to identify and interpret finger motions to actions in the application. Fingers in contact with the surface are represented as white circles. Their current motion is interpreted as a 2 finger pan and scale and is applied with the set parameters (displayed in the top-left window).

conceptually simple actions such as media display or communication, to more complex ones such as trading, modelization or industrial design. For each of these actions, specialists have to design a proper UI in order to perform them. Part of the Human-Computer Interaction (HCI) research focuses on making UIs easier to use, more precise, and allow for new and more abstract interactions. While the optimality of these UIs can be hard to measure, some UI components have been widely accepted as standards. The mouse/keyboard combo and smartphones are good examples of versatile, well accepted physical interfaces. The software part of UIs is also subject to active research, even for well-known physical interfaces. As an illustration, we display in Figure 1.1 a touch interface in development for navigation in 3D scenes.

A UI tries to maximize positive user feedback from experience: if an UI receives some negative feedback from test users, this UI can be updated to address the corresponding problems, or a new paradigm can be proposed instead to better satisfy users.

1.1.2 Machine Learning

Whenever we face a data-related optimization problem, Machine Learning (ML) is a tool to consider. Since 2012 (Krizhevsky et al. 2012), ML has seen growing interest from the academic and the industrial world through supervised Deep Learning (DL). DL, although lacking a formal definition, refers to neural models with multiple hidden layers. The interest of these hidden layers is to build internal representations of the data or the distribution the model is learning from. Shining in image classification, DL was quickly applied to any kind of data for an array of different tasks: classification, segmentation, structured representation, item generation... DL is now a widely used solver for many objective function maximization (or loss minimization) problems. Supervised DL however requires a label (or ground truth) from a human supervisor for each observed sample. This constraint makes the building of a dataset extremely time-consuming for complex tasks and prone to mislabelling. Different learning signals can be used to train DL models: Deep Reinforcement Learning (RL) for example uses instead a reward signal for the DL model to maximize. This allows for the use of unlabelled data and minimizes the need for human supervision during training.

While ML solutions still tend to generalize poorly in the face of real data, because of noise and factors not met during training, they can excel given controlled conditions. If these models are deployed in "close enough" conditions, they tend to significantly outperform handcrafted models. However, one of the major drawbacks of DL algorithms is the requirement of huge amounts of data to train a good model. In applications where data is scarse, applying DL can prove challenging. So, in short, DL is not necessarily the way to go whenever facing an optimization problem: a number of conditions needs to be met in order for DL to add value to the optimization process. In our case however, provided we correctly break down the whole UI design process, we state that ML could be used for several parts and add abstraction and precision compared to existing methods.

Applying ML to UI design is not a new concept, although mostly all the corresponding works can fit into two coarse categories: **item suggestion** in the computer-to-user direction and **action recognition** in the user-to-computer direction.

Item suggestion — In the first category, search engines are a widely used and researched application. For example, in Li et al. 2019, authors use a combination of decision trees and neural networks to train learning-to-rank algorithms for personal search. We can also cite Dehghani et al. 2017, who proposed a webpage ranking model using weakly supervised neural networks. Advertisement is another highly used application: for example, Perlich et al. 2013 describe a large-scale machine learning system for targeted display advertising using transfer learning. At last, we can cite graphical display adaptation, with Moran et al. 2018 using a machine learning model to automatically write Graphical User

Interface (GUI) prototypes from a mock-up. All these applications are focused on *"What to show to the user?"*.

Action recognition — This research topic is extremely broad, and we can find extensive work on many types of sensors with different ML architectures:

- touch gesture classification (Lü et al. 2012),
- hand gesture recognition (Benalcázar et al. 2017),
- speech recognition (Oord et al. 2016),
- action recognition in videos (Zhu et al. 2018; Sanabria et al. 2019),
- action recognition from 3D data (Halim et al. 2016),
- eye tracking (Krafka et al. 2016),
- or haptic feedback (Sun et al. 2019).

These works cover the question *"What action was performed by the user?"*.

This leaves one question to encapsulate the complete UI: *"How to interpret this action?"*. To our knowledge, there has been no attempt to answer this question using ML. Automating this process will be the main focus of this thesis.

1.2 Motivations

Developing better interfaces allows for better accessibility to specific tools, a gain in productivity for professional interfaces and even an access to new interactions (augmented reality for example).

If more and more interactions tend to be well optimized – understand intuitive, fast and precise –, abstract and complex actions still tend to suffer from limited interfaces. In these cases, the "optimal" way to interact is either hard to define, or just does not exist: different users might in fact prefer or need to perform some actions in a different fashion. This makes the optimization process of the interaction protocol potentially intractable. Furthermore, we want to optimize a vague notion of what we can call "user satisfaction" which is not directly measurable. For a same UI, this satisfaction can vary depending on the user, making it even harder to validate the quality of a solution. This validation step will usually involve a number of test users providing feedback while using the UI combined with performance metrics.

We can observe a few limitations to handcrafted UI design:

- the design can be subject to a specialist bias that might not be suitable for less experienced users,
- iteratively optimizing an existing UI can be time and resource-consuming,

- once designed, the adaptability of the UI to the user is limited.

We claim that with a proper formalism, ML can help mitigate these drawbacks while still allowing for the discovery of new UIs. As said earlier, we will work on the software user-to-computer part of the UI. The interest of doing so is twofold:

- potentially discover more efficient interaction protocols,
- and design co-adaptive interfaces, able to change during use.

We will illustrate this claim by experimenting on touchscreen-related tasks. We will gradually tackle more complex problems using ML to end-up with the automatic learning of an interaction protocol for 3D navigation. This is a real problem with no globally accepted solution and being able to automatically find a satisfying solution will serve as a realistic proof of concept for our approach.

One of the biggest challenges with using ML comes from the limited amount of data available: collecting data from humans in a specific context is inherently hard, even more so when the data we want to observe depends on a variety of factors (application state, personal preferences, type of device used...). This problem is particularly present in HCI, as some of the core challenges come from interpreting human intent and displaying information in the most comprehensible way. Thus, human feedback is potentially needed during multiple phases of the design. It is at least required for the validation of a UI.

This data limitation makes the training of ML models extremely hard, and it will require a careful definition of the training protocol and the models to still be able to obtain satisfying solutions. In the case of interaction protocol learning, training only from human gestures rapidly becomes unrealistic as performed gestures depend on both the user intent and the application state. Training a ML model would require huge amounts of data even for simple interaction protocols. As we will show throughout this thesis, we will need to design strategies and properly process data to gain knowledge from a limited set of real interactions. This will allow us to solve real-life problems without the need for human input at each step of the training phase of our models.

As we are wanting to perform more abstract and more complex actions with high dimension physical interfaces (multitouch multiuser surfaces, augmented reality, haptic sensors...), designing good UIs becomes the more complicated. Our goal with this work is to help with the design of such UIs by automating part of the design process on one hand, and on the other let the interface protocol adapt while it is used, thus fitting even more to the user's needs.

In this thesis, we formulate the software user-to-computer communication as two distinctive problems (Figure 1.2): first the recognition of discrete gestures, and second the continuous coupling of gestures and actions. The recognition of discrete gestures is the most intuitive task, requiring to identify the type and eventual properties of a user action, e.g. recognizing a zoom or a slide gesture

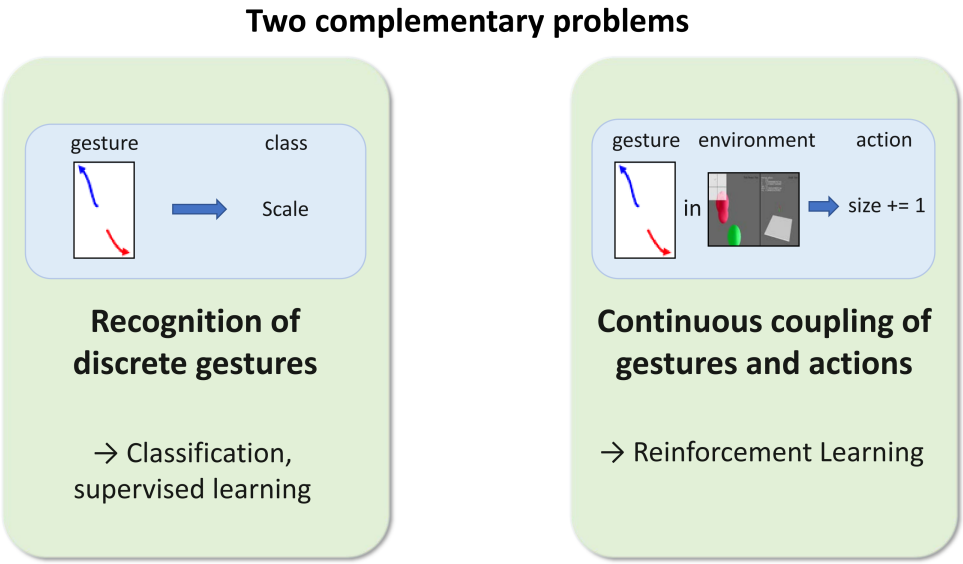

Figure 1.2 – The design process of the user-to-computer direction of a UI can be split into two problems. They are usually tackled separately with a gesture recognizer identifying the type and potential parameters of the user action and an interaction protocol assigning the corresponding action in the application. We will show later in this thesis that gesture recognition does not necessarily needs to be explicit, and a ML model can internalize this process to assign an action in the application w.r.t. the user action.

on a touch surface. This is a classification and eventually a regression task, traditionally solved in ML using supervised learning. The second problem deals with the correspondence between user actions on the device and actions in a virtual environment. For example, attributing a zoom gesture on a touch surface to a "move forward" action in a 3D navigation environment. A good solution for this task is extremely important for a UI to be satisfying: this correspondence process heavily weighs on the intuitiveness and the naturalness of the overall UI. Its design is complicated: there are no direct metrics to tell how good a solution is. This task is classically complementary to the recognition task, meaning that we attribute actions in a virtual environment with respect to the corresponding class and properties of a gesture. We will show that RL can be used to perform both tasks at once, with the recognition process being performed implicitly by the RL agent.

Figure 1.3 – Illustration of a Model E (EVA) produced by Itekube. It is a multiuser, multitouch touch table used from document manipulation to collaborative 3D design reviews.

1.3 About This Thesis

This work is the fruit of an industrial partnership between the Itekube company, the LIRIS and LITIS laboratories. Itekube produces hi-tech touch tables with a focus on industrial applications. These tables are thought to be used as a multiuser display and workspace, meant to propose an intuitive and accessible interface for potentially complex uses. Their touch tables are used in a variety of contexts, to cite a few: medical imaging, blueprint display on construction sites, computer assisted design or city mapping display.

While software and hardware requirements are getting more and more demanding, Itekube wants to use its knowledge to provide efficient hardware as well as intelligent software to interact with the client's applications in the best way possible (see Figure 1.3 for an illustration of said hardware). Along with standard methods, Itekube is looking for possible Artificial Intelligence (AI) solutions in order to provide better, co-adapative interfaces. This PhD project was elaborated to start conceiving such methods.

1.3.1 Contributions

The work achieved in this thesis is split in four different chapters, highlighting different fundamental concepts:

- Chapter 3 describes our work on gesture recognition using novel data processing and supervised learning. We compare a variety of different DL architectures on a classification task using a novel interaction gesture dataset, Itekube-7, which was made publicly available. We also develop a novel architecture, dubbed Conv-MDGRU, a compact combination of convolution operations for input observation and multidimensional recurrence between states. This architecture is our best performing model while possessing the lowest number of trainable weights. Itekube-7 contains 6591 unique gestures performed by 27 different users, split into 7 classes. We can sum up our contributions as a novel DL architecture, a touch data processing and an interaction touch gesture dataset.

- Chapter 4 addresses the problem of the continuous coupling of user gestures and actions in a simplified environment using RL. We formalize this problem with a novel ML perspective and test this setup by re-learning a well-known solution: the "pinch-and-zoom" protocol allowing for the linear manipulation of 2D objects. This training is done using a custom 2D environment and an handcrafted user model, allowing us to use synthesized user gestures instead of human gestures. The handcrafted user model however constrains us to learn a specific solution: this motivates us to automatically learn user models in specific conditions to avoid using human gestures while synthesizing coherent data to train our models. Our contributions in this chapter is the formalization of interaction protocol learning as a RL problem and the constitution of a proper environment to re-learn a known interaction protocol.

- Chapter 5 focuses on learning in an unsupervised fashion a representation of natural gestures. We will train Variational Auto-Encoders (VAEs) to build robust features from smooth and disentangled latent spaces using Itekube-7. The quality of the latent space and the reconstructions are mostly assessed through visualization. The goal is to be able to generate human-like gestures from semantically meaningful representations. Such a model can be used in a user model afterward. Our contributions are the learning and the study of disentangled representations of touch gestures as well as a novel VAE architecture for sequential data.

- In Chapter 6, we combine the work from the two preceding chapters to propose a final setup for the automatic learning of complex interaction protocols. We define a user model combining RL and part of a pre-trained VAE to act as the user. This user model is trained in parallel with the interaction protocol model in order to solve a specific task. Thus, both agents cooperate to solve the task: the user model produces gestures, and the interaction protocol model interprets them to actions in the environment.

We experiment on a 3D navigation environment. There are two contributions in this chapter: first, the definition of a novel model combining a RL agent and a VAE to produce training data. Second, the design of a Multi-Agent Reinforcement Learning (MARL) setup allowing for the automatic learning of interaction protocols.

1.3.2 Publications

The work presented in this thesis can be split into two main parts that each led to a publication.

The first part deals with multitouch gesture processing and the use of DL for gesture recognition, resulting in a novel touch gesture processing method and comparing different DL models on the task. The second part uses mainly RL for the automatic design of interaction protocols, and proposes a novel combination of RL and VAE for user modelling. Associated publications are:

- Quentin Debard, Christian Wolf, Stéphane Canu, and Julien Arné (2018). "Learning to Recognize Touch Gestures: Recurrent vs. Convolutional Features and Dynamic Sampling". In: *The International Conference on Automatic Face and Gesture Recognition (FG)*;

- Quentin Debard, Jilles Steeve Dibangoye, Stéphane Canu, and Christian Wolf (2019). "Learning 3D Navigation Protocols on Touch Interfaces with Cooperative Multi-Agent Reinforcement Learning". In: *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD)*.

Early work during this thesis led to the constitution of a multitouch gesture dataset, namely Itekube-7. After publication, this dataset was made publicly available at the url `http://itekube7.itekube.com`.

CHAPTER 2

BACKGROUND AND RELATED WORKS

Contents

2.1 User Interfaces 11
2.1.1 A Short History 11
2.1.2 Discrete Interactions: Touch Gesture Recognition 13
2.1.3 Continuous Interactions: Interaction Protocols 15
2.2 Supervised Deep Learning for Classification on Sequential Data 16
2.2.1 Fully Connected Networks and Backpropagation 17
2.2.2 Convolutional Neural Networks 20
2.2.3 Recurrent Neural Networks 23
2.2.4 Multi-Dimensional Recurrent Neural Networks 25
2.3 Reinforcement Learning for Continuous Control 27
2.3.1 Policy gradient 28
2.3.2 Q-learning 29
2.3.3 Actor-Critic 30
2.3.4 Deep Deterministic Policy Gradient 31
2.4 Artificial Intelligence and Human-Computer Interfaces 32

2.1 User Interfaces

User Interfaces (UIs) are the mediation tools between a user and an electronic device. They encompass a variety of techniques and cover the hardware as well as the software side of the interaction process. Designing a UI is a complex task at the crossroads between a variety of fields from computer science to psychology and cognitive science. This design process falls in the Human-Computer Interaction (HCI) research field.

2.1.1 A Short History

The first UIs in the 40's were programming interfaces, used in the form of punchcards to declare a sequence of operations to be computed. At that time,

computers mostly solved arithmetic problems. In the 60's, for the first time, a computer is used as an interactive tool rather than a program-based calculator (Sutherland 1964). In the 70's, Alto from the Xerox Palo Alto Research Center becomes the first Graphical User Interfaces (GUIs) prototype with a desktop notion, encasing a collection of programs. At that time, computer users interacted using Command-Line Interfaces (CLIs). More intuitive, GUIs allow for What-You-See-Is-What-You-Get (WYSIWYG) interfaces, meaning a screen can display a visual representation of a process or a file, instead of displaying a functional or programmed representation. In 1984, with researchers coming from Xerox, Apple produces the first personal GUI computer, the Macintosh: this is the beginning of the democratization of personal computers (PCs). PCs evolved in hardware and software during the 90's; GUIs were enhanced to be used for different and more complex tasks while CLIs got more and more marginal for common users.

In the 2000's, the refinement of capacitive touch screens allows for the large scale distribution of a different type of interface in the form of smartphones. While the LG Prada from LG Electronics is officially the first phone to use a capacitive touchscreen on top of its screen (released in 2007), the Apple's iPhone released the same year will get more exposure. This is the first popular alternative to the mouse-keyboard combo. If touchscreens really shone from that point on, their concept goes back to 1965 (Johnson 1965). In this paper, Johnson describes the concept of capacitive touchscreens, using the electromagnetic perturbation of a finger to detect a touch. Shortly after in 1972, optical touchscreens are described in a patent from the University of Illinois. Instead of measuring capacitance, optical touchscreens use infrared sensors to detect the intersection of a beam with a finger. In 1975, the first resistive touchscreen is described in a patent filled by George Samuel Hurst. Resistive touchscreens detect a touch using two conductive layers that come in contact when pressed. This last type of touchscreen will be the most used until the smartphone era.

Keyboard/mouse and touchscreens make up for most of our interactions with computers as of today. New interfaces such as voice command or augmented reality are getting increasing exposure with their precision and versatility improving, but their use can still be considered marginal. This is one of the reasons behind our focus throughout this thesis on touchscreen experiments: while the concepts we develop can be applied to virtually any UI, we will be experimenting exclusively on touchscreens. This allows us to provide easily understandable experiments on a largely used physical interface and minimize the necessary engineering workload to train and deploy models. As such, this thesis will not provide a full state of the art in UI but will focus instead on touch interfaces. For a thorough study of the history of HCI, see Grudin 2016.

2.1.2 Discrete Interactions: Touch Gesture Recognition

Automatic human gesture recognition is an extremly prolific field. Using many different sensors such as RGB cameras, depth sensors, body sensors, in our case touch surfaces, these gestures are classified and measured through representations: geometric features, graphs, state machines, sequences, and more recently, learned features. The classifying and measuring algorithms are also varied, ranging from deterministic decisions to Support Vector Machines (Cortes et al. 1995) and Deep Neural Networks. We brush in this subsection a state of the art of discrete touch gesture recognition, i.e. the classification of a sequence of user finger trajectories into a unique class.

Touch gestures — can be distinguished into two types:

- **symbols**, such as drawings or handwriting. These gestures are spatially complex, but their temporal properties are of no interest to properly classify them;

- **interactions**, meant to perform an action using the touch surface as an interface. These actions require spatial and temporal precision, as the user will expect the interaction to be as precise and fast as possible.

Classifying symbolic gestures is a simpler task than classifying interaction gestures. In Chapter 3, we will focus on the multitouch gesture recognition of interaction gestures using Deep Learning (DL) models. Touch gestures are traditionally dealt with handcrafted representations. The most commonly used methods have been developed by system designers, using procedural event-handling algorithms (see for instance M. Wu et al. 2003 or Malik et al. 2005). Different frameworks such as Gesture Markup Language (GestureML) were proposed in order to formalize touch gesture interactions. We describe in the following the most successful and most recent methods for touch gesture recognition.

Midas (Scholliers et al. 2011) uses a set of logical rules to classify events on the surface. They define sets of temporal and spatial operators to create lists of rules that make up for a gesture definition. With the possibility to define custom operators and priority rules, its gesture definition is extensible to some point, and it allows for interaction gesture recognition, but lacks spatial invariance or robustness to noise.

Proton++ (Kin et al. 2012) is another framework based on regular expressions for gesture recognition: a gesture is seen as a sequence of events represented with a state machine. It allows for continuous control with a callback, although it only supports a unique gesture at a time, and is limited by the rigidity of regular expressions. Multitouch gestures are also hard to define with this framework because of the absence of a finger permutation solution.

Figure 2.1 – Illustration taken from Lü et al. 2012. Touch interface used to input training data to the model. We experiment with the same type of inputs in this thesis although our classification task is defined on different classes.

$Q or Q dollar (Vatavu et al. 2018) is the latest iteration of a touch gesture recognizer for symbolic gestures. This recognizer cast the touch trajectories as a point cloud and computes a dissimilarity metric between the gesture to be recognized and templates of each class. Its computation is greatly optimized, emphasizing on limiting the complexity of the algorithm. This method performs well for well designed gestures but struggles when trajectories get too far from the templates, which can happen in real condition applications.

As efficient and fast as they can be, these methods arbitrarily declare gesture properties, putting hard constraints on the natural variance that can be found between users and conditions. The gestures are precisely defined and tend to lack generalization in a different context; this contradicts our paradigm of minimal user constraint and maximum generalization.

In contrast to these rule-defined gesture frameworks, Rubine 1991 proposed a more flexible gesture definition, learning a classifier from example using hand-crafted geometric features and Linear Discriminant Analysis for classification. Up to our knowledge, this is the first attempt at classifying gestures using Machine Learning (ML). This method however is limited: it can only discretly classify symbolic gestures with single strokes.

Gesture Coder (Lü et al. 2012) takes a conceptually similar approach to Proton++, as it defines gestures using state machines, equivalent to regular expressions on "atomic actions". However, these state machines are learnt from user gestures by successively expanding the state machine to fit examples. When hitting an ambiguity, they use binary decision trees to learn a disambiguation. These trees after training are translated to if-then-else statements. While more flexible than rule-based approaches, Gesture Coder still struggles to cope with natural variance

coming from users and devices. An illustration of the hardware used to record gestures is given in Figure 2.1.

Z. Chen et al. 2014 uses a graph representation of finger state transitions, then embeds the graph structure into a feature vector. These feature vectors are then classified using a Support Vector Machine (SVM). While SVMs allow for a flexible classification of embeded gestures, this method will not be able to handle noisy transitions messing with the embedding (e.g. an accidental touch) and can have trouble with closely related spatial features.

To our knowledge, DL has not been applied to touch gesture recognition. This can be explained by the fact that most classifiers are designed specifically for a device and for specific actions, requiring little adaptation or generalization capabilities. In this case, state machines usually provide a sufficient accuracy while being relatively simple to deploy. In Chapter 3, we describe our work on Deep Learning models, working on general models deployable in most conditions. As a final note, we also recommend Cirelli et al. 2014 as a good survey of the evolution in multitouch recognition.

2.1.3 Continuous Interactions: Interaction Protocols

An important part of our work focuses on a specific component of touch UIs which we call the interaction protocol. This protocol defines the translation from user gestures performed on touch surfaces to continuous actions in virtual environments. In the HCI literature, this interaction protocol refers to a specific part of the software side of an interaction technique (Hinckley et al. 2012, Ortega et al. 2016): the interaction technique is usually coupled with hardware and/or a specific software environment. There are at least as many interaction techniques as there are applications, in consequence we cannot propose an exhaustive list for touch interfaces. We will instead provide examples to illustrate the scientific background and our motivations to automate this process.

In Chapter 4, we address the problem of automatically learning a suitable interaction protocol for GUIs on touch surfaces, which requires users to manipulate 3D objects, for instance in Computer Assisted Design software or in Geographic Information Systems. To give a concrete example, inspecting a virtual mechanical product or navigating in a virtual building or city requires the possibility to change the camera viewpoint through rotations, translations, zooming, i.e. to manipulate 6 degrees of freedom (3 for the camera position and 3 for the camera direction) through trajectories of eventually multiple fingers in the 2D plane of the touch table. In these situations, the problem is particularly ill-posed, as the trajectories produced by a user on the flat touch screen are restricted to a 2D surface, whereas the applications require the user to perform manipulations in a virtual 3D environment. From a naive point of view, we can choose to constrain ourselves to an *active plane* in the 3D environment, defined for instance according

to the camera view and a depth parameter, in order to be able to accurately transpose 2D actions into 3D actions. This method is however tedious for the user and is therefore not suitable for a time efficient, satisfying interface.

In Ropinski et al. 2005, they discuss a navigation technique for travelling in 3D city models. The "interaction panel" in this publication is taken from Szalavári et al. 1997. Another 3D navigation UI is described in Tan et al. 2001 using either a mouse or a touch-pad. These advanced methods approach this mapping from gestures to actions in a 3D environment using several parameters: not only the 2D gesture themselves, but also the position of the camera (view of the user), the state of the 3D environment, or any information perceived by the user (D. Bowman et al. 2006; Cashion et al. 2012). Theoretically, these methods offer more complex manipulation strategies and higher efficiency. However, the challenge here lies in the combination of precision and efficiency on one hand, and ease of use and learnability (by humans) on the other hand. While all these UIs are efficient in a specific context, there is no universally accepted canonical solution for this kind of problem.

2.2 Supervised Deep Learning for Classification on Sequential Data

We describe in this section the theoretical background behind the DL algorithms used throughout this thesis and state-of-the-art related work.

Deep Learning architectures as of now can be extremely large and complicated, combining a multitude of different mechanisms and learning signals, with a variety of regularization methods. There is however some common ground for every DL architecture:

- they are trained to maximize or minimize an objective function J with respect to their learnable weights.
- Every architecture is trained using variants of backpropagation. The error gradient can come from many different sources, but the update mechanism is always the same.
- there is a limited number of methods to structure the network. As of now, connections can be built with fully connected, convolutional, recurrent or attention layers.

In a supervised learning context, the task for DL models is to learn to attribute the correct discrete or continuous label to observed data. This correct label is set by human supervision, hence the name. The learning problem can be seen as minimizing a loss function expressed as the error between the predicted label and the correct one, or ground truth.

The phase consisting in tweaking the weights of the architecture from observed data and labels (the training set) is called the training phase. The true goal of this training phase is not for the model to perfectly score on this training set. The goal is to score well on data not seen during training, without supervised labels. This true performance is measured on a test set (or before on a validation set, depending on the protocol). The core difficulty and interest of DL comes from the notion of generalization: how well can the model perform on data never seen during training? How much noise or variance can it sustain before failing? This is the motivation behind the complexity and the variety of architectures in the literature.

2.2.1 Fully Connected Networks and Backpropagation

A neuron is a function parametrized by a set of learnable weights $W = \{w_i\}_i$ and a learnable bias b, where $w_i, b \in \mathbb{R}$. The neuron receives a vector of inputs $X = \{x_i\}_i$ and produces an activation a such as:

$$\begin{aligned} z &= \sum_i w_i x_i + b\,, \\ a &= f(z)\,, \end{aligned} \tag{2.1}$$

where f is called an activation function. Its role is to add non-linearity inside Neural Networks (NNs). This function is usually a tanh or a ReLU (V. Nair et al. 2010). We can note that without an activation function, a neuron is simply a linear binary classifier. A layer is a set of an arbitrary number of neurons processing the same input and getting gradient from the same layer or output. The type of layer will depend on the constraints between weights and how the outputs of the neurons are handled.

The simplest form of layer is the Fully Connected (FC) layer: a set of j neurons as described above produces a vector of outputs $a = \{a_j\}_j$. The length of the vector a is usually referred to as the size of the layer. A NN is a stacking of K layers, with the k-th layer taking the activations of the $(k-1)$-th layer as input. In a complete Fully Connected Network (FCN), to avoid confusion, we can write the weights as $w^k_{i,j}$ and biases as b^k_j.

If we work on a binary classification task, one of the simplest architectures to try to solve this problem would be a NN with one hidden layer and an output layer with as much neurons as there are classes. This output layer produces a confidence score $\hat{y}$ traditionnally called logits for each class. We then usually produce a probability vector over the labels by applying the softmax function on the logits. Passing data into the network to produce logits is referred to as the forward pass. An illustration of such a network is given in Figure 2.2 for an hidden layer of size 4, an input size of 3 and 3 classes.

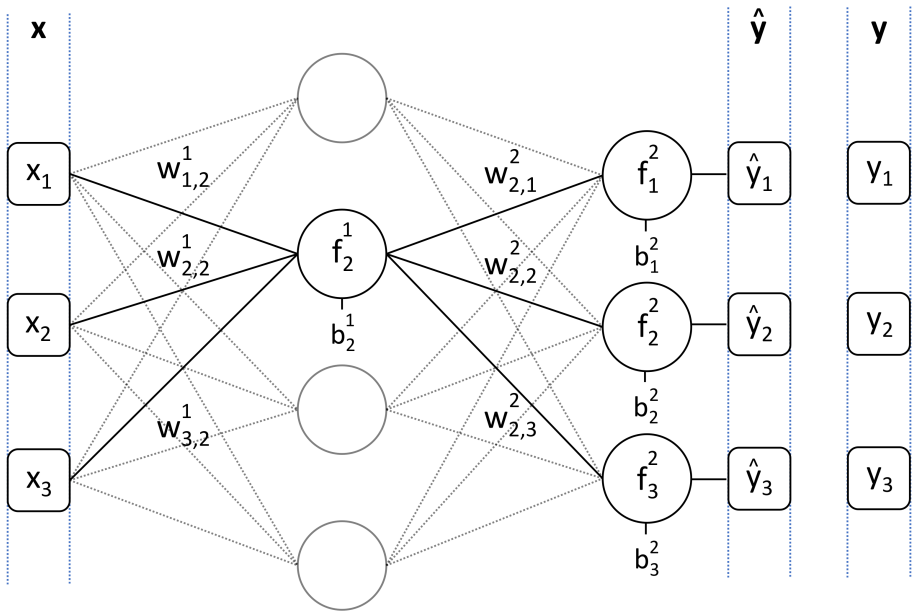

Figure 2.2 – Graphical representation of a Fully Connected Neural Network with one hidden layer of size 4, observing input x and outputting logits $\hat{y}$. We highlight the weights w and biases b related to the forward pass passing through the second neuron of the hidden layer. Scalars are represented as squares, neurons as circles.

Now that we have defined an architecture, we will describe the principles of backpropagation. Once a prediction $\hat{y}$ is produced, we want to compare it to the true label y. We can define an objective function J measuring an error as the distance (not necessarily in the mathematical sense) between $\hat{y}$ and y. The goal of the NN will be to minimize this function. In the case of function minimization, the objective function is usually referred to as a loss function. While the mean squared error can be seen as the most intuitive loss function, we tend to represent y as a one-hot vector and compute the cross-entropy between $\hat{y}$ and y. Among the justifications, we can cite a better numerical stability and a better behavior when close to convergence. The goal of backpropagation is then to minimize the loss by iteratively updating the weights and biases of the network. In other words, the goal is to find the parametrization of the NN that attains a global minimum of J. In practice, because of the complexity of cost functions, we aim for "good" local minima. The quality of a minimum is often measured through complementary metrics such as accuracy. One of the simplest way to do so is to use gradient descent. If we omit indices, a gradient descent update is:

$$\begin{aligned} w &\leftarrow w - \alpha \frac{\partial J}{\partial w}, \\ b &\leftarrow b - \alpha \frac{\partial J}{\partial b}, \end{aligned} \tag{2.2}$$

where α is a positive scalar called the learning rate. With enough iterations for each weight and bias of a neural network (where J is recalculated at each iteration), we ensure that the NN finds at least a local minimum in the cost function J. About the learning rate, it is manually fixed and controls how sensitive the updates are in the gradient direction: fixing it too high (close to 1) might result in gradient descent missing the local minimum, whereas setting it too low might trap consecutive updates in a bad local minimum. In order to smoothen this update, the stochastic gradient descent variant is preferred, computing the partial derivatives as the mean of an arbitrary number of partial derivative samples. This number of samples is called batch size in the ML literature. A lot of different methods propose to dampen the defaults of vanilla gradient descent: AdaGrad (Duchi et al. 2011), RMSProp (Hinton 2012), ADAM (Kingma et al. 2014) and AMSGrad (Reddi et al. 2018), to cite some of the most used, usually inject second order information to improve the convergence speed and robustness. While more complex optimization algorithms could be used, the size of current neural networks requires this optimization algorithm to be extremly fast for the network to be trained in an acceptable duration. As of now, simple differentiable methods thanks to backpropagation are the golden standard considering the trade-off between convergence speed and quality of the solution.

To simplify the following equations, we consider that the activation function f is the same for all neurons. We write $f'(z)$ the derivative of f with respect to z. In order to compute the partial derivative for each weight and bias in a distributed fashion, backpropagation uses the chain rule:

$$\frac{\partial J}{\partial w_{i,j}^k} = \frac{\partial J}{\partial z_j^k}\frac{\partial z_j^k}{\partial w_{i,j}^k} = \frac{\partial J}{\partial z_j^k}a_i^{k-1}. \tag{2.3}$$

The definition of a and z is given in Equation 2.1. Thanks to this notation, the gradient of every weight (and bias) of a neuron can be easily computed from the local gradient $\frac{\partial J}{\partial z_j^k}$. Moreover, this local gradient can be iteratively computed from the upper local gradients, hence the name backpropagation. If we consider indices i and j from the $(k+1)$-th layer, meaning i is the indice on the k-th layer and j the indice on the $(k+1)$-th layer:

$$\frac{\partial J}{\partial z_i^k} = \frac{\partial J}{\partial a_i^k}\frac{\partial a_i^k}{\partial z_i^k}, \tag{2.4}$$

where

$$\frac{\partial J}{\partial a_i^k} = \sum_j \frac{\partial J}{\partial z_j^{k+1}}\frac{\partial z_j^{k+1}}{\partial a_i^k} = \sum_j \frac{\partial J}{\partial z_j^{k+1}}w_{i,j}^{k+1}, \tag{2.5}$$

and

$$\frac{\partial a_i^k}{\partial z_i^k} = f'(z_i^k). \tag{2.6}$$

Thus:

$$\frac{\partial J}{\partial z_i^k} = \sum_j \frac{\partial J}{\partial z_j^{k+1}} w_{i,j}^{k+1} f'(z_i^k). \tag{2.7}$$

In the end, the only derivatives we have to compute are the derivatives of the loss function J with respect to the output layer and $f'(z_i^k)$ for each neuron, and the latter can be pre-computed during the forward pass.

While theoretically powerful approximators, FCNs tend to behave poorly when confronted to new data. The absence of regularization combined with the limited information coming from backpropagation can cause the network to learn poorly generalizable features such as noise or occurence-related features. This causes the network to succeed in solving a training set, but yields very poor results on a test set. This problem is known as overfitting. In order to overcome it, two leads are mainly explored: regularize the weights in the network to enforce more general features (e.g. Dropout, BatchNorm), or constrain the weights and the structure of the network to produce specific features. Convolutional Neural Networks (CNNs), Recurrent Neural Networkss (RNNs) and Attention mechanisms are successful examples of such constraints.

2.2.2 Convolutional Neural Networks

CNNs are one of the most used architectures in DL because of their versatility and effectiveness in a wide range of problems. They consist principally in a succession of convolutional layers. A convolutional layer learns the weights of an arbitrary number of convolution operations between the input and convolution kernels. The kernel parameters are the learnable weights. Each convolution operation on the input produces a modified version of the input called feature map. In computer vision, before Deep Learning popularity, such handcrafted kernels were largely used to exhibit specific properties in an image such as Canny, Sobel or Gabor filters. Fukushima 1980 with the Neocognitron designed first an architecture of cascading learnable convolution filters, but this is with Lecun et al. 1998 that current CNNs are defined, using backpropagation to train a succession of convolutional and subsampling layers ending with fully connected layers.

There are a few parameters commonly tweaked in a convolutional layer:

- the number of kernels/feature maps per layer,
- the size of the kernels,
- eventual padding of the input to alter the size of the resulting feature maps,
- the stride of the convolution, meaning the spacing between two consecutive local operations of the convolution (the mathematical convolution operation has a stride of 1 on every dimension),

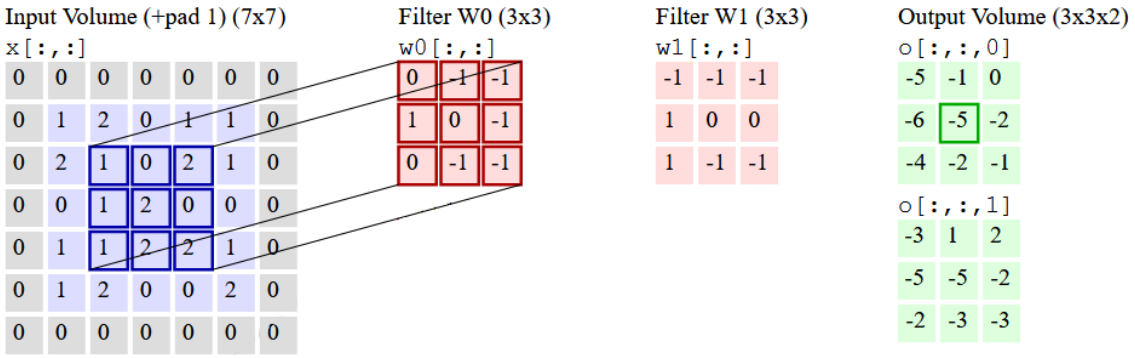

Figure 2.3 – Illustration modified from the online Standford CS class CS231n Convolutional Neural Networks for Visual Recognition. This is a graphical representation of a convolutional layer with two kernels of size 3x3 and stride 2x2, observing an image of size 5x5 padded to size 7x7. This convolutional layer produces two feature maps of size 3x3.

- and the activation function of the kernels.

If we want to conceptually compare a convolutional layer with a fully connected layer, we can imagine a convolutional layer as a fully connected layer with periodically constrained weights: let us consider a feature map made with a kernel of size 3x3, a stride 2x2 and flatten it (concatenate each row to form a vector). Such a vector would be equivalent to the activations of a fully connected layer of the same size with periodically constrained weights: instead of having a weight for each connection between the input and the activation, there would only be 9 weights reused in multiple locations. CNNs are less prone to overfitting than FCNs thanks to these constraints put on weights: convolutional features retain specific patterns more specific than unconstrained relations and as such generalize better. An illustration of a simple convolutional layer is given in Figure 2.3.

CNNs combine multiple convolutional layers in order to build hierarchical hidden representations with increasing levels of abstraction in each subsequent layer. The next layer processes the feature maps of the preceding layer, producing new feature maps from local representations built in the preceding feature maps. We can note that using convolutions add shift invariance as inductive bias to the network. Because convolutions aggregate information from a local region, the receptive field of kernels becomes larger with respect to the original input with each new layer. Once the receptive field covers the entire input, all the data has been processed and we can use a fully connected layer to produce the logits from a feature vector made from (potentially flattened) high level feature maps. In the case that the last feature maps are already reduced to scalars, flattening is not necessary: our feature vector will be the concatenation of these feature maps.

While used for two decades on static data (mostly images), CNNs have been recently reported to be also efficient for learning hierarchical features on sequential data. Even though RNNs are specifically designed to model temporal dependencies (described in the next subsection), CNNs proved to be competitive, and in some cases better performing than RNNs. With CNNs, the temporal dependencies of the data in this case are not covered by a unique hidden state (as with RNNs), but by a collection of feature maps, which capture temporal correlations of the data at an increasing level of abstraction. While the first filters capture short term behavior, the last layers capture long range dependencies. We can find the idea of learning spatio-temporal convolutional features in Taylor et al. 2010, but it was applied at that time to Restricted Boltzmann Machines.

In Karpathy et al. 2014, they experiment with different time fusion models on action recognition in videos from the dataset Sports-1M and UCF-101. The network is split into a multi-resolution architecture with two parts, each part processing the input at a different scale. The high-resolution part only observes the cropped center of the video to reduce computation.

At around the same time, Simonyan et al. 2014 propose an alternative strategy for action recognition in videos also using CNNs. One CNN is used for recognition in still frame, building exclusively spatial features, whereas another is fed a sequence of multi-frame optical flow, able to build temporal features from stacked optical flow frames. Their activation is then fused, using a SVM in their best performing variant. It is interesting to note that using only the temporal CNN already yields way better results on UCF-101 than the method described in Karpathy et al. 2014. From this observation, we can make the assumption that for complex problems, reducing noise and variance in the data by selecting a good representation really helps the convergence of CNNs. This strategy of injecting prior knowledge over the data to reduce the workload on neural networks for specific problems is extensively used in the literature.

An interesting use of CNNs is proposed in Gehring et al. 2017, where a complex variant of CNN was successfully used for sequence-to-sequence classification in a machine translation task. In this paper, they adapt an encoder-decoder model with attention Bahdanau et al. 2014, using CNNs instead of RNNs to build hidden representations and decode them, and they also modify the attention mechanism to take into account preceding attention steps. This convolutional model outperformed the state-of-the-art Long Short-Term Memory (LSTM)-based algorithm, Google's Neural Machine Translation (Yonghui Wu et al. 2016).

Considering haptic data for human-robot interaction, Albawi et al. 2018 use CNNs to classify social touch gestures from raw sensor data. More recently, Park et al. 2019 also process raw sensor data directly using a 1D-CNN to classify four different touch patterns performed on haptic sensors.

2.2.3 Recurrent Neural Networks

RNNs are a standard in DL whenever we are manipulating sequential data. Their purpose is to identify patterns and dependences in a sequence using an internal memory of past timesteps. For the following RNN equations, we will use a matrix representation of the weights. If we consider the definition of z in Equation 2.1 and want to write all the weights of a layer with n neurons and m inputs:

$$\begin{aligned} z_1 &= \sum_i w_{1,i} x_i + b_1, \\ z_2 &= \sum_i w_{2,i} x_i + b_2, \\ &\dots \\ z_n &= \sum_i w_{n,i} x_i + b_n. \end{aligned} \tag{2.8}$$

we can rewrite this equation system in matrix form:

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,m} \\ \vdots & \vdots & & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}. \tag{2.9}$$

This can simply be written:

$$z = Wx + b, \tag{2.10}$$

where W is the weight matrix and b the bias vector.

We consider sequential data $x = \{x_t\}_{t \in \{0,\dots,T\}}$. We remind that x_t is a vector. An RNN builds recurrent internal representations called a hidden state h_t, and produces an output y_t at each timestep t. This hidden state can be of arbitrary length and the output y_t is usually computed as a linear combination of weights and h_t. is a vector of the same size as y_t. If W, U and V are the cell weights and b_h, b_y bias vectors, the vanilla version can be formulated as:

$$\begin{aligned} h_t &= \tanh(W.h_{t-1} + U.x_t + b_h), \\ y_t &= V.h_t + b_y, \end{aligned} \tag{2.11}$$

where weights U process the current input, W the preceding hidden state and V the current hidden state. We can see the recursive definition of h_t depending on its preceding value. In a classification task, y_t is usually producing logits, activated using the softmax function to produce a probability vector over the labels. In some contexts, RNN layers may directly output h_t, declaring $y_t = h_t$.

Recurrent models trained using a variant of backpropagation called BackPropagation Through Time (BPTT). This is simply backpropagation also propagating the error from the current timestep to the preceding hidden state (from the W weights). This temporal term unrolls all timesteps to retropropagate the error recursively. In theory, the hidden state can store information from any point in the sequence and handle temporal dependencies of any scale, supposing the hidden state is large enough. In practice, the temporal term in BPTT causes numerical instabilities leading to vanishing or exploding gradients (Bengio et al. 1994).

LSTMs (Hochreiter et al. 1997) are a widely used variant of vanilla RNNs that aims to correct this issue. It adds a memory cell c to retain information, and use a gating system to read, write and forget from it. Gates act as a selection mechanism for key operations on the cell memory: the forget gate f selects the information to be kept from the older memory, the input gate i selects the information to be added from the temporary state $\tilde{c}$ and the output gate o selects the information to be outputed from c into h. Let σ be the sigmoid function, $\odot$ the Hadamard product, an LSTM cell can be formulated as:

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot h_{t-1} + U_f \cdot x_t + b_f), \\
i_t &= \sigma(W_i \cdot h_{t-1} + U_i \cdot x_t + b_i), \\
o_t &= \sigma(W_o \cdot h_{t-1} + U_o \cdot x_t + b_o), \\
\\
\tilde{c}_t &= \tanh(W_{\tilde{c}} \cdot h_{t-1} + U_{\tilde{c}} \cdot x_t + b_{\tilde{c}}), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
\\
h_t &= o_t \odot \tanh(c_t), \\
y_t &= h_t.
\end{aligned}
\tag{2.12}
$$

Whenever we refer to an LSTM cell, we will refer to this model. The separation of c and h is debatable; some LSTM variants fuse them, we will describe one shortly after. An LSTM cell of size X means that its memory cell c and hidden state h are vectors of size X. In consequence, the gate vectors and the output are also of size X.

Lastly, we will describe the Gated Recurrent Unit (GRU) (Cho et al. 2014), a lighter version of LSTM with comparable performance throughout the literature. GRUs simplify the gating principle of LSTMs and fuse the memory cell and hidden state back. It uses two gates: a reset gate r filtering the information to be kept from the older hidden state, and an update gate z doing a linear combination of the older hidden state and new information to produce the new hidden state. Formally:

$$
\begin{aligned}
r_t &= \sigma(W_r \cdot h_{t-1} + U_r \cdot x_t + b_r), \\
z_t &= \sigma(W_z \cdot h_{t-1} + U_z \cdot x_t + b_z), \\
\\
\tilde{h}_t &= \sigma(W_h \cdot (r_t \odot h_{t-1}) + U_h \cdot x_t + b_h), \\
\\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t, \\
y_t &= h_t.
\end{aligned}
\tag{2.13}
$$

All these models presented are recurrent in one direction. This means that if we are working on structured data of higher dimension (RGB images or multitouch gestures for example), the other dimensions need to be stacked in order to be treated as one "big" input vector. In the case of touch gesture for example, this means stacking the (x, y) information of all fingers at a time to create a tensor for each timestep, then the RNN processes this tensor at each timestep. This prevents the model from searching for sequential dependences parametrized by finger IDs.

2.2.4 Multi-Dimensional Recurrent Neural Networks

When the observed data is multidimensional, building recurrent architectures can be problematic. What if there are sequential dependencies to learn from different dimensions, say spatial and temporal dependencies? One possible strategy is to use Multi-Dimensional Recurrent Neural Networks (MDRNNs) or Multi-Dimensional Long Short-Term Memorys (MDLSTMs) (Graves et al. 2007), a straightforward extension of the 1D original models. There are also closely related variants like GridRNNs (Kalchbrenner et al. 2016).

In these models, the hidden state at some point during recurrence is not only computed from one single predecessor $(t-1)$ but from a predecessor for each dimension. In order to extend LSTMs to N dimensions, Graves et al. 2007 adds a forget gate for each dimension to control the information coming from each dimension. Omitting the gate indices, the equations for a 2D-LSTM are:

$$
\begin{aligned}
f^1 &= \sigma(W_{f^1} \cdot h_{t-1,i} + V_{f^1} \cdot h_{t,i-1} + U_{f^1} \cdot x_{t,i} + b_{f^1}),\\
f^2 &= \sigma(W_{f^2} \cdot h_{t-1,i} + V_{f^2} \cdot h_{t,i-1} + U_{f^2} \cdot x_{t,i} + b_{f^2}),\\
i &= \sigma(W_i \cdot h_{t-1,i} + V_i \cdot h_{t,i-1} + U_i \cdot x_{t,i} + b_i),\\
o &= \sigma(W_o \cdot h_{t-1,i} + V_o \cdot h_{t,i-1} + U_o \cdot x_{t,i} + b_o),\\
\\
\tilde{c}_{t,i} &= \phi(W_{\tilde{c}} \cdot h_{t-1,i} + V_{\tilde{c}} \cdot h_{t,i-1} + U_{\tilde{c}} \cdot x_{t,i} + b),\\
c_{t,i} &= f^1 \odot c_{t-1,i} + f^2 \odot c_{t,i-1} + i \odot \tilde{c}_{t,i},\\
\\
h_{t,i} &= o \odot \phi(c_{t,i}),\\
y_{t,i} &= h_{t,i}.
\end{aligned}
\tag{2.14}
$$

This model can handle sequences with variable length in every dimension: this allows for more flexibility than CNNs that require fixed length inputs. For some experiments in Chapter 3, we use a more robust variant called Spatio-Temporal LSTM (Liu et al. 2016). This variant adds a "trust-gate" which filters the input in order to compensate for noise. The intuition behind it is to reduce the effect of input too different from what the cell expects. The sensitivity of the trust gate is manually set using a parameter $\lambda \in [0, 1]$. It uses the same equations as a 2D-LSTM but applies the following trust gate τ to the $i \odot \tilde{c}_{t,i}$ product:

$$
\begin{aligned}
\tilde{x} &= \tanh(W_{\tilde{x}} \cdot x_t + b_{\tilde{x}}),\\
\hat{x} &= \tanh(W_{\hat{x}} \cdot h_{t-1,i} + V_{\hat{x}} \cdot h_{t,i-1} + b_{\hat{x}}),\\
\\
\tau &= \exp\left(-\lambda \cdot (\tilde{x} - \hat{x})^2\right).
\end{aligned}
\tag{2.15}
$$

These MDLSTMs solve one of the problems of 1D LSTMs and can scale to an arbitrarily large input in both dimensions, but requires a large number of weights in order to do so. This statement motivated us to find a lighter version of MDLSTMs by extending GRUs to multiple dimensions. We further enhance this architecture by combining convolutional and recurrent features in a novel fasion. This new architecture is defined and tested in Chapter 3.

While attention mechanisms and Transformer-based networks (Vaswani et al. 2017) are not used in this thesis, they should be mentioned when describing DL for sequential data. A large number of architectures derivated from Transformer (Devlin et al. 2019; Radford 2018) are currently state-of-the-art on a multitude of Natural Language Processing tasks.

2.3 Reinforcement Learning for Continuous Control

Before getting into the core subject, we will summarize in this section the theoretical background of Reinforcement Learning (RL) used in Chapter 4 and Chapter 6. This will allow us to justify our choices and to properly explain the formalization of the problem.

RL is a Machine Learning paradigm used to learn policies in order to solve control problems in an environment. More precisely, a RL agent learns a policy π that observes at each timestep a state s_t of the environment and takes actions a_t in this environment. This action modifies the state of the environment (s_{t+1}) and the agent receives in consequence a reward $r_{t+1} = f(s_t, s_{t+1}, a_t)$ measuring the quality of the action taken. This function f is handcrafted to reward successful behaviors in the environment. An optimal policy is learned through an optimization process by maximizing the expectation of the cumulative sum of rewards or return G_t. Let J be the objective function and θ a parametrization of π. At any time t, the optimization problem can be formulated as:

$$\begin{aligned} &\arg\max_{\theta} J = \arg\max_{\theta}(\mathbb{E}[G_t]), \\ &G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots, \end{aligned} \tag{2.16}$$

where $\gamma \in [0, 1]$ is the discount factor. This factor was added to cope with continuous tasks, where there is an infinite number of timesteps: in this case, the return can become infinite. Setting γ below 1 ensures the convergence of G_t. On episodic tasks, i.e. with a finite number of timesteps, a policy should theoretically optimize on the discounted return with an infinite horizon ($\gamma = 1$). However, experimental results when using NNs as a policy or a value estimator show that it is better set inferior to 1 (Jiang et al. 2016). In consequence, we will consider any return to be discounted to simplify the writing.

This optimal policy problem is modelled using the probabilistic framework of Markov Decision Processes (MDPs). Simply put, we cast the policy as a probability distribution $\pi_\theta(a_t|s_t)$. In this case, we consider that the dynamics of our system p are entirely determined by the unknown transition matrix:

$$p(s'|s) = Pr[s_{t+1} = s'|s_t = s]. \tag{2.17}$$

Finding an optimal policy in these conditions is known in the literature as solving a stochastic optimal control problem. Historically, RL solves this problem using variants of two strategies: policy-based methods, focusing on directly finding a good behavior from sampled trajectories, and value-based methods focusing on the evaluation of observed states and actions.

2.3.1 Policy gradient

If we think of the policy π_θ as a parametric model trainable using backpropagation, e.g. a neural network, the first intuition to solve the optimization problem in Equation 2.16 is to find a computable expression of the gradient $\nabla_\theta J(\theta)$ in order to update the policy parameters θ using gradient ascent:

$$\theta^{i+1} = \theta^i + \alpha \nabla_\theta J(\theta^i), \tag{2.18}$$

where α is the learning rate for the update. The Policy Gradient Theorem (Sutton et al. 1998) allows us to express the gradient as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[Q(s,a)\nabla_\theta \log \pi_\theta(s)], \tag{2.19}$$

where $Q(s,a) = \mathbb{E}[G_t]$ is the state-action value function, measuring the quality of an action a taken during state s. Learning to estimate this Q-value will be our main concern with Q-learning algorithms, but for policy gradient we prefer to directly estimate the Q-value using sampled rewards. Now, let us consider a trajectory τ followed by the policy, represented as an ordered sequence of T triplets (s,a,r). In this case, the policy is following the sample path $Pr[\pi_\theta(a_0|s_0) = a, ..., \pi_\theta(a_T|s_T) = a^{(T)}]$ noted $\pi_\theta(\tau)$. This sample path depends on the unknown transition matrix p and can be developped as:

$$\pi_\theta(\tau) = p_0(s_0) \prod_{t=1}^{T} (\pi_\theta(a_t|s_t) p(s_{t+1}|s_t)), \tag{2.20}$$

where p_0 is the distribution initializing the state of the environment. Because p and p_0 are independent from θ, we can write the gradient of $\log \pi_\theta(\tau)$ as:

$$\nabla_\theta \log \pi_\theta(\tau) = \sum_{t=1}^{T} (\nabla_\theta \log \pi_\theta(a_t|s_t)). \tag{2.21}$$

This result is important: it means that we can compute the gradient of the policy along the whole trajectory independently from environment dynamics.

Now, we can rewrite the Policy Gradient Theorem applied to the trajectory τ. Lets approximate $Q(s_t, a_t)$ with the sampled G_t from τ. Injecting Equation 2.21 in Equation 2.19 gives us:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=1}^{T} (G_t \nabla_\theta \log \pi_\theta(a_t|s_t))]. \tag{2.22}$$

The last thing to do to explicit this gradient is estimate the expectation. A simple way to do so is to approximate its value by sampling a large number of trajectories and average the values. The gradient is computed along the whole trajectory,

applying a Markov Chain Monte-Carlo method, and is backpropagated through π_θ using stochastic gradient ascent. Such gradient will enforce policies following trajectories with high return (not necessarily individual actions with high rewards) while moving away from low return trajectories. This training algorithm is known as the REINFORCE algorithm (Williams 1992), and is the simplest form of RL using stochastic gradient ascent for policy optimization. In this case, the policy is stochastic, meaning it produces a vector of probabilities over the possible actions; we can sample it in order to decide which action to take.

2.3.2 Q-learning

We present here the precursor of most value-based methods in RL as of today. We retain all notations presented earlier in this section.

By taking an action a while observing state s, all the possible trajectories afterward can only expect at most a certain return. Taking a different action (i.e. following a different policy) during state s might give a different maximum expected return for the next state. This value, depending on the action taken, can be seen as a measure of how good an action is depending on the observed state. It is formally known as the state-action value function $Q^*(s,a) = \max_\pi \mathbb{E}_\pi[G_t|s_t = s, a_t = a]$. Knowing this function means solving the control problem, and the optimal policy π^* in this case can be defined as $a^* = \arg\max_a Q^*(s,a)$. Contrarily to policy gradient that consider sampled returns as unbiased samples for the Q-value, Q-learning focuses on learning an estimate of this optimal Q-value in order to select the optimal action.

The Bellman equation is an important tool to compute the Q-value:

$$Q^*(s_t,a_t) = \mathbb{E}_{s_{t+1}}[r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1},a_{t+1})|s_t,a_t]. \tag{2.23}$$

In the case of discrete states and actions, it allows us to describe the Q-function in a recursive fashion and compute an estimate of the Q-value iteratively:

$$Q^{i+1}(s_t,a_t) = \mathbb{E}_{s_{t+1}}[r_{t+1} + \gamma \max_{a_{t+1}} Q^i(s_{t+1},a_{t+1})|s_t,a_t]. \tag{2.24}$$

Thanks to the Bellman equation, we can solve our problem using a Temporal Difference (TD) method, directly using two consecutive instants to update the estimate rather than needing a full trajectory. It means that once an action is taken and a reward is given, we can look back at the Q-estimate used to perform the action and correct it. If used as is for value iteration, without a Q-function estimator, the quality of the solution is however limited: it is equivalent to learning a look-up table of Q-values independentely from each state, which is not desirable for generalization purposes, and when the environment can display a large number of states.

A better way of solving the control problem is to use a Q-function approximator $Q_\phi(s,a) \approx Q^*(s,a)$ parametrized by ϕ. In practice, this approximator is a neural network. Using Equation 2.24 to write the target of $Q_\phi(s,a)$ as $y = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_{a_{t+1}} Q_\phi(s_{t+1}, a_{t+1}) | s_t, a_t]$, we can express a loss function for our estimator, updated at each timestep:

$$L(\phi) = \mathbb{E}_{s_t,a_t}[(y - Q_\phi(s_t,a_t))^2]. \tag{2.25}$$

It is interesting to note that the target – as with policy gradient – depends on the parametrization we want to optimize. This loss can easily be differentiated with respect to the weights and, ignoring the factor 2, we end up with the gradient:

$$\nabla_\phi L(\phi) = \mathbb{E}_{s_t,a_t}\Big[\Big(r_t + \gamma \max_{a_{t+1}} Q_\phi(s_{t+1}, a_{t+1}) - Q_\phi(s_t,a_t)\Big)\nabla_\phi Q_\phi(s_t,a_t)\Big]. \tag{2.26}$$

Instead of computing the expectation, we will sample multiple transitions in order to approximate it and bootstrap the Q-value, performing stochastic gradient descent to update the weights of $Q_\phi(s,a)$. This is known as the Q-learning algorithm (Watkins et al. 1992).

So, in short, Q-learning tries to learn an estimation of the value of the states and the actions taken by a deterministic policy, whereas policy gradient tries to directly solve the control problem by learning the policy. Both strategies have advantages and drawbacks, to name a few:

- deterministic policies in noisy environments will have trouble finding states with good values. We need to inject some noise in the decision, e.g. follow an ϵ-greedy policy during training to enforce exploration.
- TD methods such as Q-learning tend to converge faster than simple policy gradient methods.
- The stochastic policy of REINFORCE naturally allows for (approximate) continuous control by sampling from a multivariate gaussian with means computed by the policy.

These methods however are not exclusive: we can intuitively picture a policy-based algorithm using an estimate of the Q-value using TD. This is precisely what Actor-Critic methods do.

2.3.3 Actor-Critic

Actor-Critic methods combine the advantages of both policy gradient and value-based methods. Once REINFORCE and Q-learning are described, the definition of vanilla Q Actor-Critic (Degris et al. 2012) is straightforward: let π_θ be a differentiable policy parametrized by θ and $Q_\phi(s,a)$ a differentiable

estimator of $Q^*(s,a)$ parametrized by ϕ. We will respectively call these models the actor and the critic. If we take the policy gradient of REINFORCE defined in Equation 2.22 and replace the sampled G_t with $Q_\phi(s_t,a_t)$, we can bootstrap and produce a gradient timestep by timestep for the actor : $Q_\phi(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)$. The gradient of the critic was already expressed in Equation 2.26 with the Q-learning algorithm. Thus, training the actor and the critic together becomes in pseudo-code:

Algorithm 1: Q Actor-Critic

Initialize s, θ, ϕ and learning rates α_θ and α_ϕ
Sample $a_1 \sim \pi_\theta(a|s = s_1)$
for $t = 1, \dots, T$ **do**
  Receive reward r_{t+1} and next state s_{t+1}
  Sample next action $a_{t+1} \sim \pi_\theta(a|s = s_{t+1})$
  Update the actor: $\theta^{i+1} = \theta^i + \alpha_\theta Q_\phi(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)$
  Update the critic: $\phi^{i+1} = \phi^i + \alpha_\phi \nabla_\phi L(\phi)$ (Equation 2.26)
end for

We can see that both models have non-stationary targets. This makes this nonconvex optimization process complex (NP-hard at worst), even more so when using NNs as actor and critic, and convergence is not assured. A great deal of papers propose empirical methods to stabilize the optimization process and quicken the convergence speed of Actor-Critic RL agents (Lillicrap et al. 2015; Mnih et al. 2016; Yuhuai Wu et al. 2017; Haarnoja et al. 2018).

2.3.4 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2015) is the algorithm we use for our RL agents in Chapter 4 and Chapter 6. Because we do not contribute in this thesis in the training process of the policy, we will perform an overview of DDPG, noting the differences with Q Actor-Critic, rather than an in-depth description. DDPG is an actor-critic method combining training stabilization tricks from Deep Q-Network (Mnih et al. 2013) and Deterministic Policy Gradient algorithms (Silver et al. 2014). DDPG allows for off-policy continuous control and can be extended to a Partially Observable Markov Decision Process (POMDP) modelization.

The first addition compared to Q Actor-Critic is the use of a replay buffer to store rollouts in the environment. That means keeping in memory the quadruplets (s,a,r,s') visited during the rollout phase. The training phase of the actor and the critic will be done by randomly sampling batches of quadruplets from the replay memory. This allows us to break noisy correlations from trajectories and

consider every sample independently from each other (desirable with our MDP modelization).

The second addition is the use of target networks μ' and Q' to update the actor and the critic to perform "soft updates". This means that target networks are slowly moved in the direction of the current updated networks instead of being directly updated.

The third addition is that the policy gradient is computed a bit differently in order to take into account the deterministic policy μ. If we modify the Actor-Critic algorithm in Algorithm 1, with samples taken from the replay memory, the network updates become:

$$\begin{aligned}
\theta^{i+1} &= \theta^i + \alpha_\theta \nabla_a Q_\phi(s_t, a_t) \nabla_\theta \mu_\theta(s_t)|_{a=\mu_\theta(s)}, \\
\phi^{i+1} &= \phi^i + \alpha_\phi \nabla_\phi L(\phi), \\
\\
\theta_T^{i+1} &= \tau\theta^{i+1} + (1-\tau)\theta_T^i, \\
\phi_T^{i+1} &= \tau\phi^{i+1} + (1-\tau)\phi_T^i,
\end{aligned} \tag{2.27}$$

where

$$\nabla_\phi L(\phi) = \mathbb{E}_{s_t}\Big[\Big(r_t + \gamma Q'_{\phi_T}(s_{t+1}, \mu'_{\theta_T}(s_{t+1}))Q_\phi(s_t, a_t)\Big)\nabla_\phi Q_\phi(s_t, a_t)\Big].$$

The factor τ is set largely inferior to 1, and will modulate the importance of the updates on the target networks.

2.4 Artificial Intelligence and Human-Computer Interfaces

Our work in this thesis stands between two active fields of computer science: HCI and ML. While both these fields have been active for more than 50 years, their interaction is still relatively recent; we will paint an overview of the current uses of ML in HCI in this section.

Adaptive User Interface — To quote Langley 1997, "An Adaptive User Interface (AUI) is an interactive software system that improves its ability to interact with a user based on partial experience with that user". In practice as of now, this adaptability is still limited to the visual display of pertinent information to the user, depending on a request or preferences. ML in this case is traditionally used for user profile modelization. To the best of our knowledge, Pazzani et al. 1996 is the first proposal of ML for an AUI. In this paper, the authors learn user profiles using a bayesian classifier in order to display webpages of interest. This is around the same time that PageRank is developped (the ranking algorithm behind

Google, published in (Brin et al. 1998)). As of today, most "pertinence" algorithms use ML. For an overview of state-of-the-art AUI with adaptive visualization, see Alvarez-Cortes et al. 2007.

Reinforcement Learning and UI design — RL is a machine learning framework in which a software agent learns to solve an environment by taking actions that maximizes some cumulative reward (cf Section 2.3). RL has seen some specific uses for AUI design, more precisely for user profiling and representation tasks.

- In Seo et al. 2000, a RL agent learns to detect user preferences implicitly from observing user behavior instead of direct feedback. This agent adapts user's profile by maximizing the number of times it properly selects an hyperlink chosen by the user.

- In Ferretti et al. 2014, a RL agent uses user feedback to learn a user profile for personalized web page display. Each element of the UI (font size, colors...) is affected a value. This value can be updated by a RL agent with respect to the number of times a user chose or discarded an item in the UI.

- X. Chen et al. 2015 is a good example of potential GUI design automation using RL. The goal of this paper is to model user behavior in menu search. Building upon Bailly et al. 2013, this modelization can allow for the testing of newly designed interfaces or to predict rational strategies in aimed movements. An illustration of the setup is given in Figure 2.4.

Machine learning and 3D interactions — 3D UI design has been studied for about 20 years (D. A. Bowman et al. 2001). In the 3D UI context, a good overview of current state-of-the-art 3D UI methods is given in Ortega et al. 2016. On the ML side, while it has been used in user interface and user experience design for about two decades (Langley 1997; Weld et al. 2003), using ML for interaction design is to the best of our knowledge an application yet to be explored. In the user-to-computer side of the UI, ML is classically used to improve the accuracy of an existing interaction protocol:

- in Weir et al. 2012, a gaussian process regression is used to improve touch accuracy by automatically learning the offset function used to correct parallax bias. They show that this function is non-linear and user-dependent.

- in Lü et al. 2012; Z. Chen et al. 2014, supervised deep learning is used to improve the recognition rate of some multitouch gesture classes compared to standard state machine methods. This is important to note that these works enhance the sensitivity of an already existing UI with a defined interaction protocol, they do not cope with the automatic design of such a protocol.

- In the 3D interaction context, Lacoche et al. 2019 propose to automatically select the best suited interaction technique for a user based on a prior 2D

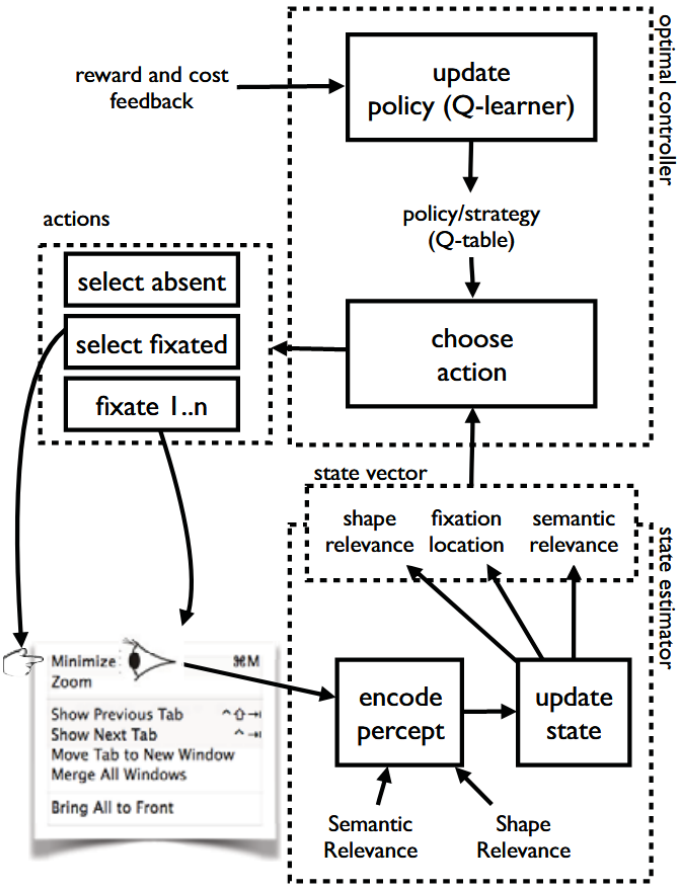

Figure 2.4 – Illustration taken from X. Chen et al. 2015. Definition of the user's menu search problem as a RL problem, allowing for the automatic evaluation of newly designed interface menus.

test. With a dataset consisting of performances and preferences of a set of users on three different interaction techniques, they train an SVM-based model in a supervised fashion to learn the best interaction technique for a user. As stated in this paper, adapting interaction techniques to the user is not a common process in the field of 3D UIs.

Knowledge discovery from unlabelled data — In Chapter 5, we try to automatically learn the prior knowledge of a user for touch interactions from a small part of all the possible interactions a user can perform. Extracting knowledge from natural data without supervision is a fascinating challenge taken on by unsupervised learning in the ML community. This challenge is closely related to natural data generation: if the model learns a close estimator of a natural distribution, we can without problems sample from it and produce "realistic" synthesized data. These generative models usually derived from Generative Adversarial Networks (GANs) (Goodfellow et al. 2014) and Variational Auto-Encoders (VAEs) (Kingma et al. 2013) are extensively used in a variety of contexts. Knowledge

discovery using unsupervised learning is an extremly prolific field that we cannot fully cover here. We instead propose to cover UI related work.

To our knowledge, generative models in HCI were only used for GUI generation:

- Nguyen et al. 2018 use a combination of RNNs and GANs to generate UI design samples from user-provided descriptions.
- Cha et al. 2019 use a spatio-temporal autoencoder as a feature extractor for facial gesture recognition with AR/VR headsets. Once trained, the encoder processes the input and a classifier is trained to recognize the facial gesture.

While not unsupervised, we can mention Beltramelli 2018 that trains a supervised DL model to generate the corresponding code from a GUI sketch: this is an example of a generative task constrained with the use of annotated information.

Reinforcement learning and generative models — In Chapter 6, we craft a user model by combining RL and the decoder part of a VAE. While our user model is a novel combination of such models, combining the latent representations of generative models with policy learning was explored in some recent work. In Higgins et al. 2017, the encoding part of a modified VAE is used to build disentangled representations for a RL policy to use in domain adaptation tasks. In A. Nair et al. 2018, a VAE together with an RL agent are trained for different purposes: synthesize training data from real observations of the policy, embed the observations to provide latent representations to the policy and measure reward signals in the latent space.

CHAPTER 3

TOUCH GESTURE RECOGNITION

Contents

3.1 Touch Gesture Data 38
3.1.1 Itekube-7 39
3.1.2 Feature Preserving Sampling 42
3.2 Neural Models 46
3.2.1 Convolutional MDGRU 48
3.3 Experimental Results 50
3.3.1 Experimental setup 52
3.3.2 Ablation Study 54
3.3.3 Comparison with the state of the art 55
3.4 Conclusions 57

Chapter abstract

We will describe in this chapter the nature of touch gesture data, their processing and the training of a device-invariant model for a classification task.

- *We address the problem of classifying sequential data characterized by variable amounts of data per time instant. We propose a range of different Deep Learning (DL) models and compare their performance on the task.*
- *We propose a new dataset of multitouch sequential gestures, Itekube-7. It is, up to our knowledge, the first of its kind, as existing datasets are restricted to symbolic gestures.*
- *We describe a novel input sampling method which drastically reduces the length of the input sequences while at the same time preserving important sharp transitions in the data.*
- *We also compare DL models against state-machine methods on Itekube-7, and the state of the art on a standard dataset in touch gesture recognition.*

Part of the work in this chapter led to the following publication:

- Quentin Debard, Christian Wolf, Stéphane Canu, and Julien Arné (2018). "Learning to Recognize Touch Gestures: Recurrent vs. Convolutional Features and Dynamic Sampling". In: *The International Conference on Automatic Face and Gesture Recognition (FG)*.

3.1 Touch Gesture Data

This chapter is focused on touch gesture recognition, and more specifically on interaction gestures (see Section 2.1.2). These gestures are meant to be interpreted as actions in an application and can have both spatial and temporal properties affecting their meaning. As an example, the smartphone interaction gestures are some of the most used: tap to select, slide a finger to move the view of a document or an image, stretch to zoom... Recognizing classes with strong spatial and/or temporal correlation combined with the natural variance of users and input conditions make for a challenging classification task. Throughout this chapter, we will process gestures recorded with Itekube's software CaptureIT displayed in Figure 3.1.

A trajectory performed by a finger on a touch surface can be seen as a sequence of (x, y) pairs with a specific time length. Defining a device-invariant model for such data though can already be tricky: devices can vary in height, width, ratio and sampling rate. Worse, this sampling period can vary in time on the same device. In practice, we observed sample rates between 5 ms and 25 ms. For a classification task, gestures of a same class can vary in spatial and temporal features. Multitouch interactions add another layer of difficulty:

- we need to track each finger to correctly build each sequence,
- and we need to synchronize them to properly interpret the gesture.

The tracking problem is systematically handled by low level libraries of current operating systems. One of its downside is that these methods can track a finger only as long as it is in contact with the surface. If it is lifted up and put on the surface again, it will get a new ID for the new trajectory; this will have to be handled by the model. In the meantime, we need to structure and denoise the dataflow returned by the device. The first part of this task is mostly performed by Itekube's library. For the purpose of training Deep Learning (DL) models, we will record touch interactions using Itekube's CaptureIT, a tool that logs interactions in an XML format (see Figure 3.2). This format includes metadata about the type of gesture performed, the user identification and the device properties, along with the datapoints sent by the device.

Beforehand, these raw data are pre-processed: x and y positions are usually given by a device using pixels or its relative coordinate system. That is problematic because deformations can occur when we compare two gestures from different

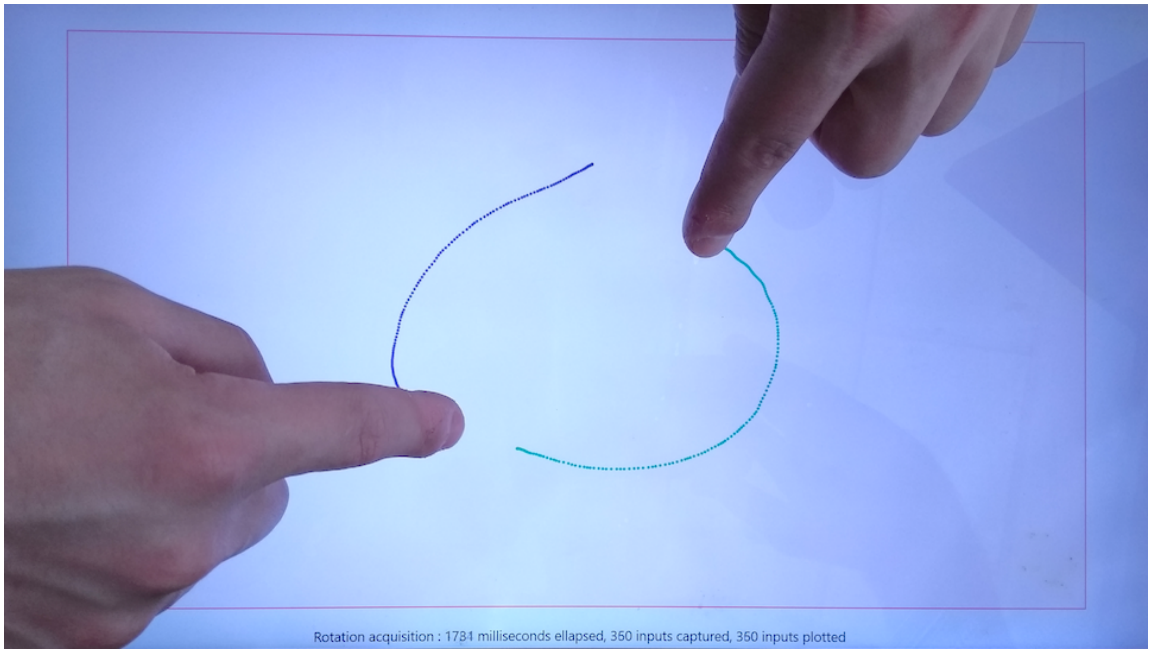

Figure 3.1 – Acquisition of an interaction touch gesture of the rotation class using CaptureIT. Each finger trajectory is displayed with a different color, each point of a trajectory corresponds to a sample. The complete gesture is saved as a XML file.

devices because of resolution, scale and/or ratio differences. In order to avoid adding workload to the model, we need absolute measurements. This can be done simply by normalizing x and y values between 0 and 1, and use the ratio of the device to deform measurements to a 1:1 ratio. Although we need the real dimension of the device in order to be scale-invariant, this information is not always available. This variation will have to be taken care of by the model. Our processing though ensures that other geometric properties are retained through devices.

Denoising is also applied: some devices tend to occasionally send the same datapoint multiple times, or give multiple positions for a unique finger during a same timestamp. This is easily handled by deleting duplicates and taking the mean position whenever multiple positions at a time are given. Mislabelling can happen when two fingers are close to each other. As there is no simple method to correct these, the model will have to handle this noise.

Whenever we refer to raw data, we mean to refer to this XML representation.

3.1.1 Itekube-7

We will now describe in this subsection the dataset collected for the classification task.

```
<export>
  <head>
    <meta name="class" value="Press Scale"/>
    <meta name="userID" value="17"/>
    <meta name="ScreenWidth" value="1536" />
    <meta name="ScreenHeight" value="864" />
    <meta name="ScreenDpiX" value="96" />
    <meta name="ScreenDpiY" value="96" />
    <meta name="Device type" value="Laptop with touchscreen" />
    </head>
  <datas>
    <data id="10" xNorm="0.817091454272864" yNorm="0.835985853227233" t="0"/>
    <data id="11" xNorm="0.800599700149925" yNorm="0.666224580017683" t="10"/>
    <data id="10" xNorm="0.817091454272864" yNorm="0.835985853227233" t="10"/>
    <data id="10" xNorm="0.817091454272864" yNorm="0.835985853227233" t="20"/>
    <data id="11" xNorm="0.800599700149925" yNorm="0.666224580017683" t="20"/>
    <data id="10" xNorm="0.817091454272864" yNorm="0.835985853227233" t="30"/>
    <data id="11" xNorm="0.800599700149925" yNorm="0.666224580017683" t="30"/>
    <data id="10" xNorm="0.817091454272864" yNorm="0.835543766578249" t="40"/>
    <data id="11" xNorm="0.800599700149925" yNorm="0.666224580017683" t="40"/>
    <data id="10" xNorm="0.817091454272864" yNorm="0.835101679929266" t="50"/>
    <data id="11" xNorm="0.800599700149925" yNorm="0.666224580017683" t="50"/>
```

Figure 3.2 – Processed XML representation of a multitouch gesture. Includes metadata and a sequence of datapoints. Each datapoint records, from left to right, the ID of the finger, its normalized x position and y position, and the relative timestamp.

We introduce a new touch gesture dataset containing 6591 gestures of 7 different interaction classes: Press Tap, Press Double Tap, Press Scale, Press Twist, Rotation, Scale and Translation, illustrated in Figure 3.4. "Press" is used to describe the action of holding a finger (usually the thumb) on the surface while performing at the same time the other action with another finger (usually the index). Dataset statistics regarding duration and gesture size are illustrated in Figure 3.3.

The recording of a gesture starts at the first contact of a finger with the touch surface and finishes when no contact is registered for more than 300ms. Data were collected from a total of 27 different people. These persons are from different professional backgrounds and aged from 12 to 62. The anonymized and pre-processed dataset is available at `http://itekube7.itekube.com`.

It is important to note that even though some gesture classes would require continuous coupling in a real application (Rotation, Scaling...), we cast the problem as a discrete recognition task without considering for example the regression of gesture parameters.

The gesture descriptions provided to the users were deliberately minimal, in order to grasp as many user variations as possible. The gestures can be executed anywhere on the screen, with any orientation, scale or velocity. Users were asked to perform the gesture naturally, we did not insist on a very strict definition of the finger state transitions. It means the user knows the different classes, and

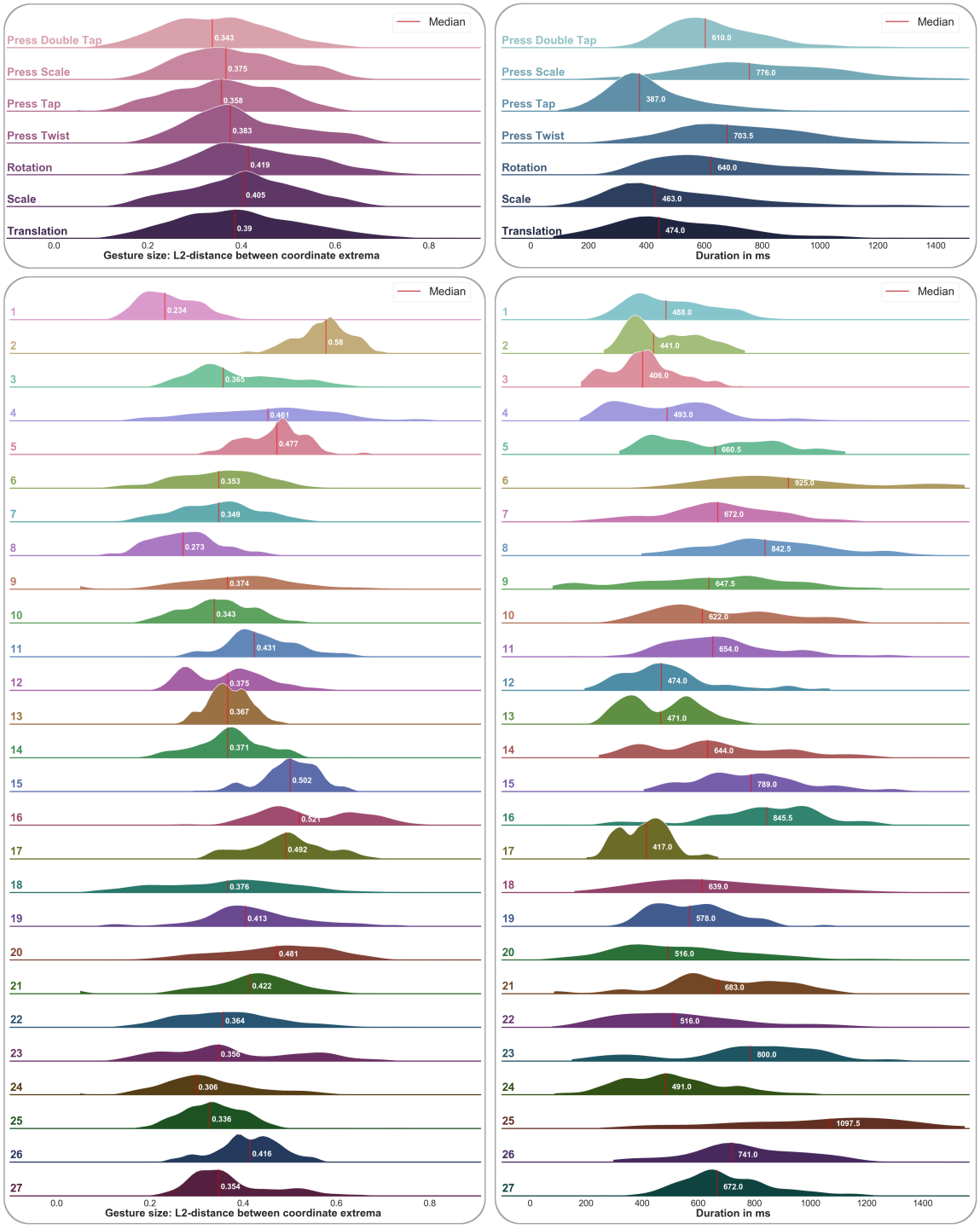

Figure 3.3 – Distribution of gesture size and gesture amplitude with respect to gesture class (top) and user (bottom) in Itekube-7. Gesture size is computed as the L2-distance between extrema coordinates $d((min_x, min_y), (max_x, max_y))$.

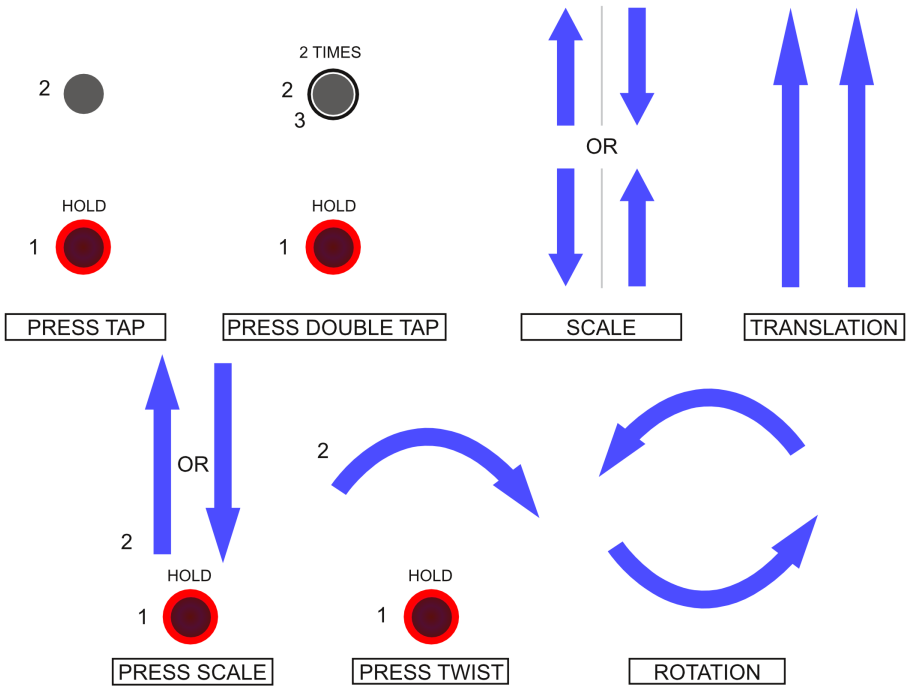

Figure 3.4 – The classes of the proposed novel multitouch gestures dataset. Red presses are held throughout the gesture. Grey touches are taps: quick, almost immobile contacts with the surface. Blue touches are slides.

performs each one as he wants as long as we can distinguish the different classes. In consequence, some classes can be defined by different transition sequences; for example, on Press Tap, some users lift the press finger first, whereas others lift the tap one first. Some classes are highly correlated: Press Tap and Press Double Tap are only distinguishable from their transitions, Press Scale and Scale differs from one small variation. From our experiments, these two classes were usually the hardest to separate (see Table 3.2).

We will define a classification problem using Itekube-7 in Section 3.3. In the next section, we describe the different models considered to solve this problem.

3.1.2 Feature Preserving Sampling

We will process raw input to a more structured representation reducing device variance and allowing for an easy synchronization of the different fingers. We call this representation a contact matrix $U_{t,i}$ (see Figure 3.5). Each element $\boldsymbol{u}_{t,i}$ of this contact matrix is a $(x, y)_{t,i}$ coordinate pair, where t the t-th timestep, and i is the ID of a finger. We define a minimum number of fingers depending on the application, and zero pad the unused entries: the number of rows in the contact matrix will define how much fingers can be processed simultaneously at a time

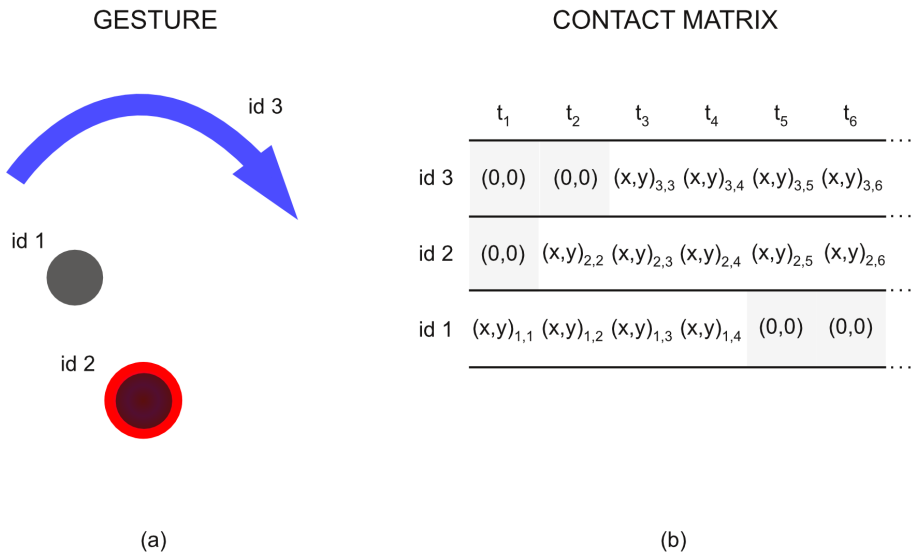

Figure 3.5 – From raw data to a normalized representation: (a) a graphical representation of a gesture, where red presses are held throughout the gesture; grey touches are taps: quick, almost immobile contacts with the surface; blue touches are slides. (b) synchronized sequences of 2D coordinates (x, y) over time for tracked finger IDs.

by our model. In practice, this number should be increased to 10 for one user, but because the classes in our experiment only take up to 3 different IDs, we will keep it down to 3 here.

During online inference for real-time models, the number of columns of the contact matrix will be fixed to a relatively small size: the contact matrix will act as a circular buffer for datapoints in these scenarios, where the newest element pushes the oldest once the buffer is full. This will give the model a small history at each timestep to take a decision ; such a model will also require an internal memory of some kind to cope with long term dependences.

In this chapter though, our experiments will deal with fully performed gestures, without early detection. This means that the contact matrices of recorded gestures will have a varying number of columns. Variable temporal length is a hard problem for convolutional models, that require fixed input size. In theory it can be dealt with sequential models, like recurrent neural networks and their variants. In practice, effects like vanishing/exploding gradients (Bengio et al. 1994) tend to limit these models to relatively short sequences, despite efforts to tackle these limitations (Hochreiter et al. 1997, Cho et al. 2014).

In the experimental section we will compare different neural sequence models, but here we chose to restrict ourselves to fixed length representations. In order to reduce the dimensionality of the input and help the model cope with redundancy and noise, it is common to sample the input before feeding it to the model.

Algorithm 2: Dynamic sampling for a desired number of k sampled points

Input: a gesture $U = \{\boldsymbol{u}_{t,i}\}, t \in \{0, ..., T\}, i \in \{0, 1, 2\}$
a transition indicator $O = \{\boldsymbol{o}_t\}, \boldsymbol{o}_t \in \{0, 1\}$
Output: Ordered and indexed set of sample points U'

1. $U' = \{U_0\} \cup \{U_x | \boldsymbol{o}_x = 1, x \in \{0, ..., T\}\}$ where U_x is the x-th row of U
2. **While** $|U'| < k$:
   $m = \underset{x \in \{0,...,T-1\}}{\arg\max} |U'_{x+1} - U'_x|$
   $U' = U' \cup \lfloor U_m \rfloor$

Existing work tends to perform sampling spatially, as for instance in Wobbrock et al. 2007: because the task in these papers is to classify symbolic gestures, temporal features such as state transitions or velocity are of minimal interest. We call state transition the moment when at least one finger is added onto or withdrawn from the touch surface. When the task involves interaction gestures (Z. Chen et al. 2014, Lü et al. 2012), dimension reduction is often done through feature extraction/embedding, not sampling.

We chose in this case to normalize the data through a static transform which compresses it into a fixed length representation. State transitions, which are key features in touch gesture recognition (Lü et al. 2012), are preserved with this sampling strategy. For gestures with high temporal content, where spatiality does not alter much the classification (such as a Press Tap, see Figure 3.4), missing one of these transitions will most likely result in a misclassification of the gesture. Using a uniform sampling, quick transitions such as a tap can be missed.

The goal is to transform a variable length sequence $(\boldsymbol{u}^n_{t,i}), t = \{1..T^n\}$ into a fixed length representation $(\boldsymbol{u}'^n_{t,i}), t = \{1..K\}$. We perform this by choosing N sampling time instants t which are common over the finger IDs i. The set sampling points should satisfy two properties:

- (i) the points should be spaced as uniformly as possible over the time period;
- (ii) the sampled signal should preserve finger transitions, i.e. transitions (finger up or finger down) should not be lost by the transform.

To formalize this, we introduce a transition indicator variable o_t:

$$o_t = \begin{cases} 1 \text{ if transition at time } t, \\ 0 \text{ else.} \end{cases}$$

Then, the *inverse* problem, namely creating observed gesture sequences from a set of given sample points, can be modeled as a probabilistic generative model,

in particular a Hidden Semi-Markov Model (HSMM) (Yu 2010) with explicit state duration modelling. Obtaining samples from the observed sequence then corresponds to decoding the sequence of optimal states.

In this formulation, the indicator variables o_t correspond to the observations of the model, whereas the hidden state variables S_t correspond to the sampling periods. Each state variable can take values in the set of hidden states $\Lambda=\{1..K\}$, which correspond to the K target samples. The desired target sampling points correspond to instants t where changes occur in the hidden state S_t. The transition function of the model is a classical forward model (upper case letters indicate random variables, lower case letters realizations):

$$\begin{aligned} \boldsymbol{q}_{(i,d,j)} &\triangleq P(S_{[t+1:[}=j|S_{[t-d+1:t]}=i) = \\ &= \begin{cases} \frac{1}{S} & \text{if } j=i+1 \\ 0 & \text{else} \end{cases} \end{aligned} \tag{3.1}$$

The duration probability distribution encodes above property (i), which aims sampling points equally spaced with a target period of $\frac{T^n}{K}$:

$$\begin{aligned} \boldsymbol{p}_{j,d} &\triangleq P(S_{[t+1:t+d]}=j|S_{[t+1:[}=j) = \\ &= \frac{1}{Z}(d-\frac{T^n}{K})^2 \end{aligned} \tag{3.2}$$

where Z is a normalization constant.

The observation probabilities encode the hard constraints on the transitions (above property (ii)):

$$\begin{aligned} \boldsymbol{b}_{j,d}(o_{t+1:r+d}) &\triangleq P(o_{[t+1:t+d]}|S_{[t+1:t+d]}=j) = \\ &= \begin{cases} 0 & \text{if } \sum_{j=t+1}^{r+d} o_j > 1 \\ \frac{1}{Z'} & \text{else} \end{cases} \end{aligned} \tag{3.3}$$

where Z' is a normalization constant. In other words, sampling periods spanning over more than 1 transition are forbidden, which makes it impossible to lose state features.

If the number of transitions is lower than the number K of desired sampling points, then the Semi-Markov model parametrized by the above equations [1] can be decoded optimally using the Viterbi algorithm (Yu 2010). Because the complexity is high, we solve the problem faster and heuristically with a greedy algorithm, which first selects all transition points for sampling, and then iteratively adds additional sampling points in the longest remaining sampling intervals (see Algorithm 2 and Figure 3.6).

1. For space reasons, we omitted the initial conditions which ensure that the optimal sequence begins with state 1 and terminates with state T.

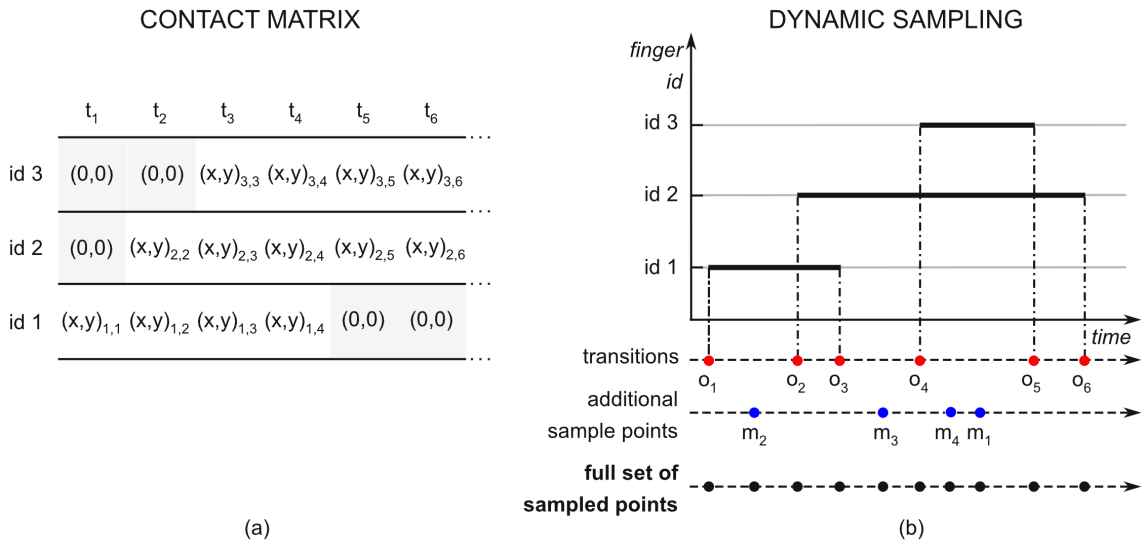

Figure 3.6 – (a) Example of a gesture; (b) its feature preserving fixed length representation. First, transitions (in red) are sampled. Then additional sample points (blue) are selected to temporally homogenize the sample until it reaches the limit sample size (here: 10).

A drawback of the proposed sampling method is the variations in the sampling rate over gestures: since the sampling rate is not uniform over the full interval, we lose velocity information if we only consider spatial coordinates. One possibility would be to keep timestamp information additionally to coordinates, making it possible for the model to extract the required local velocity. In practice, experiments showed that the resulting gestures are sufficiently equally sampled and adding timestamps did not improve performance.

3.2 Neural Models

Our classification problem is a sequential learning problem of fixed length but with input dimensions varying over time. Each input gesture is a sequence $(\boldsymbol{u}^n_{t,i})_{t=1,T_n}$ of length T_n where n is the index of the touch gesture, i the finger ID provided by the touch device and $\boldsymbol{u}^n_{t,i} = (x^n_{t,i}, y^n_{t,i})^T$ are the spatial 2D coordinates of finger i on the touch screen. An example of such a sequence is illustrated schematically in Figure 3.5a. In the following, gesture indices n can be omitted for clarity, unless necessary for comprehension.

Our contact matrix representation helps us cope with this varying input dimension. We could also have used handcrafted representations to embed this set of samples into a fixed length representation, which describes the spatial distribution of the points. In the literature, several representations have been proposed, but we will only mention Shape Context (Belongie et al. 2002). In Belongie et al. 2002,

log polar histograms are computed for a point cloud, which describe the positions of individual points relatively to other points in the cloud.

In our work, we prefer to automatically learn a suitable feature representation from data and fix the max number of IDs observable at a time. This feature learning can either be done statically by using fully connected or convolutional features, or sequentially. The latter implies to integrate the different samples $\boldsymbol{u}_{t,i}$ iteratively over i using a sequential model.

It is very important to remark here that the dimension over finger IDs i is *unordered*. In other words, the model is required to ignore the order of the data over its finger ID dimension. Example given, if we consider three finger trajectories with ID 1,2 and 3, all gestures combining these trajectories at the same time($\{1,2,3\}$, $\{3,2,1\}$...) are the same. The model will need to learn to embed the data into a spatial representation, in a similar spirit as Shape Context histograms. Ensuring invariance to finger order is therefore important; a simple way to enforce is data augmentation, i.e. shuffling finger IDs during training in order for the model to observe every possible combination of finger ID.

The resulting features f_t can then be integrated temporally: $\hat{y}=\psi(f_{1:T},\theta_\psi)$, parametrized by θ_ψ. This model ψ operates in the time dimension and predicts a gesture class $\hat{y}$ for the sequence. We will provide four Deep Neural Network (DNN) models applying four different strategies to perform this task:

A baseline Recurrent Neural Networks (RNN) (Figure 3.7a), in which features f_t over finger IDs i are computed statically in a fully connected fashion. These features are then integrated temporally with the recurrent mechanism. The architecture is a two-layer Long Short-Term Memory (LSTM) iterating on the time dimension and a Fully Connected (FC) layer activating the output of the last layer to produce logits.

A baseline Multi-Dimensional Recurrent Neural Network (MDRNN) displayed in Figure 3.7b, where f_t is also computed sequentially. The model iterates upon the finger ID and time dimension all together in a grid-like fashion. Our architecture is a two-layer Multi-Dimensional Long Short-Term Memory (MDLSTM). The layer stacking with MDRNNs means that at time and space (i,j), the cell of a higher layer will use as an input the activation of the lower cell during the corresponding space and time positions (i,j), observing this transitory representation rather than the input data. We feed the last activation in both the time and space dimension to a FC layer to produce logits.

A baseline Convolutional Neural Network (CNN) (Figure 3.7c) which, instead of an iteration mechanism, builds a hierarchy of local features in both dimensions. At first, the visible scope of each feature is limited to the convolution kernel size, but with each layer, the scope widen (building local features of feature maps) until the whole sequence has been parsed to build features.

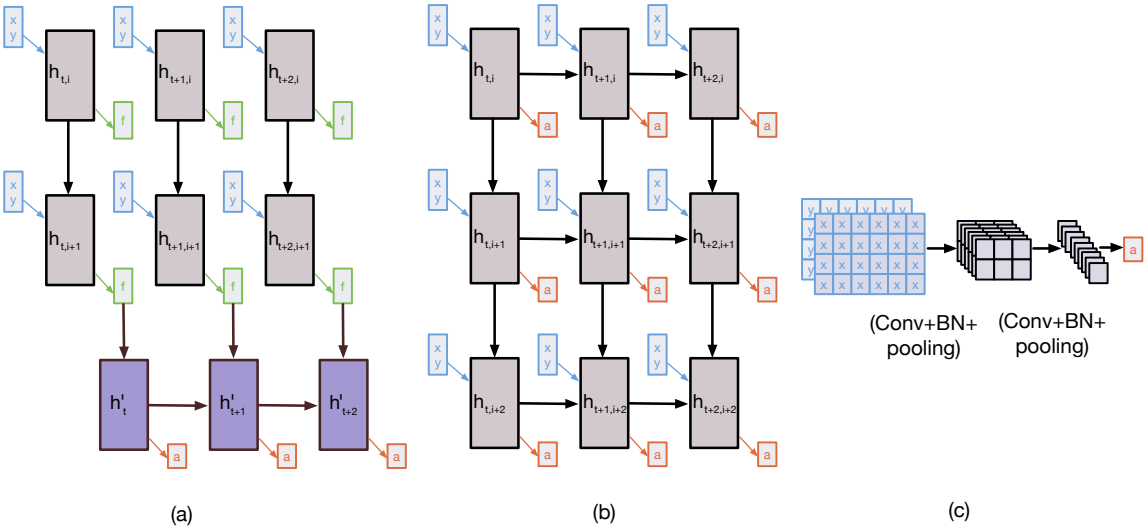

Figure 3.7 – Feature representations for sequences: (a) Sequential learning of fixed-length representations. data are aggregated in a fully connected fashion by an RNN and then integrated temporally; (b) a MDRNN integrates in both dimensions, iterating from top-left to bottom-right; (c) convolutional features for sequential data. Higher layers integrate information from a bigger window in the sequence. Input data is shown in blue, output in red (best viewed in color).

A new Convolutional Multi-Dimensional Gated Recurrent Unit (MDGRU) architecture, combining both multi-dimension recurrence and convolutional features. The proposed CNN uses 2D spatio-temporal filters. The sequential input data is structured into a 3D tensor, with the finger ID as first dimension, time as the second dimension, and input channels as the third dimension (x coordinates correspond to the first channel and y coordinates to the second channel). An illustration of the architecture used in the experiments is given in Figure 3.8, and a detailed description will be given in Section 3.2.1.

We will argue the superiority of convolutional features in our case, and in Section 3.3 we will corroborate these arguments through experiments. Details of RNNs, LSTMs, MDLSTMs Gated Recurrent Units (GRUs) and CNNs can be found in Section 2.2.

3.2.1 Convolutional MDGRU

We now describe our novel convolutional MDGRU architecture. The idea of combining convolutional operations and multi-dimensional recurrence is not new: Van Den Oord et al. 2016 inspired us in this sense, using convolution operations to process the input and compute the states of MDLSTM layers in an image generation

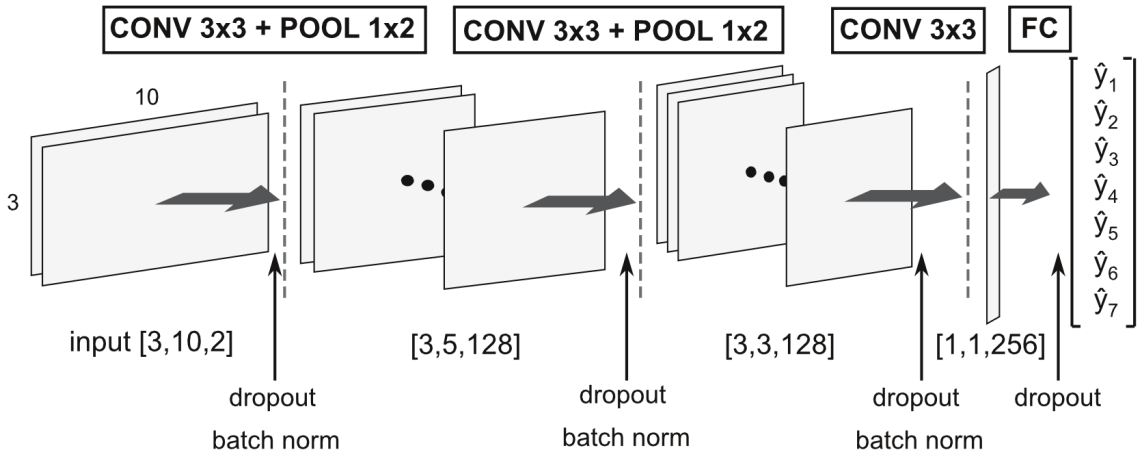

Figure 3.8 – Illustration of the CNN pipeline (batch dimension omitted). Input is processed by a first conv layer of kernel size $(3,3)$ followed by a max-pooling $(1,2)$ that produces 128 feature maps of size $[3,5]$. These feature maps are processed by another similar layer producing 128 feature maps of size $[3,3]$. A last conv layer reduces the feature maps to 256 scalars. Finally, a FC layer is applied to produce the logits.

task. While alternating convolutional and MDLSTM layers has seen some successful uses (Moysset et al. 2015; Moysset et al. 2018; Naz et al. 2017; Breuel 2017), we believed that these architectures could be drastically reduced in size while still being capable of combining both the robust local features of CNNs and the long term dependency modelization of MDLSTMs. Instead of alternating these layers, we can change the way the MDLSTM processes its inputs. Even more so, we can reduce the number of weights by generalizing GRUs to multi-dimensional recurrence. While this model can be theoretically applied to inputs of any dimension, we will provide the equations for the 2D case for lisibility purposes. We build upon the equations presented in Section 2.2, mixing the concept of MDLSTM with the specific gating of GRUs to formulate an implementation of a MDGRU:

$$
\begin{aligned}
&\begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \\ \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix} = \sigma\left(\boldsymbol{x} \cdot \boldsymbol{U} + \boldsymbol{h}_{i-1,j} \cdot \boldsymbol{V}_1 + \boldsymbol{h}_{i,j-1} \cdot \boldsymbol{V}_2\right) \\
&\tilde{\boldsymbol{h}}_1 = tanh(\boldsymbol{x} \cdot \boldsymbol{U}_{\tilde{h}_1} + \boldsymbol{r}_1 \odot \boldsymbol{h}_{i-1,j} \cdot \boldsymbol{V}_{\tilde{h}_1}) \\
&\tilde{\boldsymbol{h}}_2 = tanh(\boldsymbol{x} \cdot \boldsymbol{U}_{\tilde{h}_2} + \boldsymbol{r}_2 \odot \boldsymbol{h}_{i,j-1} \cdot \boldsymbol{V}_{\tilde{h}_2}) \\
&\boldsymbol{h} = (1 - \boldsymbol{z}_1) \odot \boldsymbol{h}_{i-1,j} + (1 - \boldsymbol{z}_2) \odot \boldsymbol{h}_{i,j-1} + \boldsymbol{z}_1 \odot \tilde{\boldsymbol{h}}_1 + \boldsymbol{z}_2 \odot \tilde{\boldsymbol{h}}_2 \\
&y_t = h_t
\end{aligned}
\tag{3.4}
$$

In this case, we define a reset gate r_i and an update gate z_i for every dimension using information from both preceding hidden states and the current input, and

we compute the new hidden state by independently leveraging information from both dimensions. We can criticize a few points in this implementation:

- the number of added weights is important, with two new gates and a supplementary transitional hidden state per dimension,
- the update mechanism of each dimension does not take into account the transitional hidden state of the other dimension,
- and the input weights U of each component only use the current input value rather than a local neighborhood.

This last point highlights the limitations of MDRNNs compared to CNNs: they rely solely on the iteration upon hidden states to compute both local and global features from the data. Because of the simple nature of the hidden state (a vector), the bandwidth of features encoded in the hidden state is limited, even more so with vanishing and exploding gradient problems, whereas CNNs natively separates features with different levels of abstraction and can simply build collections of local features with a more stable gradient. This motivates us to try to learn local features in the input weights $\boldsymbol{U}$ instead and use the hidden state to focus on long term dependencies between such local features. This can be simply done by changing the matrix multiplication $\boldsymbol{x} \cdot \boldsymbol{U}$ by a convolution operation $\boldsymbol{x} * \boldsymbol{U}$. Considering the first two bullet points, we can instead choose to compute a unique transitional hidden state using information from both dimensions, and fuse the update process:

$$\begin{aligned}
\begin{pmatrix} \boldsymbol{z} \\ \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{pmatrix} &= \sigma\left(\boldsymbol{x} * \boldsymbol{U} + \boldsymbol{h}_{i-1,j} \cdot \boldsymbol{V}_1 + \boldsymbol{h}_{i,j-1} \cdot \boldsymbol{V}_2\right) \\
\tilde{\boldsymbol{h}} &= tanh(\boldsymbol{x} * \boldsymbol{U}_{\tilde{h}} + \boldsymbol{r}_1 \odot \boldsymbol{h}_{i-1,j} \cdot \boldsymbol{V}_{\tilde{h}} + \boldsymbol{r}_2 \odot \boldsymbol{h}_{i,j-1} \cdot \boldsymbol{W}_{\tilde{h}}) \\
\boldsymbol{h} &= (1 - \boldsymbol{z}) \odot \boldsymbol{h}_{i-1,j} + (1 - \boldsymbol{z}) \odot \boldsymbol{h}_{i,j-1} + \boldsymbol{z} \odot \tilde{\boldsymbol{h}}
\end{aligned} \tag{3.5}$$

This is our Conv-MDGRU cell, processing at each iteration a local neighborhood of the current input and using in one step both preceding hidden states to compute the new one. While using the same factor $(1 - \boldsymbol{z})$ to leverage both older hidden states can lead to information loss, we postulate that this also enforces a global structure of the hidden state, precisely in order to limitate this information loss. Our full Conv-MDGRU model is simply a one-layer Conv-MDGRU, with a FC layer processing the last activation to produce the logits.

3.3 Experimental Results

We experimented mostly on Itekube-7, and a second dataset was used for additional evaluation: the Mixed Multistroke Gestures (MMG) dataset (Anthony

	Methods	Dynamic sampling	Data augmentation	Accuracy
A	State Machine	-	-	59.50
B	LSTM	-	X	58.71
C	LSTM	X	X	73.10
D	ST-LSTM	-	X	60.01
E	ST-LSTM	X	X	87.72
F	CNN	-	-	65.96
G	CNN	-	X	73.00
H	CNN	X	-	83.93
I	CNN	X	X	**89.96**
J	Conv-MDGRU	X	X	**92.38**

Table 3.1 – Results on the proposed multitouch dataset: different sequential models and ablation study. The state machine model is a deterministic recognizer developped at Itekube tweaked to compete on Itekube-7. While its results are perfectible, it would require a lot of engineering and will most likely not attain the precision of our best performing models. Except for the Conv-MDGRU, all results were discussed in Debard et al. 2018.

et al. 2012). The latter dataset contains 9600 gestures from 20 participants in 16 classes Figure 3.9. All classes are symbolic gestures: the temporality of the sequence has no importance for the classification, we classify the symbol drawn by all the trajectories. This classification problem is thus simpler than the Itekube-7 one. Each class is performed 10 times by a participant at three different speeds, resulting in 30 occurrences per class per participant. We could not find other interaction gesture datasets to compete on, and results on other symbolic gesture datasets would not be of much interest, given the simpler nature of the task. We will detail the experiments on the MMG dataset at the end of the section.

Because the variance from one user to another is a capital notion that a model should generalize on well, we will be splitting the gestures per user, following a "Leave-One-Subject-Out" protocol. Indeed, we need to ensure that the model can correctly label gestures from an unknown user. To do so, each user from the dataset was given a number, from 1 to 27. All the gestures performed by users 2, 3, 9, 12, 17, 23 and 27 were used as the test set. The rest of the gestures was used for the training and the validation set. The hyperparameter tuning was optimized over the validation set.

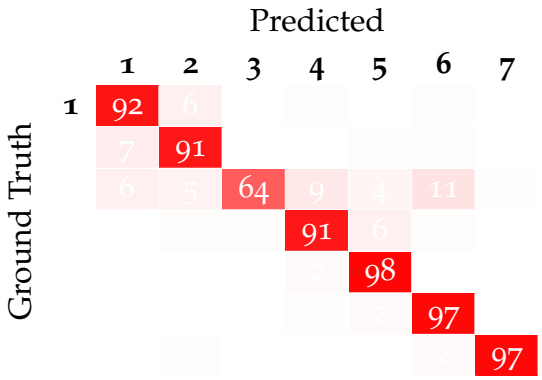

Table 3.2 – Confusion matrix on the test set for our dataset. Classes are: Press Tap, Press Double Tap, Press Scale, Press Twist, Rotation, Scale, Translation

It is important to know that our best performing model **Conv-MDGRU was developped after the initial submission of this work to the conference, and as such is still unpublished**. We will provide the details and the results obtained with this model but we will keep the convolutional model as a reference for ablation studies, as performed in the publication.

3.3.1 Experimental setup

Data augmentation is performed for the training set on the ID permutations: each gesture of the training set is augmented by permutating its lines (corresponding to a single finger trajectory). For our multitouch dataset, we set the maximum number of fingers to 3 and thus keep all 6 permutations of each gesture. We also perform data augmentation on gesture scale, applying a random scaling factor (between 0.75 and 1.5) to each of the permutations, adding up 6 artificial gestures per gesture. This brings the size of the augmented training set to 12 times the original one. Thanks to the freedom given to the subjects to perform gestures, data augmentation on gesture orientation was not necessary.

Architectures and implementation details — We implemented all models in Tensorflow (Martín et al. 2015). All hyper-parameters have been optimized over the validation set, the test set has not been used for this. In particular, the number of sampled points is set to K=10: we experimentally found this sampling size to be the best trade-off between information maximization and redundancy minimization for our problem. All models use a learning rate of 0.001 and an Adam optimizer with decay rates of 0.9 (β_1) and 0.999 (β_2).

- For readability purposes, we refer to convolutional layers with x feature maps as CONVx layers, and max pooling layers as POOL. The convolutional model has the following architecture: a CONV128 layer, a 1×2 POOL layer (max

pooling only on the time dimension), again a CONV128 layer and a 1×2 POOL layer, a CONV256 layer and a FC layer providing the prediction score for each class. Activation functions are ReLU (Krizhevsky et al. 2012), all convolutional kernels are 3×3. The FC layer is linear (no activation function). We apply dropout (Srivastava et al. 2014) to each CONV layer and FC layer. Dropout is set to 0.5, applied to all layers except the last. The network is further normalized using batch normalization (Ioffe et al. 2015). The model is trained for 400 epochs.

- The LSTM used in Table 3.1 is the standard version of Hochreiter et al. 1997, trained for 300 epochs. For this model, all 3 (x, y) coordinate pairs are concatenated to produce a 6 dimensional feature vector. There are 128 hidden units for a cell, and a FC layer is used to produce the logits.

- For the 2D Spatio-Temporal LSTM we used the variation of Liu et al. 2016, which is itself a variant of the MDLSTM (Graves et al. 2007). The model is detailed in Equation 2.15. We apply recurrent dropout as defined in Semeniuta et al. 2016 with the drop rate set to 0.35 and recurrent batch normalization (Cooijmans et al. 2016). It is trained for 150 epochs. In our model, each cell possesses 64 hidden units and the trust parameter is set to 0.5. An activation layer takes all cell outputs (from the whole grid) to compute predictions.

- Lastly, the hidden state size of the Conv-MDGRU is set to 128 with no activation function. The kernel of the input convolution is of size $(3, 3)$. Recurrent dropout (modified to randomly drop $\tilde{h}$) and batch normalization is applied to this layer. A FC layer uses the last activation of the Conv-MDGRU layer to produce logits. This model is trained for 100 epochs.

For every model, we use batches of 128 gestures. Models are optimized on the cross-entropy loss, computed from logits by applying a softmax.

Evaluation protocol — We report classification accuracy on the test set, which has been used neither for training nor for architecture and hyperparameter optimization. The split between test data, validation data and training is subject-wise. No subject (user) is in more than one subset of the data.

All models have been optimized using the validation error which is measured with the leave-one-subject-out cross-validation (LOSOCV) protocol, common in gesture recognition. All but one user are used as training data, and one is used as validation. We perform the training for each possible permutation and average each score to obtain the final result. After optimization of architectures and hyperparameters, the full combined training+validation set was used again for retraining the final model tested on the test set.

	Sampling type	Accuracy
A	No sampling	73.00
B	Uniform	80.95
C	Rand. Uniform	80.62
D	Dynamic	**89.96**

Table 3.3 – Comparison between different sampling methods for our convolutional model. The uniform sampling samples evenly spaced datapoints. The random uniform sampling is a noisy version of the uniform sampling, randomly sampling around the uniformly selected datapoints using a gaussian distribution.

3.3.2 Ablation Study

In order to assess the effectivness of each part of the process, we proceed to an incremental evaluation of our method. All the results are displayed in Table 3.1.

- The two recurrent baselines perform worse than the convolutional model, which confirms the reasoning that hierarchical respresentations over time are useful for sequences. However, 2D-LSTMs do have several interesting properties, while performing close to the convolutional model (-2 points). They use only 138,631 trainable weights in total (against 446,983 of the CNN), and they can be unrolled on varying input dimensions. In general, recurrent models are more easily generalizable while CNNs tend to perform better for this task.

- We trained a model without transition preserving dynamic sampling. To this end, and in order to still have a fixed length representation for the convolutional model, the sequences were cropped or padded to 104 points, which is equivalent to 1200ms. This value is fitting 95% of all the dataset gestures. We observed that longer gestures were most likely held for too long or noisy. The architecture was optimized for this experiment (again on the validation set), which resulted in two additional convolutional layers with pooling which are able to cope with these longer sequences. We added 1×3 max-pooling over the time dimension. With 65.96% on the test set, the model performs much worse than the version with dynamic sampling. With data augmentation the recognition rate rises to 73.00%, but is still far from the 89.96% we obtain from the full model with dynamic sampling.

- Data augmentation is an important part of the method, which increases the invariance of the respresentation with respect to the order of the finger

Methods	Evaluation protocol	
	Number of parameters	Accuracy
LSTM	22,663	73.10
4 layers stacked LSTM	472,839	73.65
CNN	112,903	86.99
CNN	446,983	89.96

Table 3.4 – Result difference when changing the number of parameters of the models.

IDs delivered by the device, as well as the scale of the gestures. The best performing model w/o augmentation scores at 83.93%, almost 6 points below the best performance with augmentation.

- Our sampling method was also compared to a uniform and a randomized uniform sampling (see Table 3.3). This last sampling method was defined by uniformly segmenting the sequence, and then picking a sample from each segment using a normal distribution.

Convolutional and sequential models — In order to ensure that the result gap between convolutional and sequential models was not correlated to the number of parameters, we show in Table 3.4 the different results obtained with varying number of weights with these two types of models. The weights of the convolutional model were tweaked by changing the number of feature maps, while the LSTM model was made "deeper" by stacking layers and increasing the hidden state size (more efficient than just increasing the hidden state size).

Runtime complexity — All computations were done on Nvidia Titan-X Pascal GPUs. Training the convolutional models on our dataset takes 1h 11min (∼400 epochs). Testing a single gesture takes 1.5 ms, including the dynamic sampling procedure. To assert the portability of this model, runtime on CPU (Intel i7-7700HQ, laptop) is 5ms, only 3.3 times slower than on GPU. This is because input tensors are small, resulting in limited GPU acceleration, I/O being the bottleneck. It is also possible to greatly reduce the number of weights in the CNN while losing little precision. This trade-off will be considered if a device requires minimal computation.

3.3.3 Comparison with the state of the art

We applied the method on the MMG dataset, one of the very few standard datasets of this problem. On this dataset, gestures are complex drawings with a

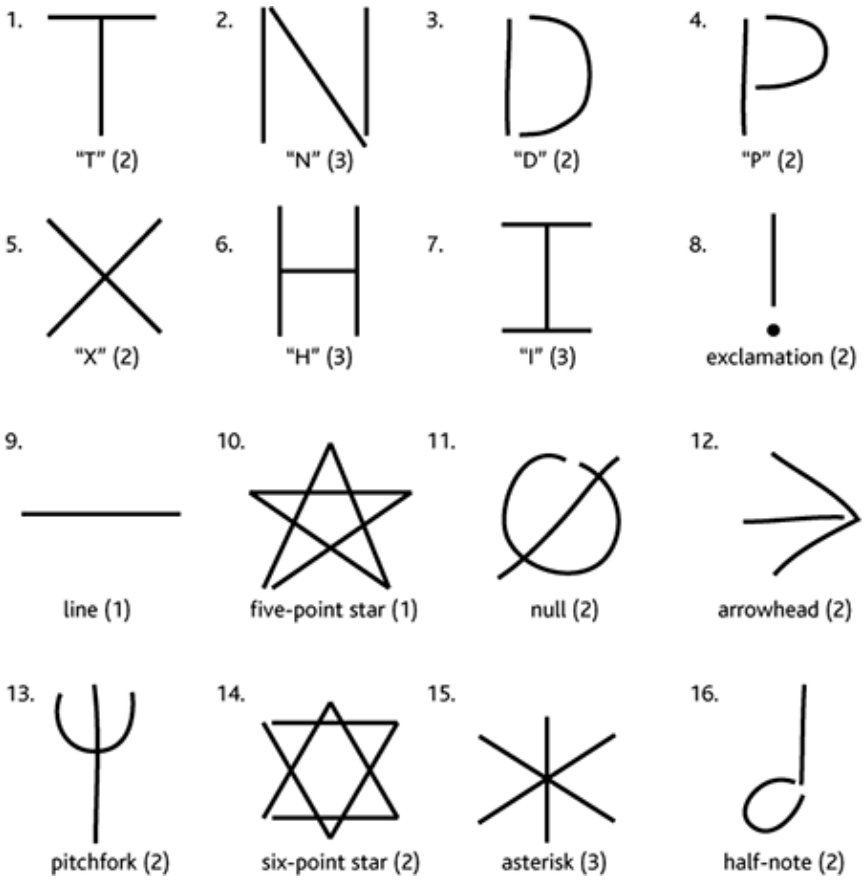

Figure 3.9 – Illustration of the different MMG classes (illustration taken from Anthony et al. 2012). Input is processed by a first conv layer of kernel size $(3,3)$ followed by a max-pooling $(1,2)$ that produces 128 feature maps of size $[3,5]$. These feature maps are processed by another similar layer producing 128 feature maps of size $[3,3]$. A last conv layer reduces the feature maps to 256 scalars. Finally, a FC layer is applied to produce the logits.

finger or a stylus, performed with a varying number of strokes for a same class. This problem can be seen as symbol recognition. As such, state transitions are not relevant for this task, because temporality does not give meaningful information. We therefore detected geometric transitions as spatial discontinuities (corners) in the finger trajectories. To this end, we calculated thresholded angles of the spatial gradient and thresholded derivates of these angles. We then sampled 48 points of each gesture using the feature preserving method given in Section 3.1.2 (as opposed to uniform sampling of 96 points done in Anthony et al. 2012).

For this dataset, we chose an architecture similar to the 6-layers CNN for our own dataset. However, convolutions are 1D, as there is only one stroke at one time on the surface. The architecture is described as follows: 2x(CONV-128+POOL), 2x(CONV-256+POOL), CONV-512, 1x FC. The first CONV layer uses a kernel of

Methods	Evaluation protocol	
	LOSOCV	User-independent
Proposed method	98.62	99.38
Greedy-5 (Anthony et al. 2012)	N/A	98.0

Table 3.5 – Results on the MMG Dataset. The User-independent protocol is detailed in Anthony et al. 2012.

size 5 while the others use a kernel of size3. There are a total of 747,536 trainable parameters.

Table 3.5 presents the results on this dataset. We used different evaluation protocols: there is no test set for this dataset, so we used the LOSOCV protocol described above for our validation error, as well as the user-independent protocol used in Anthony et al. 2012. In this protocol, a user among 20 is randomly selected as the test subject, while the training is performed on the 19 other users. The training is done using 9 samples from every class randomly taken from every training user. We then classify one random sample of every class from the test subject. This process is performed 100 times and classification results are averaged.

We can see that the obtained performance of 99.38% is close to perfect recognition and beats the state of the art of 98.0% given in Anthony et al. 2012. This further confirms the interest of our model and sampling procedure.

Because of the lack of publicly available datasets for interaction gestures, we cannot further compare our model to existing methods, and we claim that comparing our model on another symbolic gesture dataset would not yield any meaningful information.

3.4 Conclusions

We have proposed a novel method for multitouch gesture recognition based on learning convolutional features from fixed length input. We have also introduced a dynamic sampling algorithm which preserves sharp features in the input data. We demonstrated the effectiveness of DL models, validating this approach on our dataset of interaction gestures and on an existing symbolic gesture dataset. As expected, the symbolic gesture task proved easier to solve with DL models: the need to recognize a combination of temporal and spatial patterns makes our classification task more complex. This work is the first step toward a rich and adaptative model for touch surface interactions, its runtime complexity allowing for the development of real-time applications.

There are two evident generalizations of this classification task:

- segmentation of a data flux in order to recognize multiple gestures at once. This will open our research to multi-user interactions and long dependency gestures, e.g. a gesture performed with a sequence of successive (but not occuring at the same time) finger trajectories.

- Real-time classification of gestures: this implies working with hierarchical models and early detection processes (Hoai et al. 2014).

A lead for the first generalization would be either deterministic segmentation (i.e. clustering), or model the task as a seq2seq problem. Indeed, we can imagine a model processing the data before classification, returning a cluster label for every finger in contact. Because prior knowledge about classes is relevant for clustering, this model could also be part of a bigger one performing clustering as well as classification.

The second generalization can most likely be addressed using hierarchical models combining convolution features and recurrent models. Given a small time window (that will fix the reaction time of our model), a small CNN computes local features of the contact matrix buffer, while a higher level recurrent model aggregates these features and produces one label (or the absence of label) at each timestep. Combining both real-time and clustering capabilities is key to tackle the challenge of a general multi-user multitouch action recognizer.

With the touch gesture recognition subject covered, we will now address the learning of interaction protocols.

CHAPTER 4

CONTINUOUS COUPLING OF GESTURES AND ACTIONS

Contents

4.1 Interaction Protocols as Reinforcement Learning Agents **60**
4.1.1 Formalizing the Interaction Protocol Design Problem . . 61
4.1.2 Learning Signals and Environments 65
4.2 Experiment: Learning a Touch Interaction Protocol **67**
4.2.1 Environment and known solution 68
4.2.2 Handcrafted User Model 69
4.2.3 Reward Function 70
4.2.4 Experimental Setup 71
4.2.5 Results 73
4.3 Conclusions **73**

Chapter abstract

Now that we treated the subjects of touch gesture recognition and touch data processing, our next step is to focus on learning how to interpret these gestures. In other words, how to map gesture properties to actions in a virtual environment.

In this chapter, we will formalize the interaction protocol design problem, expressing it as a continuous optimal control problem; we will provide the methodological and experimental ground to establish Reinforcement Learning as a powerful learning mechanism for interaction protocols. Using Reinforcement Learning allows for both the learning of interaction protocols and the online tuning of the protocol during use, making a first step toward co-adaptive interfaces.

This modelization will then be tested on a virtual environment requiring for the user to move an object to a destination, with the goal of automatically learning an already known optimal interaction protocol for the task. We will describe the experimental setup and discuss generalization.

Some of the work in this chapter is part of the following publication:

- Quentin Debard, Jilles Steeve Dibangoye, Stéphane Canu, and Christian Wolf (2019). "Learning 3D Navigation Protocols on Touch Interfaces with Cooperative Multi-Agent Reinforcement Learning". In: *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD)*.

4.1 Interaction Protocols as Reinforcement Learning Agents

When a user uses an electronic device in order to achieve a task, he performs actions using an interface. These actions modify the environment or the application of the task and the user receives a feedback (usually visual and/or audible) to signal corresponding modifications. We call the translation of user finger motions to actions in the virtual environment an interaction protocol.

We give an illustration in Figure 4.1 of Autodesk Navisworks, an application for 3D design review, used here to display a construction. In this case, the interaction protocol must interpret user gestures on a touch screen to camera motion in the 3D environment. The interaction protocol currently used by our industrial partner is defined by Itekube's TouchIt plugin with settings displayed on the top-left. The current finger motion is a two-finger pinch (or scale) where two fingers are drawn closer through time. This finger motion is interpreted as a translation on the z-axis of the camera, orthogonal to the display plan: the viewpoint is moved back proportionally to the finger motion with a coefficient set in TouchIt. The complete interaction protocol is a collection of correspondance rules like this one to cover all the actions that a user may want to perform in the virtual environment.

The preceding chapter dealt with the automatic categorization of such user actions for touch devices. We now focus in this chapter on the correspondance between user actions and actions in virtual environments. These categorization and coupling tasks can be complementary (and lighten the load on the coupling model) but we will treat them separately here: our coupling model will categorize gestures internally.

An interaction protocol is application-dependent: some applications may use different sets of user action categories, and the same user action in two different environments might have different semantics. Even more so, an interaction protocol might be parametrized by the current state of the environment (e.g. current viewpoint in a 3D navigation system). Its complexity can rapidly scale with the complexity of said application/environment.

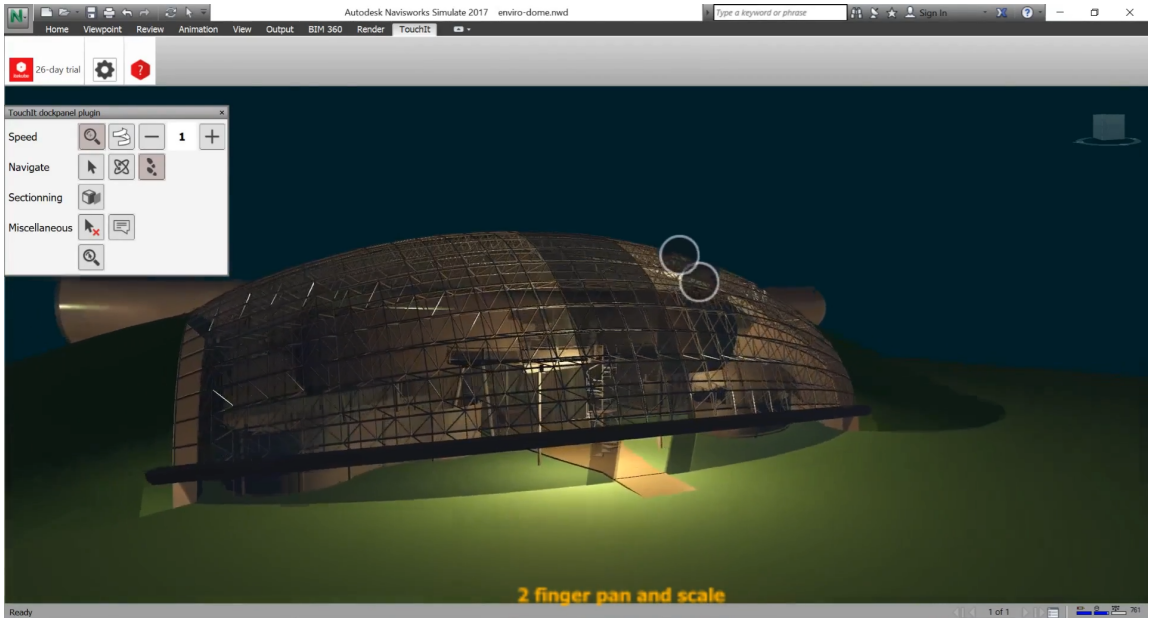

Figure 4.1 – Example of a touch User Interface (UI) for 3D navigation. This is the Autodesk Navisworks software combined with Itekube's TouchIT, an overlay used to identify and interpret finger motions to actions in the application. Fingers in contact with the surface are represented as white circles. Their current motion is interpreted as a 2 finger pan and scale and is applied with the set parameters displayed in the top-left window.

4.1.1 Formalizing the Interaction Protocol Design Problem

In the following work, we will sometimes use the term interface agent to describe an interaction protocol modelled as an Reinforcement Learning (RL) agent for lisibility purposes. Stricto sensus, the interaction protocol is just part of an interface, but because we will not mention any other part of a UI in this work, we believe this simplification should not induce confusion.

If we condense the properties of an interaction protocol, it can be seen as a function with:

- **two inputs**: user actions and eventually environment properties,
- **one output**: a vector of actions to be performed in the environment,
- a parametrization optimized to **maximize usability**.

We want in this chapter to automatically learn such interaction protocols, coupling user actions with actions in a virtual environment. Among the challenges, quantifying usability (and more precisely user satisfaction) and collecting training data will be the hardest to overcome. We will experiment again on touch inter-

faces to showcase practical cases. The motivation of automatically learning these interaction protocols is twofold:

- allow for the design of co-adaptive interfaces, able to continue learning during real use: the interface also adapts to the user,
- and potentially discover new interaction protocols for complex applications.

Indeed, Machine Learning (ML) is not meant to replace humans in the design process of UIs: ultimately, these UIs are to be used by humans, and the complexity of priors, expectations and the notion of intuitivity seem too complicated to be entirely modelled as of now. We claim that we can however model part of it, and doing so will allow for the design of better interfaces through the data processing power of ML.

In this chapter, we propose to solve this interaction protocol design problem for touch interfaces by casting the interaction protocol as an RL agent. RL is traditionnally used to solve Markov Decision Processes (MDPs): we need to find a setup that can be interpreted as such. In the following section, we explain the thought process allowing for the definition of an RL problem. For a detailed description of RL and MDPs, we invite the reader to go back to Section 2.3.

An MDP can be condensed as a tuple (S, A, p, R) where:

- S is a finite set of states of the environment that the agent can observe,
- A is a set of actions the agent can perform,
- p is the system dynamics represented as a transition matrix between states,
- R is a set of rewards the agent can receive depending on the action taken and the consequent state of the environment.

Following the MDP formalism, an RL agent observes the state $\boldsymbol{s}$ of an environment and takes action $\boldsymbol{a}$, effectively updating the environment state to $\boldsymbol{s}'$. The agent receives in consequence a reward $\boldsymbol{r}$ measuring the quality of the new state the environment is in. The goal of the agent is to maximize this cumulative reward G_t, also called return, by learning a policy $\pi_\theta(a|s)$. This policy is parametrized by θ and produces an action $\boldsymbol{a}$ with respect to the state $\boldsymbol{s}$. Let J be the objective function of the RL agent, the agent objective at time t is defined as:

$$\begin{aligned} \arg\max_\theta J &= \arg\max_\theta(\mathbb{E}[G_t]), \\ G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots, \end{aligned} \tag{4.1}$$

where $\gamma \in [0, 1]$ is called the discount factor, used for convergence purposes.

Now falling back to our interaction protocol design problem, the environment of an RL agent modelling the interaction protocol would obviously be the application the user wants to interact with: the agent thus produces actions in this

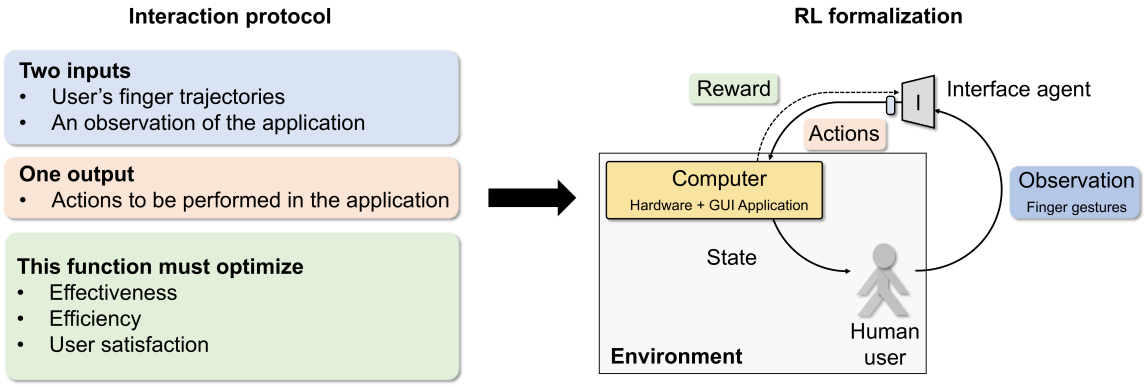

Figure 4.2 – Casting the problem of designing interaction protocols as a Reinforcement Learning problem.

application/environment. Defining its observation becomes more complicated: in a classical MDP problem, the agent directly observes a representation of the application, its state. In our case however, the agent must take action with respect to user gestures rather than the application state. This is illustrated in Figure 4.2. Because user gestures are not a direct representation of the environment state, our problem is currently impossible to model using a fully observable MDP background. This intermediate representation is problematic: the agent must observe and take action with respect to user gestures. These gestures are more correlated to the action to be performed than the properties of the environment.

If we relax the constraints of state observation, we can model our problem using a Partially Observable Markov Decision Process (POMDP) framework. POMDPs are a generalization of MDPs: instead of considering the observations of the agent as samples of the environment properties, we consider instead that the agent only receives partial information. A POMDP can be described as a tuple (S, A, p, R, Ω, O) where:

- the set of environment states S is not observable (in our case the virtual environment variables),
- the agent takes action in A (the possible actions to be performed in the environment),
- the environment dynamics are modelled through a transition matrix p (the probability to go from state s to s'),
- rewards from R depend on the new state (measuring the quality of the action taken),
- Ω is a finite set of observations perceived by the agent (all the possible user gestures),

- and O is the probability distribution describing the probability for the agent to observe $\boldsymbol{o}$ depending on the new state $\boldsymbol{s}'$ and the taken action $\boldsymbol{a}$ (in our case, this distribution can be interpreted as the user intent).

For an RL agent to solve a POMDP, it must build an additional internal representation of this distribution O. This is precisely what we want: **this distribution O can be seen as the user decision process to perform a gesture $\boldsymbol{o}$ with respect to the preceding agent action $\boldsymbol{a}$ and the following state of the application $\boldsymbol{s}'$**. In other words, for the interface agent to solve the problem, it must learn to model both the environment dynamics and the probability for the user to perform a gesture $\boldsymbol{o}$.

Completly modelling O as we said earlier is of course impossible: the decision process of a human user is extremly complex and it can vary depending on the user. Part of this distribution however may be modelled: when a use-case is simple enough, we can expect an almost deterministic choice of $\boldsymbol{o}$, e.g. in a "click on this button" type of query. For this assertion to hold, we need to consider during the training of the agent that **the user will systematically try to solve the task and produce optimal gestures**. This implies that there is a deterministic relation between O and p. Because the decision process of the user O is directly correlated to the solution of the task and depends on the environment dynamics, learning to model O means effectively solving the POMDP. While human users do not perform optimal actions every time, this setup still allows us to learn a realistic interaction protocol: this optimal user assertion can be seen as training sample selection (or denoising) and will not prevent a human user to use it sub-optimally afterward.

Our interaction protocol design problem is a special case of POMDP where O implicitly depends on p. Solving this problem is still a very challenging task: in practice, the agent might need to perform an action for each finger motion to act in real time. At time t, the observation $\boldsymbol{o}_t$ should be seen as the user's finger positions, $\boldsymbol{s}_t$ the current application state and $\boldsymbol{a}_t$ the corresponding action. A gesture is then a sequence of observations of arbitrary length. Because of these long-term dependencies, our model would need an internal memory to process the full history of states and actions to correctly model O.

Assessing the complexity of such a modelization for a novel application, we prefer at first to limit the model. We will consider user gestures in their simplest form, i.e. finger motion between two timesteps, or in other words two positions (start and finish) per finger. An observation $\boldsymbol{o}_t$ will be the stacking of these finger positions. In consequence, $\boldsymbol{o}_t$ will be the complete observation of a user gesture. This modelization gets rid of the realistic long-term dependencies but still learns to translate atomic user gestures to actions in the environment. With the deterministic relation between O and p, this POMDP can be solved in a similar fashion to fully-observed MDPs where $\boldsymbol{o}$ effectively replaces $\boldsymbol{s}$. To summarize, MDP-based RL can be used to learn an interaction protocol given:

- a task-based virtual environment,
- optimal atomic user gestures o_t,
- and a reward correlated to usability.

This last point is the topic of our next section.

4.1.2 Learning Signals and Environments

A major setting in the formulation of the RL problem is the reward function: after each action, what reward should be given to our agent? A good reward definition is mandatory as the agent will be trained to maximize its accumulated return. If the reward function is not carefully designed, our agent might converge to unexpected behaviors, oscillate without converging or just diverge (numerical instabilities).

In our case, the goal is to maximize the usability of a specific application. The International Organization for Standardization (ISO) defines the usability in ISO-9241 as "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use". Following this definition, our reward function should be designed to enforce policies maximizing these properties:

- **Effectiveness**: from the ISO-9241, effectiveness is defined as the "accuracy and completeness of user goal achievement". In the case of UIs, this is a performance metric measuring the capacity of a user to successfully perform a task while using a specific UI. Effectiveness is directly correlated to the achievement of a task and is in consequence easily measurable.
- **Efficiency**: from the ISO-9241, efficiency is the "resources spent by user in order to ensure accurate and complete achievement of the goals". It is usually measured as a function of the time spent by the user to complete a task. In our case, we can replace the notion of real time by the number of time steps required to solve an episodic task. This is again easily measurable. Special cases of efficiency such as ease-of-use can be derived from it by comparing for example first time-user efficiency with expert efficiency on a specific UI.
- **User satisfaction**: defined in ISO-9241 as "comfort and relevance of application". This satisfaction concept encompasses all the subjective appreciations of the user and is hard to explicit. W. J. Doll et al. 1988 define the user satisfaction as "the opinion of the user about a specific computer application, which they use". In other words, this notion condenses the user's perception of features like the visual aspect of the UI, its complexity with respect to the task or the optimality of the interactions. User satisfaction is obviously

correlated to effectiveness and efficiency but also depends on personal perception. In consequence, user satisfaction is not directly measurable, and this is problematic: it means that we cannot inject in the reward function the value we want to optimize on. We need to either find an estimate of this implicit notion or constrain our model so that optimizing on correlated metrics (effectiveness and efficiency) is enough to also optimize user satisfaction. The measurement of user satisfaction is a challenging topic still discussed in a variety of applications (W. Doll et al. 2004; Song et al. 2017; Machmud 2018; Al-Fraihat et al. 2019).

The most straightforward (and unbiased) approach to the reward design would be to ask human users to perform gestures and tell wether they are satisfied or not by the action the interface took. This optimal training setup is sadly not a viable strategy, because RL agents can take millions, even billions of steps before properly converging. Requiring human input at each timestep, either to craft the reward or perform gestures, is therefore unrealistic. This leads us to formulate two statements:

- not only **the reward function should be designed without human input**,
- but **the gesture selection also needs to be simulated**, at least for a subset of interactions.

In consequence, we need to model two human-related tasks: the measurement of user satisfaction corresponding to the interface behavior and the gesture selection with respect to the environment state. On a side note, using a simulated user still allows for the learning of co-adaptive interfaces, in two consecutive stages: a first offline training stage using a simulated user, meant to learn a pre-built interface usable by a human user, followed by an online tuning phase where real users use the interface and provide feedback to further adapt the interface to their expectations.

Currently, optimizing user satisfaction without direct human feedback is a seemingly impossible challenge: it would require to model all these implicit human perceptions. However, we focus here on learning interaction protocols, not the whole interaction technique. It means that we should only consider parts of the user satisfaction correlated to the interaction protocol. If we restrain the training process to specific tasks to be performed by the user ("move from A to B", "fetch this document"...), we make the assumption that the user satisfaction related to the interaction protocol is strongly correlated to performance metrics. The only subjective part is related to the type of gesture performed: some gestures are easier to perform and more intuitive depending on the action. For example, if we want to scroll down while reading a document, sliding one finger to the top feels very intuitive whereas tapping on the bottom less so. In this case, using a task-based environment is mandatory to train the interaction protocol: it makes

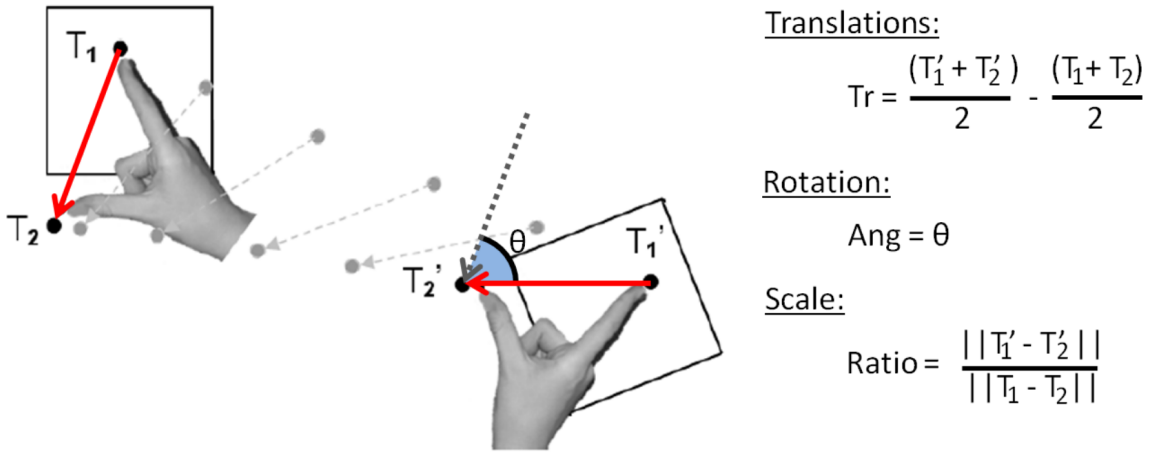

Figure 4.3 – Illustration taken from Brouet 2014. The square is manipulated using two-finger Rotate-Scale-Translate (RST) interactions. One motion can be interpreted as a combination of rotation, scaling and translation of the square.

measuring the performance easier, and better, it allows for the definition of an optimal performance. The downside to this approach is restricting training to situations with known outcomes and objectives. However, the learned interaction protocol can still generalize to other tasks and environments: as long as the actions to be performed are semantically equivalent (manipulating a document, navigating on a map...), the interaction protocol will be relevant.

4.2 Experiment: Learning a Touch Interaction Protocol

As a first proof of concept, we will attempt to solve a well-known interaction protocol for which a commonly accepted solution does exist, the goal being to verify whether ML can discover the existing solution. Our choice here is the protocol of the most commonly used Rotate-Scale-Translate (RST) multitouch technique for 2D object manipulation (see Figure 4.3). protocol widely used for smartphones and tablets. The name is a misnomer, since the interface does not only allow to zoom, but also to translate and rotate the content of surface through 2D gestures made by two fingers.

We suppose that a user performs gestures with exactly two fingers on a touch screen and we investigate the motion between two different time instants. We denote by $\boldsymbol{l} = [l_x\ l_y]^T$ the screen coordinates of a single finger at the first instant and by $\boldsymbol{l}' = [l'_x\ l'_y]^T$ the coordinates at the second instant. If we need to explicitly identify a finger, we will index finger i with a superscript as in $\boldsymbol{l}^i$ or $\boldsymbol{l}'^i$.

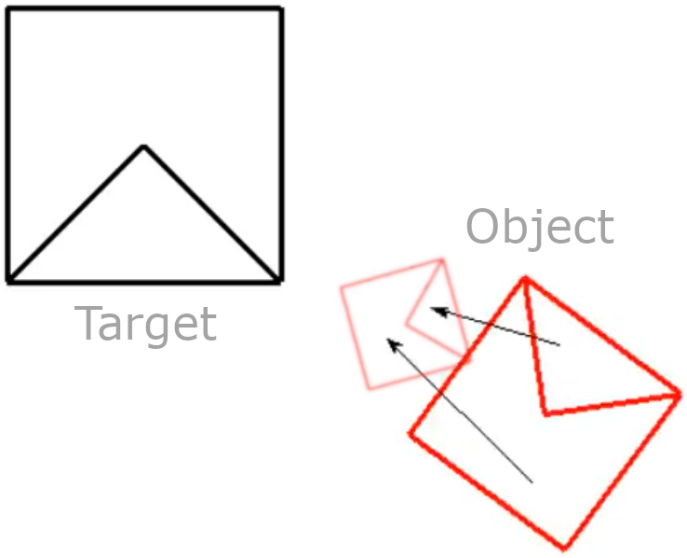

Figure 4.4 – 2D environment used to (re-)learn the RST protocol. The user must superimpose the red shape (object) onto the black one (target). User gestures are displayed as black arrows and the corresponding new position of the object is displayed in transparency.

4.2.1 Environment and known solution

We need to define an episodic task that requires the use of RST interactions. This can be done with a simple 2D manipulation task. Let there be an object on a 2D plane with position, scale and angle properties. Let there be a target with the same properties initialized differently. While the target is immovable, the user will be asked to move the object to match it with the target. The environment is displayed in Figure 4.4, with the object symbolized by a red shape and the target by a black shape. User gestures are displayed as arrows.

We assume from the current state of touch devices that the optimal interaction protocol for this task is the RST protocol: gestures performed by the user are interpreted as a combination of translation, rotation and scaling. Conceptually, the RST protocol can be interpreted as "a 2D finger motion on the screen induces the same 2D motion of the manipulated surface". In other words, the transformation applied to the segment drawn between fingers is the same that is applied to the manipulated object.

We will derive the analytical form of this known solution before describing the experiments learning it. The transformation interpreted from user gestures and applied to the object is a special case of affine transformation where the shear component is zero. It transforms coordinates $\boldsymbol{l}$ into $\boldsymbol{l}'$ as $\boldsymbol{l}' = \boldsymbol{A}\boldsymbol{l} + \boldsymbol{t}$ where

$\boldsymbol{t} = [\boldsymbol{t}_x\ \boldsymbol{t}_y]^T$ is the translation component and the rotation+scaling matrix A can be calculated from the rotation angle α and the scaling factor σ as follows:

$$\boldsymbol{A} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix} = \begin{bmatrix} \sigma\cos\alpha & -\sigma\sin\alpha \\ \sigma\sin\alpha & \sigma\cos\alpha \end{bmatrix}. \tag{4.2}$$

The 4 parameters of the motion are thus $\alpha, \sigma, \boldsymbol{t}_x, \boldsymbol{t}_y$, which we will combine into a parameter vector $\theta = [\sigma\cos\alpha\ \ \sigma\sin\alpha\ \ \boldsymbol{t}_x\ \ \boldsymbol{t}_y]^T$. If we have the motion of two different fingers $(\boldsymbol{l}^1, \boldsymbol{l}'^1)$ and $(\boldsymbol{l}^2, \boldsymbol{l}'^2)$, using homogenous coordinates, we can rewrite the linear relation of each finger position $l' = Al + t$ as $\boldsymbol{l}' = \boldsymbol{L}\theta$ where $\boldsymbol{l}'$ is a vector containing the last finger coordinates and $\boldsymbol{L}$ is a matrix containing the first finger coordinates in a suitable form. Given a pair of corresponding pairs $\boldsymbol{l}$ and $\boldsymbol{l}'$ for two fingers, we can express their relationship as follows:

$$\begin{bmatrix} l'^1_x \\ l'^1_y \\ l'^2_x \\ l'^2_y \end{bmatrix} = \begin{bmatrix} l^1_x & -l^1_y & 1 & 0 \\ l^1_y & l^1_x & 0 & 1 \\ l^2_x & -l^2_y & 1 & 0 \\ l^2_y & l^2_x & 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma\cos\alpha \\ \sigma\sin\alpha \\ t_x \\ t_y \end{bmatrix}. \tag{4.3}$$

If a user gesture is given, the RST solution corresponds to finding the $\hat{\theta}$ parameters of the transform to be applied to the object given gesture $([\boldsymbol{l}^1, \boldsymbol{l}^2], [\boldsymbol{l}'^1, \boldsymbol{l}'^2])$. Because $\boldsymbol{L}$ is always invertible (except for the degenerated case where both fingers are at the origin), the linear equation in Equation 4.3 can be solved easily as:

$$\hat{\theta} = \boldsymbol{L}^{-1}\boldsymbol{l}'. \tag{4.4}$$

This final vector $\hat{\theta}$ is the set of parameters of the affine transform to be applied to the object.

4.2.2 Handcrafted User Model

As stated earlier in this chapter, we need to simulate user gestures to have enough training data for our model. Because we already know the optimal solution for the interaction protocol, finding an analytic formulation of a user is pretty straightforward: let us reverse the problem and ask "what is an optimal user gesture given an object and a target position?". Because the transform applied to the object is the same transform applied to the segment between fingers from one instant to the other, we already almost solved this problem if we use Equation 4.3. Indeed, if we know the vertices of the object and the target, it is trivial to compute the corresponding parameters of the transform. Supposing we randomly select two finger positions on the object, we can then apply the known transform to these positions and compute their corresponding position on the target. The motion from the first positions to the second is then the optimal

gesture. The only consideration to have is that even an optimal human user would not solve the problem in one step: because of physical constraints, he would only perform part of this optimal gesture and move a certain distance in the direction of the target, the distance moved being proportional to the time spent between the two timesteps. In consequence, for realism purposes, we need to compute an intermediate optimal finger position as long as the object is too far from the target.

To formalize the above strategy, let us consider two object vertices v_1 and v_2 and their corresponding vertices v'_1 and v'_2 on the target. We consider that the user will move the object toward the target while keeping the positions v_i on the segments $[v_i,\ v'_i]$ (which is the optimal way to solve the task). It means we can find intermediate positions of v_i on these segments using the linear combination:

$$v_i^{inter} = (1-\mu)v_i + \mu v'_i\ ,\ \mu = \max(1, \frac{0.5}{||v'_i - v_i||_2}), \tag{4.5}$$

where μ can be interpreted as the user's gesture velocity. A small μ will mean small relative increments toward the target. μ is arbitrary and can be seen as a coefficient inversely proportional to the refresh rate of the device. This definition of μ goes in the sense that a human user will tend to do faster gestures while far from the target and slower, more precise gestures while close to it. Now that we have two points of the wanted intermediate object position, we can use Equation 4.3 to get the transformation of every point of the object to the intermediate position. At last, we can choose two random finger positions p_1 and p_2 on the object, transform them using the computed $\hat{\theta}$ parameters and build the two trajectories $[p_1,\ p_1^{inter}]$ and $[p_2,\ p_2^{inter}]$. The state $\boldsymbol{s}_i$ of the agent will be the concatenation of these two trajectories, resulting in a vector of size 8 (4 (x, y) couples).

4.2.3 Reward Function

Fundamentally, sparse rewards (+1 in case of success, -1 otherwise) offer the simplest form of reward and minimize the prior injected during credit assignment: in such a case, we simply ask the agent to complete the task without enforcing specific behaviors. If the sampling of the objective space done during rollouts is good enough – meaning that the agent perceives enough rewards to correctly evaluate the policy or the value estimator–, after some iterations the agent will find the optimal policy to solve the task. Sparse rewards are desirable because we do not indirectly supervise the agent, and in environments where the optimal policy is not known, the RL algorithm might find better behaviors that the one currently used by human specialists.

In our environment however, positive reward is extremly hard to perceive for untrained policies: at first, the policy will randomly move the object without

correlation with user gestures. Until it "luckily" moves the object on the target and gets a positive reward, it will not be able to start learning because the agent will never have had a positive learning signal to start evaluating good behaviors. This would lead to an extremly long training process until the agent starts matching the object with the target. To avoid this situation, we can help the agent and give it a reward at each timestep correlated to the quality of each action it takes: we change our sparse reward function to a dense one. An intuitive qualitative measurement for our problem is to use the distance between the object and the target: because we consider user gestures to be optimal, the distance between object and target should diminish at each timestep.

We can define our reward function in this task as the sum of L_1- and L_2-distances between the object vertices and the target vertices:

$$r = -\sum_i \left(||\boldsymbol{o}_i - \boldsymbol{t}_i||_1 + ||\boldsymbol{o}_i - \boldsymbol{t}_i||_2 \right) - 0.2, \tag{4.6}$$

where $\boldsymbol{o}_i$ and $\boldsymbol{t}_i$ are the coordinates of the $i-th$ vertice of the respective shapes (an alternative would be the Huber loss). Let us recall that the interface agent does *not* have access to these positions, or else it would learn to solve the task independently from user gestures. The constant -0.2 reward is set to continuously encourage fast solutions. A positive reward of $+25$ is given if the agent successfully finishes an episode. While these constant rewards are not mandatory for convergence, they help mostly during early stages of the training to find rewarding trajectories and avoid stalling close to the target position (because the distance is small). These values are chosen so that the expectation of the sum of rewards per episode is close to 0 for an agent close to the optimal solution.

4.2.4 Experimental Setup

For our episodic task, because there is no failure condition per se and object position are not limited, we must set a maximum number of steps per episode in order to avoid early returns to explode: because of the early random manipulation of the object, the agent might sometimes get the object far from the target. This can lead to highly negative return values and can make the optimization process unstable. We experimentally set the maximum number of steps per episode to 50: it has experimentally be observed as a good trade-off between discarding early bad trajectories and exploration limitation.

We use Deep Deterministic Policy Gradient (DDPG)(Lillicrap et al. 2015) to train the policy (actor) and a Q-function approximator (critic). Details of the algorithm are given in Section 2.3. In what follows, we call FCX a Fully Connected (FC) layer with X hidden units.

The actor is a fully connected network made of two hidden FC32 layers activated with ReLU, and a FC output layer activated with tanh. Layer normalization (Ba

et al. 2016) is applied to each hidden layer. The output of the actor is a vector of size 4: X-axis and Y-axis translation, scaling factor and rotation angle. We arbitrarily scale the amplitude of the outputs with respective coefficients 0.025, 0.025, 0.05 and 0.1 in order to produce coherent motion with respect to numerical values of the environment.

The critic starts with a FC32 hidden layer activated with ReLU, getting the state as an input. The action of the actor is concatenated to the output of the first hidden layer, and a second FC32 layer with ReLU is applied on this vector. The FC output layer has no activation and produces the Q-value estimate. Layer normalization is also applied to hidden layers. We observed that with smaller architectures the training of the agent was unstable, prone to divergence, whereas bigger architectures led the agent to sometimes be stuck in local minima. We apply L2-regularization on the critic weights.

We apply some adaptive parameter noise (Lillicrap et al. 2015) injecting noise directly in the actor weights rather than adding noise to the action to enforce exploration. Because the policy is stochastic, adding noise is theoretically not mandatory, but we observed faster convergence using this exploration method.

Hyperparameters – DDPG is very sensitive to hyperparameters. The agent needed proper tuning before being able to converge even on this simple problem. After experimentation, we found the following set of hyperparameters to allow for the fastest convergence:

- discount factor $\gamma = 0.99$
- target network update coefficient $\tau = 0.001$
- L2-regularization factor set to 10^{-2}
- critic learning rate $\alpha_c = 10^{-3}$
- actor learning rate $\alpha_a = 10^{-4}$
- batch size 2048 (but batch size from 1024 to 4096 led to similar results)
- noise parameter set to 0.1
- the episode is considered a success when $\sum_i ||\boldsymbol{o}_i - \boldsymbol{t}_i||_2 < 0.08$

The training session is split in two phases: a rollout phase to record data, and a training phase to update the agent. At first, the agent performs 100 rollouts in the environment. All the (s, a, r, s') quadruplets are saved in the replay memory. After the rollouts, the agent is trained using 50 batches sampled from the replay memory. The agent is then evaluated on one episode without noise. We call the combination of rollout, training and evaluation phase an epoch. The training is manually stopped when no improvement of the policy is observed.

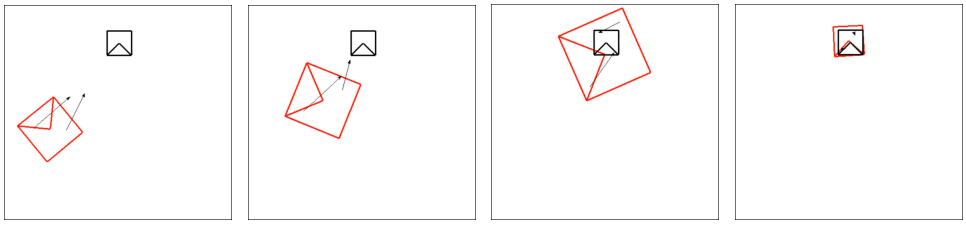

Figure 4.5 – An example rollout of the interface policy learned for the RST problem. Shapes need to be superimposed, finger trajectories are indicated by arrows. There are 5 steps passing between each image.

4.2.5 Results

We compare our trained agent to the optimal solution computed using Equation 4.4. This optimal solution is easily modelled as we defined both the analytic solution of a user and of the interface. On an average of 100 episodes, the optimal solution finishes an episode in 40 steps and obtains a reward of +0.5 per episode.

After a training of about 300k steps, the interface agent obtains very similar results: on an average of 100 episodes, it finishes an episode in 41 steps and obtains a reward of +0.4. It is visually impossible to separate the optimal solution from the learned agent. An illustration of a rollout in the environment is given in Figure 4.5.

4.3 Conclusions

We showed in this chapter that we are able to automatically re-learn an interaction protocol in controlled conditions: this is an important step toward both adaptive interaction protocols and the automatic design of such interaction protocols when the optimal solution is unknown. We learned in this chapter that human decision processes can be automated if their objective is properly defined: once usability is correctly measured, the modelling of both the interface and the user is feasible and the interface can be optimized using RL.

This RL problem is very challenging: real data cannot be used even though the learned protocol must be usable by humans, and the value we want to optimize on cannot be entirely explicited. Because we cannot effectively measure user satisfaction, we proposed a setup in which user satisfaction is strongly correlated to efficiency and effectiveness: in this case, we are able to optimize user satisfaction indirectly by optimizing on performance metrics. With our assumption of optimal user gestures, we were able to simulate a user and produce training data for the agent. However, crafting such a deterministic user is only possible if the interaction protocol solution is already known. Applications of this setup are thus

limited to re-learning existing solutions and eventually adapting them during real use.

In order to generalize to potentially unknown solutions, we cannot use a handcrafted user model, but we also cannot use real human users: we have to learn to synthesize coherent training data. The focus of Chapter 5 is to build knowledge representation of human user touch gestures in order to learn a user model. Such a model will be used in Chapter 6 to propose an entirely automated setup allowing for the learning of interaction protocols when the optimal protocol is unknown, i.e. when optimality may not be attainable.

CHAPTER 5

USER MODELLING

Contents

5.1 Natural Gesture Distribution Approximation 76
5.1.1 Generative Models for Distribution Approximation . . . 76
5.2 Recurrent Variational Auto-Encoder 78
5.3 Experiments 80
5.3.1 Architectures 80
5.3.2 Reconstruction and Generation Capabilities 82
5.3.3 Walking the Manifold Encoded in the Latent Space . . . 84
5.4 Conclusions 85

Chapter abstract

Because of the need for huge amounts of training data, user modelling is a necessity to produce complex and potentially unknown interaction protocols. However, providing a generic user model is obviously an overly complex task considering all the prior knowledge required. Before considering the selection process of which gesture to perform, we will learn what the properties of human touch gestures are.

In this chapter, we focus on learning this specific prior knowledge: we use a novel type of Variational Auto-Encoder (VAE) to automatically learn meaningful features from a touch gesture dataset, assessing their quality through reconstruction and latent space visualization.

Some of the work in this chapter is part of the following publication:

- Quentin Debard, Jilles Steeve Dibangoye, Stéphane Canu, and Christian Wolf (2019). "Learning 3D Navigation Protocols on Touch Interfaces with Cooperative Multi-Agent Reinforcement Learning". In: *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD)*.

5.1 Natural Gesture Distribution Approximation

As stated in the preceding chapter, learning interaction protocols whithout knowing the solution requires a user model to produce training data for our interface agent: it is unrealistic to use human data or to deterministically model the decision process of which gesture to perform. The question is how to learn a model of this decision process for an episodic task?

Deciding which gesture to perform requires at first to determine what choice there is, i.e. what gestures could be performed by a human user. These gestures must be intuitive to the user and respect some intrinsic human concepts: physical limitations, semantic correlation with the action to be performed in the environment and topological similarity between finger trajectories and spatial or temporal trajectories of the action.

A good interaction protocol will try to use the most intuitive gestures possible for a given action in order to maximize user satisfaction. However, the difficulty with interaction protocol design is that maximizing user satisfaction can be a competitive process to maximizing performance (effectiveness and efficiency). In complex applications, there is a trade-off between user satisfaction and performance, and the optimality of such a trade-off can be user-dependent. We expect that if our user model is good enough, our interface agent will naturally converge to an acceptable trade-off.

In order to build such a user model, we will focus at first in this chapter on learning "what a good gesture is", independently from a specific environment or task.

5.1.1 Generative Models for Distribution Approximation

If we consider the space of all the possible gestures that can be performed on a touch screen, our objective is to learn a subspace which corresponds to gestures naturally performed by humans. To this end, we suppose the existence of a training dataset of natural gestures $X = \{x_i\}$, which have been collected from user interactions. This training data can be collected without any manual annotation as simple interaction traces. These trajectories do not need to be correlated to the environment the agents will be working on, but a higher variability in the trajectories will give the user model more flexibility toward the definition of a solution. In our case, Itekube-7 (Chapter 3) is a good candidate for a varied dataset of human touch gestures.

Conceptually, we want the most intuitive gestures to be very likely to be used whereas more technical (but still natural) gestures are to be used in specific contexts. This is what we call the **natural distribution of touch gestures: we want to learn an approximation of this distribution**.

For a proof of concept, we will restrain ourselves to two-finger gestures. Let $x \in \mathcal{X}$ be an observation in the form of natural two-finger trajectories performed by a human user. We define an observation as an N-length sequence of 4-tuples, each 4-tuple consists of a pair of coordinates (x, y), one for each of the two fingers. Considering normalized positions, $\mathcal{X} = [0, 1]^{4N}$ is the space of observations of length N. The gesture space $\mathcal{X}$ thus covers all possible pairs of 2D trajectories, including trajectories which are anatomically impossible to perform by human fingers. Human gestures are a small subset of this observation space, parametrized by some latent properties.

We want to build a model capable of capturing latent properties producing any natural gesture x from a latent representation z. Sampling values from z should provide us samples of the manifold of natural human gestures. This involves learning a distribution $p(x|z)$ and to be able to evaluate it from a given z. Restricting our simulated user to produce samples of the latent representation z should therefore restrict it to produce natural gestures.

Several approaches exist for learning generative models of probability distributions from training data, among which are Generative Adversarial Networks (GANs) (Goodfellow et al. 2014) and Variational Auto-Encoders (VAEs) (Kingma et al. 2013). Our definition of $p(x|z)$ can be related to either the generative part of a GAN or the decoder part of a VAE. In this work, we chose VAEs for two reasons:

- they are simpler to train and less sensitive to hyperparameters,
- and the latent space dimensions are more independent, due to the soft constraint put on the prior to be close to a multivariate Gaussian.

This last point is in fact extremely important: it ensures that the representation of touch gestures we learn transition seemlessly from geometrically and semantically close gestures. This property is illustrated in Section 5.3 and will be put to use in Chapter 6.

The main drawback of VAEs is the fuzzyness of the generated samples: because the code is sampled from a distribution (see the following equations), the generated examples tend to display some natural noise that can be detrimental in a generation task. Our novel VAE architecture alleviates this problem by encoding all the gesture properties in the code and letting the "drawing" process entirely up to the decoder (see Section 5.3 for a comparison between architectures). With our architecture, the gesture properties will show some natural noise related to the sampling, but the drawing process of the gesture will be independent from the sampling process.

The VAE is trained on the dataset X, approximating the distribution $p_\theta(x)$ by measuring the reconstruction error on a sample x_i coded by an encoder E into a code z_i, then reconstructed into $\hat{x}_i$ using a decoder D. To describe the problem

from a probabilistic point of view, the probability $p_\theta(x)$ of a sample x can be decomposed into a prior and a likelihood as:

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz \tag{5.1}$$

where the prior on z is defined as a standard Gaussian distribution $p_\theta(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. This choice of distribution is common in VAEs (Kingma et al. 2013). In our case (continuous values), we can also impose that the likelihood is Gaussian distributed:

$$p_\theta(x|z) = \mathcal{N}(x|D(z, \phi_d), \sigma^2\mathbf{I}) \tag{5.2}$$

where $D(z, \phi_d)$ is the decoder of the VAE. The integral is difficult to evaluate, but can be approximated by a point estimate $z{=}q_\phi(x)$ from the variational distribution q:

$$p_\theta(x) \approx \mathcal{N}(x|D(q_\phi(x), \phi_d), \sigma^2\mathbf{I}) \tag{5.3}$$

$q_\phi(z|x)$ can be seen as an encoder, noted $E(x, \phi_e)$. In this case, we need to ensure that $q_\phi(z|x)$ is a good estimate of the true posterior $p_\theta(z|x)$. This is done using the Kullback-Leibler divergence, noted D_{KL}. Considering the approximation error for only a sample x_i, the KL divergence becomes $D_{KL}(E(x_i, \phi_e)||p(z))$. As stated earlier, $p_\theta(z) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. If we use the L_2-norm to measure the reconstruction error, the total error can be written as a variation of the evidence lower bound (ELBO) (Kingma et al. 2013):

$$ELBO_i = ||x_i - D(E(x_i, \phi_e), \phi_d)||_2 - \beta\ D_{KL}(E(x_i, \phi_e)||\mathcal{N}(\mathbf{0}, \mathbf{I})) \tag{5.4}$$

where β is a parameter allowing us to adjust the tradeoff between the reconstruction precision and the latent space regularity (Matthey et al. 2017). Ideally, this parameter should be high enough for our latent distribution to get close to a standard normal distribution. This would ensure the smoothness of our latent space and favor semantic interpretations of the latent code z (see Section 5.3.3). In practice, the complexity of this distribution and the limitations of our models would cause the VAE to make poor reconstructions with such constraints. This requires us to lower β to allow some distorsion of $q_\phi(z|x)$ and adjust the reconstruction/regularity tradeoff. We can then update ϕ_e and ϕ_d by minimizing this error using backpropagation.

5.2 Recurrent Variational Auto-Encoder

Usually with VAEs, the decoder network produces fixed length outputs in order to reconstruct the input. We can find encoder and decoder architectures using Fully Connected Network (FCN), Convolutional Neural Network (CNN) or Recurrent Neural Networkss (RNNs) in the literature. In our case, classically using

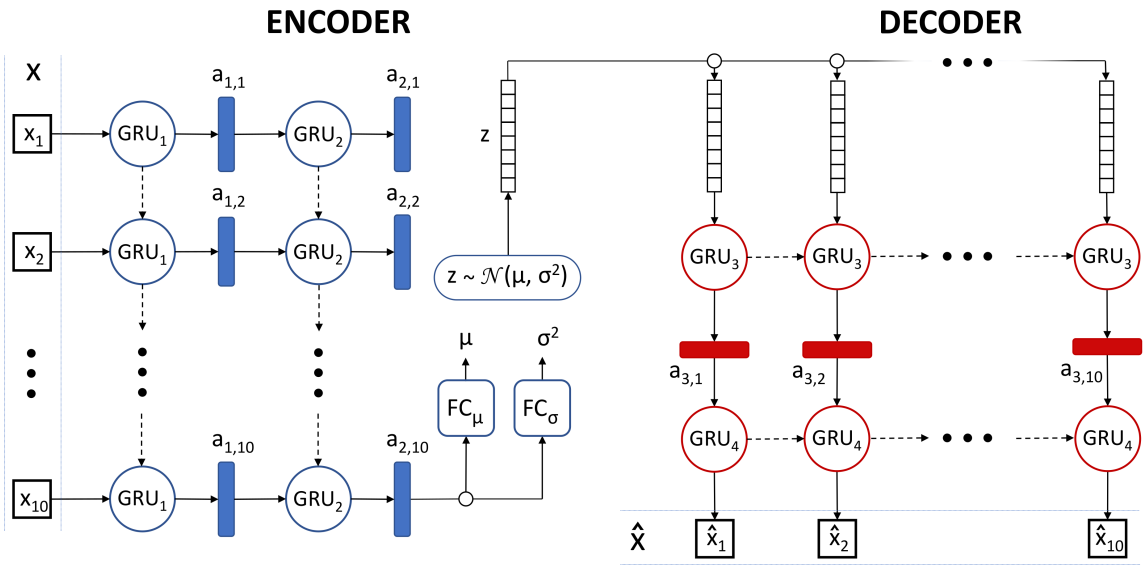

Figure 5.1 – Architecture of the sequential VAE. The encoder in blue processes input x to produce the latent code z by sampling it from a Gaussian distribution, then the decoder in red reconstructs the input $\hat{x}$ from z. x can be a sequence of arbitrary length, where x_i is a tuple containing the finger positions at time i.

FCN or CNN produced poor, fuzzy reconstructions, not suitable to make satisfying human-like gestures (see Figure 5.2). At one point, we made an important observation: considering the sequence of finger positions that is a gesture, if we are observing a continuous gesture, the semantic remains the same throughout the sequence (rotation, translation...). In other words, the meaning of a gesture is independent from its duration, although the parametrization might differ (rotation angle, scaling factor...). We chose in consequence to encode the semantics of the gesture in the code and leave the unrolling of the gesture through time to the decoder: this is the reason behind the architecture of our sequential VAE (Figure 5.1). This architecture allows us to:

- get rid of the fuzzy reconstructions by moving the sampling process to gesture parameters,
- and be able to unroll the gestures to an arbitrary length, effectively generating shorter or longer sequences of a semantically equivalent gesture.

The encoder is made at first of a Gated Recurrent Unit (GRU) unrolling on timesteps. The last activation of this GRU is processed by two different Fully Connected (FC) layers producing the mean and the variance of the latent Gaussian distribution, from which the latent code z is sampled. The decoder then uses the full code z **at each timestep** to produce the next finger positions, using its internal

memory to unroll the features encoded by z to the needed length. This allows our VAE to treat not only input of varying length, but also produce outputs of varying length during inference. This can be used to reconstruct missing or noisy datapoints, and in our case to produce gestures of the required length.

This architecture drastically reduces the natural fuzziness of VAE outputs compared to static architectures: while the semantics encoded by z are noisy by nature, the sequence is generated from the unrolling of the decoder, and these temporal properties can be learned with low dependence from z because z is entirely observed at each timestep. Basically, the dependence of temporal features to z is reduced, and this helps in reconstructing closer gestures while maintaining the properties encoded by z.

5.3 Experiments

We use a subset of Itekube-7 (introduced in Section 3.1.1) to train every VAE, using gestures from the translation, pinch and rotation classes. All gestures are sampled to 10 datapoints using dynamic sampling (Section 3.1.2). In consequence, a gesture is represented as a tensor of size $(10, 2, 2)$ containing 10 datapoints, each datapoint being two (x, y) couples. The latent code size is set to 8. All VAEs are trained for 50 epochs, a batch size of 128 and a learning rate of 0.005. We linearly increase the value of β from 0 to its maximum value during the first half of the training. This technique is known as warm-up (Sønderby et al. 2016) and is used to prevent the early inactivation of latent units observed in variational free energy minimization models (MacKay 2001).

5.3.1 Architectures

We will discuss in this section the results obtained with 3 different architectures based on different features: a fully connected, a convolutional and our recurrent VAE.

The hyperparameters for each architecture were optimized from experience. Because of their important number, we will not display this grid search process. If not explicitly stated otherwise, the default architectures used for these experiments are the following:

Fully Connected VAE – The input is flattened to a vector of size 40. The encoder is made of two FC layers of size 128 activated with ReLU. Two distinct FC layers produce the respective mean and variance for a multivariate Gaussian from which the code z is sampled from. The decoder is made of a FC layer of size 128 activated with ReLU, and a following FC of size 40 with no activation producing the reconstructed gesture. The output is then reshaped to match the initial shape $(10, 2, 2)$.

Convolutional VAE – The input is reshaped to $(10, 4, 1)$. It means that convolutions are applied in a spatio-temporal fashion, and we consider in consequence a unique channel. No padding is used to apply convolutions, and batch normalization (Ioffe et al. 2015) is applied to every convolutional layer (conv layer) and transposed convolutional layer (trans conv layer) besides the last one. ReLU is used as the activation function for every layer except the last trans conv one, which has no activation function. The encoder is made of 3 conv layers and a max pooling:

- a conv layer of kernel size 3x2 and a 1x2 stride producing 64 feature maps,
- a max pooling of kernel size 2x1 and stride 2x1,
- a conv layer of kernel size 3x2 and a 1x1 stride producing 128 feature maps,
- and a conv layer of kernel size 2x1 and a 1x1 stride producing 256 feature maps.

The output of this last conv layer is a vector of size 256. This vector is processed by two FC layers to produce again the mean and variance of the latent distribution. The FC layer producing the variance is activated with the softplus function to forbid incoherent values. Each latent value of z is then seen as a channel for the decoder, i.e. the input size of the decoder is $(1, 1, 8)$. The decoder is then:

- a trans conv layer of kernel size 3x1 and a 1x1 stride producing 256 feature maps,
- a trans conv layer of kernel size 3x2 and a 1x1 stride producing 128 feature maps,
- and a trans conv layer of kernel size 6x1 and a 1x1 stride producing 4 feature maps.

This last trans conv layer produces vectors of size 10, so, after reshaping, we get our 40 values to reconstruct the gesture. This upsampling process was motivated by two remarks: first, put the emphasis for the decoder on temporal features rather than spatial ones, falling back to the concept of semantic-encoding latent space, and second this uses way less weights with respect to the number of features than reconstructing the gesture in the convolution domain.

Sequential VAE – The architecture was detailed in the last section. As for hyperparameters, the two GRU layers of the encoder have a hidden state of size 128. The first GRU layer of the decoder has a size of 256, and its second layer of course as a hidden state of size 4 to generate a datapoint.

Architecture	Reconstruction loss	KL-loss	Total Loss
FC	0.42	39.34	0.82
CNN	0.74	41.26	0.95
RNN	1.03	13.12	1.95

Table 5.1 – Comparison of best performing models for each architecture.

5.3.2 Reconstruction and Generation Capabilities

While VAEs are trained to minimize the combination of the reconstruction loss and the KL-loss, this numerical value alone is not sufficient to assess the quality of a model; visual results should also be taken in consideration. There are two main reasons: the $L2-$norm is not perfect to describe the distance between two gestures, as gestures with close spatial properties can have different meanings, and the encoding of visually poorly correlated features by the VAE is also a desirable quality. This property emphasizes on a natural semantic disentanglement of latent features.

The numerical results for each architecture are displayed in Table 5.1. These results were obtained using a β of 0.01, 0.005 and 0.07 respectively for the FC, CNN and RNN architectures. This choice of β was motivated by the reconstruction quality, the disentanglement and expressiveness of latent features and the quality of generated samples by each VAE. It is important to note that the optimization problem gets more complicated with the increase of β. This makes sense, considering we are asking the model to solve the same reconstruction task while constraining even more the topology of the latent space. In consequence, results in Table 5.1 have to be taken with a grain of salt. From this table, we can see that the sequential VAE is seemingly the worst performing of all 3 with respect to the global loss. As we just said, this can be explained by the bigger β: the sequential VAE is in fact tackling a more complex problem. If we train the FC and convolutional VAE with the same β, they are converging to a similar reconstruction loss (even smaller), but their latent space and reconstructions are worse. Another important observation is the KL-loss, smaller for the sequential VAE. This means that its latent space will exhibit better topologic properties. As stated in the beginning of this chapter, this will be very important for our user model.

An illustration of the differences between architecture reconstructions is given in Figure 5.2. Both the convolutional and the fully connected VAE tend to reconstruct closely the gesture: reconstructed datapoints are close to their original counterpart. We can see however artifacts in the reconstructions, making it easy to distinguish from real gestures: the convolutional and the fully connected VAE did not learn the inherent smoothness of human gestures and seem to just try to fit datapoints.

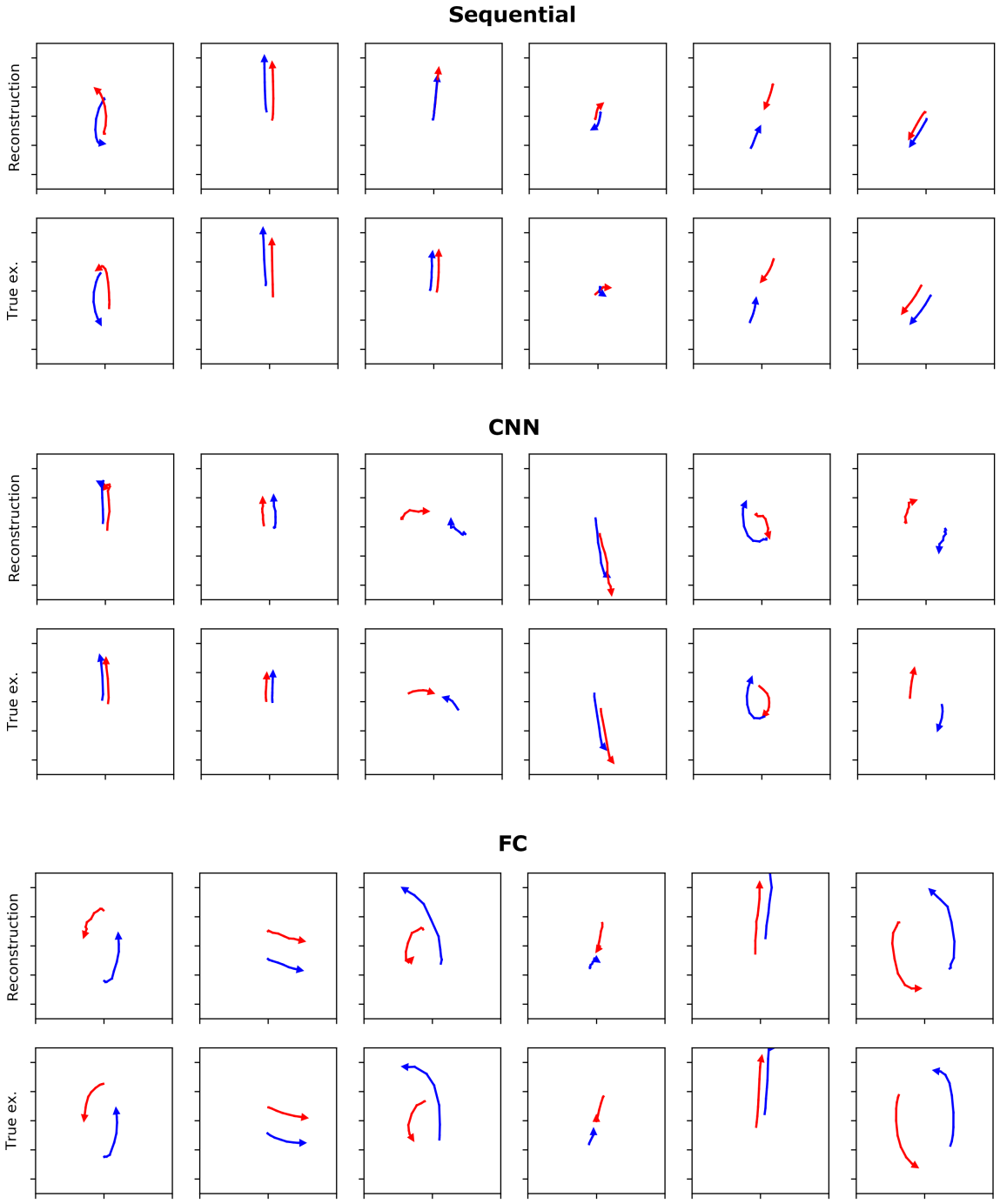

Figure 5.2 – Reconstruction examples using the different VAEs. For each architecture, the first line displays reconstructed gestures while the second displays the corresponding original gestures.

The sequential VAE displays different properties: its reconstructions are smooth, but sometimes tend to get further from the original gesture than the other VAEs. So, considering the $L2-$distance between datapoints, its reconstructions are worse. However, these reconstructions maintain the semantic of the original gesture while displaying satisfying geometric properties: this is in our case a more desirable property than straight close reconstruction in the $L2-$distance sense.

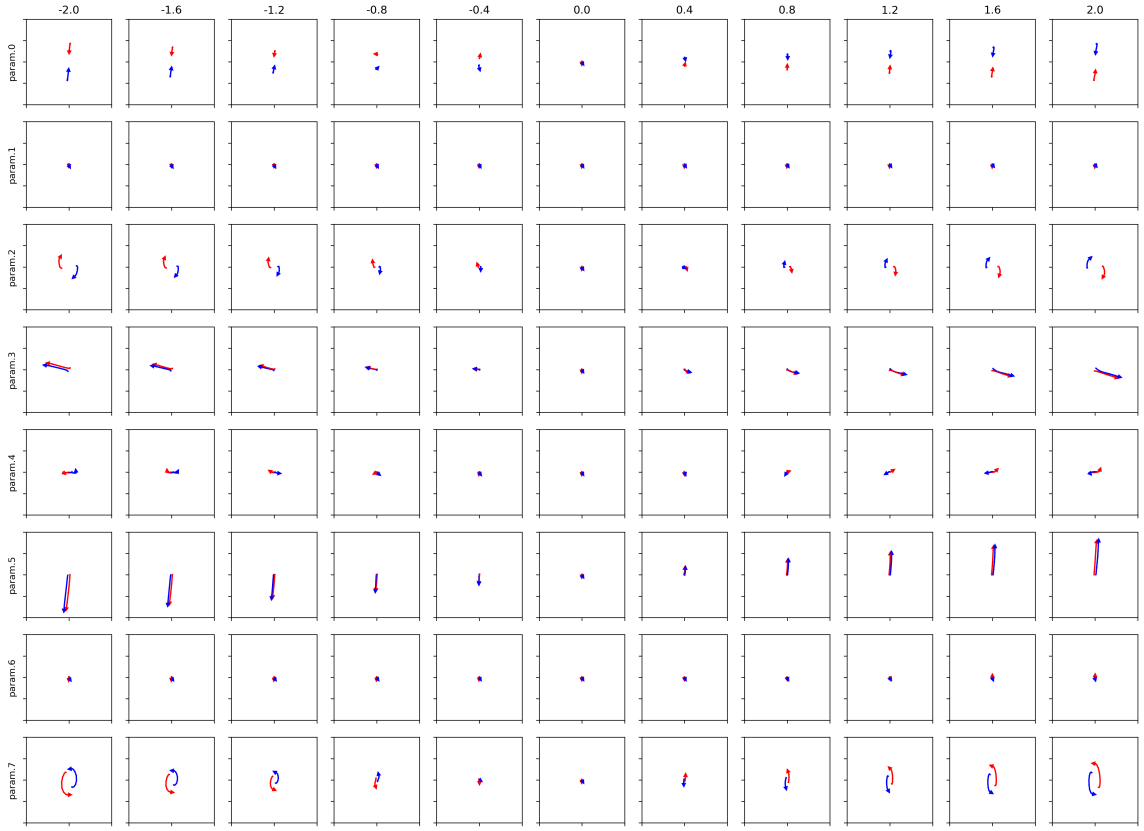

Figure 5.3 – Navigating the latent space learned by the VAE by displaying gestures generated using different codes z. We set the default code to all zeros (middle column). We then change the values of one variable at a time in the range $[-2, 2]$. So, any code used in this graph will have at most one non-zero value. Lines correspond to different modified latent variables (dim=8). Columns correspond to different values.

5.3.3 Walking the Manifold Encoded in the Latent Space

Observing the reconstruction capabilities of the VAEs is more of a sanity check rather than a proper estimation of the VAE quality. In order to really assess the quality of the learned latent space, we must evaluate its topological properties. A smooth latent space with poorly correlated latent variables is desirable if an

Reinforcement Learning (RL) agent is to use this latent space to take action: its exploration of the latent space will be faster and local features it may learn during training will generalize more easily to different parts of the problem. To illustrate this second statement, lets consider the continuity between code values and properties of the corresponding generated gesture. If for a certain code value, a gesture is generated, we expect small changes made to the code (say the increment of a latent variable) to induce small changes to the generated gesture. This property is enforced by the KL constraint onto the latent distribution.

We illustrate the sequential VAE latent space in Figure 5.3. Once trained, we feed handcrafted codes to the decoder and observe the generated gestures. Because we want to observe the features encoded by each latent variable, we set at first the code to all zeros, then we alter the value of a latent variable one at a time. The results displayed in Figure 5.3 are very satisfying: we can see that most latent variables are encoding semantically meaningful properties. The first line corresponding to variations of the first latent variables encode clockwise rotations around an axis. The 5th and 8th variables encode translation-like gestures on a different axis. So, the latent space is not only smooth, but also decently disentangled, meaning individual latent variables encode particular and semantically meaningful properties.

In order to not overload this thesis with similar graphs, we will not display the corresponding latent space visualizations of the fully connected and the convolutional VAE. The properties of their corresponding latent space was similar: while some slight disentanglement could be observed for translation-like gestures, most of the generated gestures were semantically correlated. These gestures also shown the same "fuzzy" artifacts that can be observed on the reconstructed gestures shown in Figure 5.2.

5.4 Conclusions

We proposed in this chapter a modelization of the natural gesture space using a novel VAE architecture to automatically find a low dimension parametrization of this space with specific topologic properties. This will allow us in the next chapter to build a user model using this pre-trained VAE in order to produce data and learn complex interaction protocols.

While applied to touch gestures in this thesis, we believe this sequential VAE could see some applications in sequential data processing, for tasks such as the reconstruction of an incomplete or a noisy signal, thanks to its varying input and output length.

CHAPTER 6

LEARNING COMPLEX CO-ADAPTIVE USER INTERFACES

Contents

6.1 Multi-Agent Reinforcement Learning Setup 88
6.1.1 User Model 88
6.1.2 Formalization 91
6.2 Experiment: Learning a 3D Navigation Interaction Protocol . . . 93
6.2.1 Environment 94
6.2.2 Experimental Details 95
6.2.3 Results 96
6.3 Conclusions 97

Chapter abstract

In Chapter 4, we have cast the interaction protocol design as a Reinforcement Learning problem and validated this approach on a problem with a known solution. In Chapter 5, we have built representations of natural gestures using unsupervised learning.

In this chapter, we will combine these works to propose a Multi-Agent Reinforcement Learning setup learning simultaneously a user and an interaction protocol model. We combine in a novel fashion Reinforcement Learning and Variational Auto-Encoders to craft a user model: this allows us to synthesize training data for the interface agent without the need for human input. This user modelling is mendatory for training interaction protocols with unknown solutions because no optimal user can be analytically defined.

We will experiment on a 3D navigation environment, learning to navigate in the 3D space to attain a target position. Along with the difficulties discussed in Chapter 4, solving this problem will also require us to solve an ill-posed mapping problem with no known optimal solution.

Work in this chapter is part of the following publication:

- Quentin Debard, Jilles Steeve Dibangoye, Stéphane Canu, and Christian Wolf (2019). "Learning 3D Navigation Protocols on Touch Interfaces with

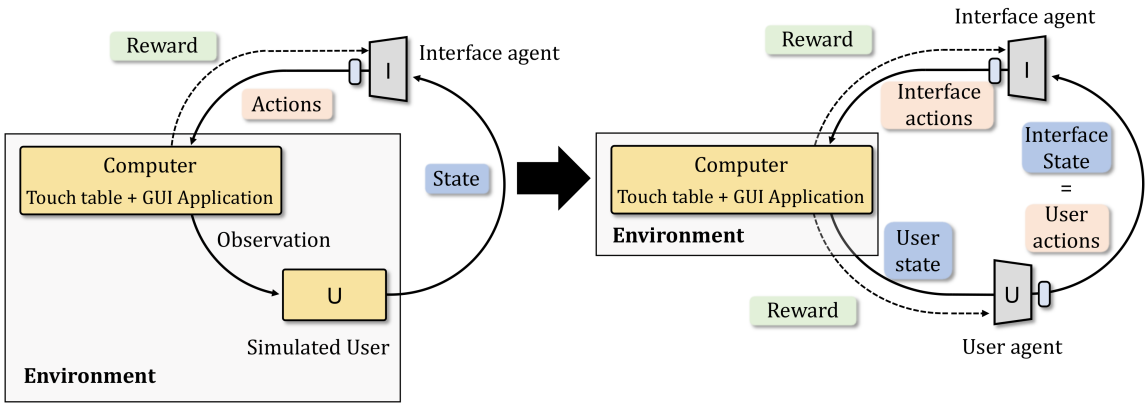

Figure 6.1 – Intuitive casting of the human user as a RL agent for automatic interaction protocol design. The user agent observes the environment and produces gestures while the interaction protocol agent interprets this gesture to produce the corresponding action in the virtual environment.

Cooperative Multi-Agent Reinforcement Learning". In: *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD)*.

6.1 Multi-Agent Reinforcement Learning Setup

We now come back to learning our interaction protocol by modelling it with a Reinforcement Learning (RL) agent. In Chapter 4, we proposed a setup allowing for the automatic learning of already known interaction protocols. This method was limited by the need for a simulated user to synthesize gestures as training data for our agent: if the solution is unknown, we cannot analytically model user gestures. The impossibility to train our interface agent from real data and the unknown nature of an optimal solution compels us to learn a user model in order to generate training data. Following the casting of the interaction protocol as a RL agent (see Section 4.1.1), this user model should be able to produce coherent human-like gestures for each state of the environment.

6.1.1 User Model

A simple intuition for such a user model could be to also use a RL agent to model the user. This is illustrated in Figure 6.1. However, a naive implementation of this solution does not work: without any proper constraint between the user and the interface agent, there is no incentive for the agents to communicate using human-like gestures. They will converge to simpler modes, the simplest one

being the regression of one policy to an identity, letting the other agent solve the problem by directly observing and taking action in the virtual environment. This means that if we are to use a RL agent to take decisions as a user, we need to constrain it to do so using human-like gestures.

As stated in Chapter 4, the goal of our interaction protocol is to maximize usability, decomposed in three components: effectiveness, efficiency and user satisfaction (more details in Section 4.1.2). For our user model to produce the required training data, it needs to select an appropriate gesture given an environment state at each timestep. By appropriate, we mean:

- intuitive, allowing for the interface agent to maximize user satisfaction,
- and coherent with the current environment state and task, allowing for the maximization of efficiency and effectiveness.

The coherence of gestures can be estimated in goal-driven environments and is supposed to be the same for every user: the fastest way to successfully perform a task is optimal in the sense of efficiency and effectiveness. Intuitivity however depends on a multitude of conditions: physical constraints, knowledge transfert, personal preferences... Satisfying those conditions effectively increase the consequent user satisfaction of an interaction protocol. This is the most difficult part of the interaction protocol design because user satisfaction is not directly measurable while being extremly important to the overall quality of an interface.

Training a model to learn what gestures to perform with respect to the environment state seems hardly feasible in a supervised fashion: we fall back to the problems of the absence of measurable values to optimize on and the limited amount of available data. There is however information to learn from observing unlabelled gestures: the range of spatial and temporal properties of human-performed gestures. Collecting unlabelled gestures is way less resource- and time-consuming: simple interaction logs are enough. Modelling these latent properties may allow our model to learn constraints to generate human-like gestures. Once this generative model is trained, we must also model the decision process of which specific gesture to produce. In consequence, we designed a two-part user model:

- one part of this model is trained to learn human gesture properties from a limited interaction set using unsupervised learning. The training of this part was described in Chapter 5: we used a Variational Auto-Encoder (VAE) to generate data from Itekube-7. In the process, the VAE encoded gesture properties in its latent space. If we discard the encoder of our VAE and use the decoder, we can choose to generate arbitrary gestures by manually inputting a latent code.
- The other part of our user model will learn to produce a coherent latent code with respect to the current environment state. This part is modelled by a RL

agent. To summarize, our user model will be a RL agent producing codes interpreted by a trained decoder, soft-constraining our user model to human-like gestures; *the policy of the user agent will learn to produce gestures by navigating the latent space of the VAE*.

Formally, like the interface agent, the user RL agent will have to solve a Partially Observable Markov Decision Process (POMDP) (see Section 4.1.1 for a detailed description). The cause however is different:

- the interface agent perceives an observation o_i in the form of user gestures while it takes action in the virtual environment. For the problem to be a fully observed Markov Decision Process (MDP), the interface agent would need to directly observe a state of the virtual environment. Because we chose not to model long-term dependencies (processing pairs of time instants independently from the past), this special case of POMDP can be solved like a fully observed MDP, without internal memory.

- The user agent directly observes a state of the virtual environment, but its action, i.e. the choice of code to be interpreted by the decoder as a gesture, does not directly impact the environment. Instead, after being produced by the decoder, the gesture is interpreted by the interface agent. In consequence, the transition probability $p(s'|s, a_u)$ is not fixed and depends on π_i: such a case breaks the MDP framework. Instead, we can consider that p is fixed, but there is a probability distribution O for the agent to observe o_u with respect to a_u and s'. In our case, O is in fact entirely parametrized by π_i. We can note that instead of a classical POMDP, the set of states S in this case is equal to the set of observations ω. Again, because we do not consider long-term dependencies, this POMDP can be solved like a fully observed MDP.

As stated in Chapter 5, the most used gestures in Itekube-7 (statistically the most intuitive) are close to an all-zeros code, the all-zeros code being the absence of finger motion. The corollary is that unusual or hard-to-perform gestures are coded with high values: it means that we can directly control how unlikely we can authorize the gestures of our user model to be by limiting the norm of the code produced by the RL agent. In consequence, our user model will not be able to produce every humanly possible gesture; it will however be able to use a consequent range of human-like gestures and will tend to choose likely gestures while being able to measure semantic differences between them directly from the code values. Because the interface agent will need to learn to interpret these gestures from a semantic viewpoint, the more different the gestures are for different actions, the easier its convergence will be: well-defined gestures for specific actions will be a preferred convergence mode over less defined boundaries.

6.1.2 Formalization

We formulate the task of jointly learning the interaction protocol and a user model as a cooperative Multi-Agent Reinforcement Learning (MARL) problem, as shown in Figure 6.2. Two agents are learned all together, each with its own policy and Q-value estimator:

- We call A_i the agent learning the interaction protocol. Learning its policy π_i is the original goal of this work, as this agent is responsible for translating 2D finger gestures into actions in a virtual environment. Its observations noted as o_i are the gestures produced by the user model during training or by a human user during real use. The agent A_i takes action a_i in the virtual environment with respect to the gesture produced. The reward r perceived is shared with the user agent A_u.

- The user model U models a user performing a task in a virtual environment, using the interaction protocol defined by the policy of A_i. Its purpose is to replace human users during the costly pre-training phase of A_i by feeding coherent training data to the interface agent. It is made of a RL agent A_u that observes a representation of the environment o_u and outputs a code a_u interpreted by a pre-trained VAE decoder D to produce gestures. We note this $o_i = D(a_u)$.

These A_u and A_i agents are trained to maximize the same objective function, sharing the same reward r, which makes this problem a *cooperative* MARL problem. It is however an unusual MARL problem: only A_i directly takes action in the virtual environment, whereas A_u acts indirectly by producing the input of A_i. Both agents need to learn good behaviors for the setup to converge to a good solution. Because the decoder is fixed, the agents will have to learn to communicate using gestures.

The method can be more formally described as follows. The task is a sequential cooperative MARL setup where A_u produces the observation of A_i and A_i does not get any observation of the virtual environment. In what follows, we denote $o_\cdot^t$ as the observation of an RL agent at time t, $a_\cdot^t$ as the action at time t and $r_\cdot^t$ as the resulting reward from action $a_\cdot^t$. $\pi_\cdot$ will denote an agent policy. All symbols are indexed by subscripts u or i, which stand respectively for the agent A_u and A_i. Let s^t be the state of the software environment at time t, for instance the viewpoint in a building or the 6D pose of a mechanical object in a computer-assisted design

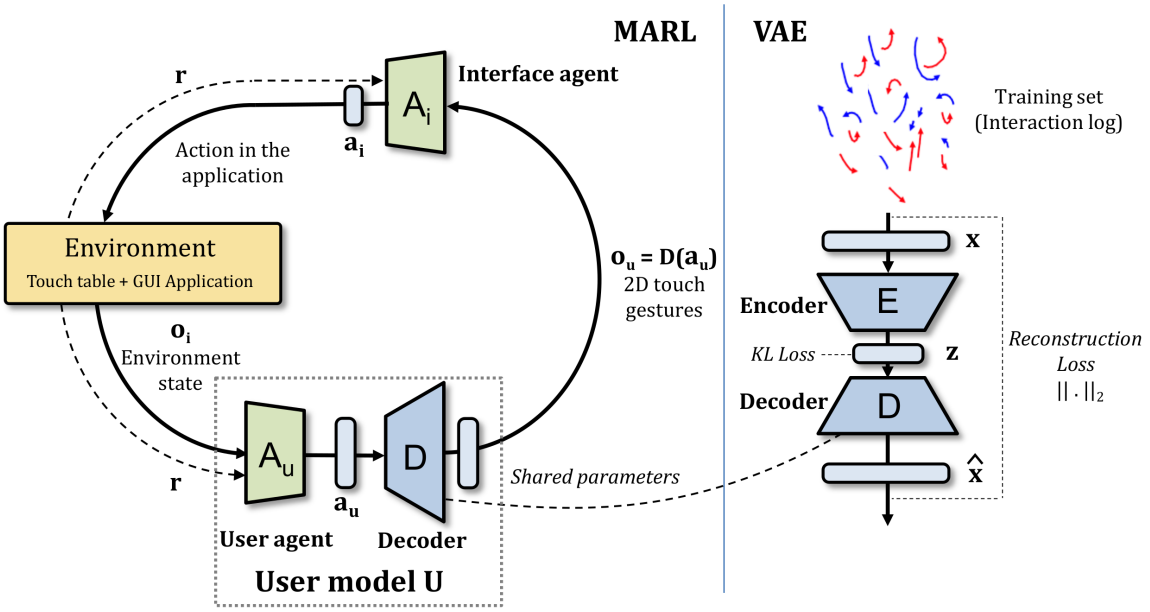

Figure 6.2 – Cooperative MARL problem for jointly learning the interaction protocol and a user model. Generative model parts are blue and RL agents are green. The user agent A_u observes the environment and produces a code interpreted by the decoder D as a user gesture. This user gesture is observed by the interaction protocol agent A_i which consequently produces an action in the environment. At the end of this iteration, both agents receive a reward r strongly correlated to efficiency and effectiveness.

problem concatenated to the target position. Then, a given time step t in our sequential cooperative MARL setup will unroll as follows:

$$\begin{aligned} o_u^t &= s^t, \\ a_u^t &= \pi_u(o_u^t), \\ o_i^t &= D(a_u^t), \\ a_i^t &= \pi_i(o_i^t), \\ o_u^{t+1} &= s^{t+1}, \\ r_u^t &= r_i^t = r(a_i^t, s^{t+1}), \end{aligned} \tag{6.1}$$

where $r_u^t{=}r_i^t$ is the joint reward at time t computed using the reward function r. For convergence purposes, these agents must be trained sequentially: when an agent updates its policy, the other is just used during the rollout phase to collect training data by letting the agents take action in the environment.

To clarify the usage of our MARL setup for learning adaptive interaction protocols, we are going to detail the complete processing chain of our adaptive interaction protocol design, from untrained models to human use:

1. At first, a VAE is trained on a set of unlabelled human gestures to approximate the natural gesture distribution. This process is described in Chapter 5. The goal here is to automatically learn human-like constraints on touch gestures to be used by our user model. Once trained, the encoder is discarded and the decoder is used as part of the user model.
2. Once the VAE is trained, we can use our MARL setup to learn a first version of the interface agent. This is a generalization of our work in Chapter 4, allowing for the learning of potentially unknown interaction protocols. From a human perspective, this phase can be seen as a pre-training phase: this automated training process is not meant to fully replace human supervision, but it allows for the crafting of an interaction protocol usable by humans without using human input. It is important to note that depending on the initialization of the pre-training phase, the resulting interaction protocol might be substantially different.
3. After the pre-training phase, the interaction protocol can be further tuned from a few real interaction samples performed by specialists: this tuning step can be used to alleviate unwanted biases or enforce a specific behavior. The reward function will be different (and sparser) as we cannot expect human users to be optimal at all times. At the end of this phase, we will have designed a generic interaction protocol, only requiring human input during the fine-tuning of the interaction protocol.
4. At last, the model can be used by the end user and will continue adapting from use to further maximize user satisfaction by using the same reward function as specialists. In practice, these interfaces could end up taking the form of personal user profiles, saving interface models for specific applications.

During this thesis, we did not have enough time to implement the human fine-tuning of the interface (step 3-4); we will in consequence discuss our work on the pre-training phase (step 1-2) in the following.

6.2 Experiment: Learning a 3D Navigation Interaction Protocol

In order to test the pre-training phase of our setup, we defined a navigation task in a virtual 3D environment. Compared to our experiments in Chapter 4 on re-learning the Rotate-Scale-Translate (RST) protocol, there is no uniquely accepted solution: mapping 2D gestures to 3D actions is an ill-posed problem. In consequence, there is no evident transform to interpret user gestures and design choices must be done. To this end, we extended the 2D toy problem to 3D, maintaining the user's goal of moving a (now 3D) content to superimpose an object over a non-moving object, as shown in Figure 6.3. As there is no simple

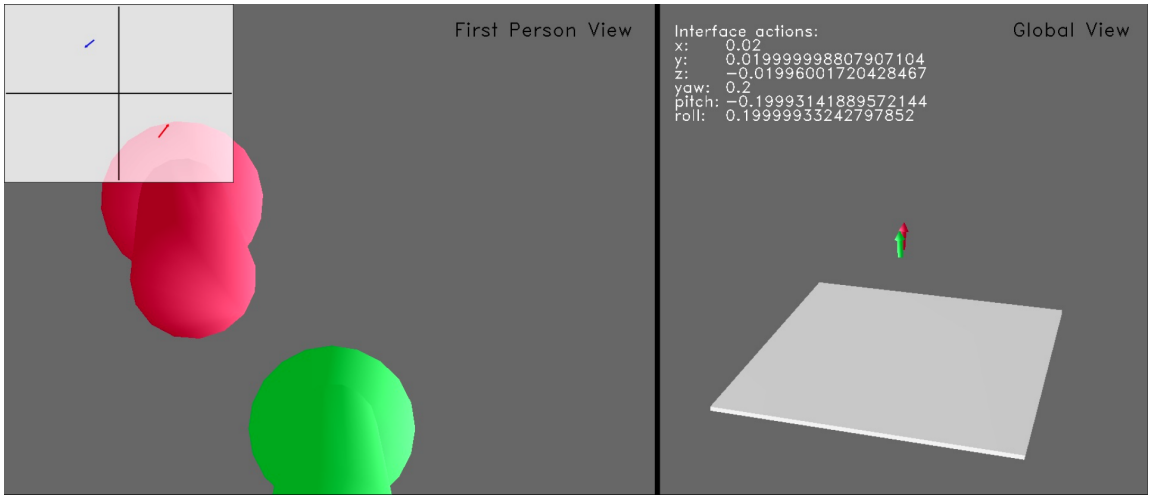

Figure 6.3 – Representation of our virtual environment for 3D navigation. Two views of the same scene are displayed, a first-person view on the left and a fixed view on the right. The user must move in the 3D space to attain the target position illustrated by a red arrow. The green arrow is fixed to the current user view to help visualization: the task can be formulated as superimposing the green arrow with the red one. User gestures of the current timestep are displayed on the top-left while the corresponding actions on the view/green arrow are displayed top-right.

handcrafted way to simulate a human user, we use the multi-agent RL setup illustrated in Figure 6.2.

6.2.1 Environment

The environment is illustrated in Figure 6.3. The 3D affine transformation to be learned by the interface agent can be expressed using homogeneous coordinates in 4 dimensions as a 4×4 matrix ϕ:

$$\phi = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{6.2}$$

As we do not consider scaling (redundant with forward motion), the matrix is totally parametrized by 6 coefficients: $[\tau_x, \tau_y, \tau_z, \rho_x, \rho_y, \rho_z]$, where $\tau_.$ and $\rho_.$ are, respectively, translation coefficients and Euler angles on the 3 axis. We limit the Euler angles to $]-\pi, \pi]$ to ensure their unicity given an axis. Such a vector can define transformations as well as object positions when using a fixed referential in the environment. With this formalization, The user agent A_u gets as observations

the camera position vector and learns a policy over actions which are finger trajectory vectors $[l_x^1\ l_y^1\ l_x'^1\ l_y'^1\ l_x^2\ l_y^2\ l_x'^2\ l_y'^2]$. The interface agent A_i observes the output of U and learns a policy over residual transformation vectors of the camera. In other words, if s^t is the current $s^{t+1} = s^t + a_i$. We define the reward function similarly to the reward used in the 2D problem described in Chapter 4:

$$r = -\sum_i \left(||\boldsymbol{o}_i - \boldsymbol{t}_i||_1 + ||\boldsymbol{o}_i - \boldsymbol{t}_i||_2 \right) - 0.2, \tag{6.3}$$

where $\{\boldsymbol{o}_i\}$ and $\{\boldsymbol{t}_i\}$ are respectively the vertices of the green and the red arrow (see Figure 6.3). The difference with the 2D reward function is that each object has 2 vertices instead of 3, and that vertices are in the 3D space.

6.2.2 Experimental Details

We chose the model-free off-policy actor-critic method Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2015) described in Section 2.3.4. In our setup, an epoch cycle consists of two phases: (i) a rollout phase on an episode, where all quadruplets [state, action, reward, new state] are stored in the replay memory of the agents; (ii) a training phase where quadruplets are randomly sampled from the memory, batched (in sizes of 4096) and used to train the actor and the critic of the agents. We define an epoch as 100 epoch cycles. The training is arbitrarily stopped when no improvement on the metrics is observed. A training session takes about 2 days on a Titan-X Pascal GPU.

Joint training of both, A_u and A_i, was not successful. We suspect the added variance and the moving value of state-action pairs for both agents as a source of the problem. Similar to Boutilier 1996, we chose to train them in an alternating manner: during an epoch, only one agent will be trained, while the weights of the other agents are kept fixed.

We consider two ways for the two agents to communicate. The simplest one is a two-instant communication: the decoder produces a gesture of only two time instants, and the interface produces the corresponding action. It is simple, but leads to non-smooth user gestures difficult to appreciate from a human viewpoint. We also consider stacking time instants: A_u produces a complete gesture of 10 timesteps, and A_i must produce the corresponding sequence of actions (9 if there are 10 timesteps). In this case, a step from the RL perspective will contain 10 update steps of the environment. This decouples the update speed of the agents from the sampling speed of the finger gestures, referred to as "*Stacking*" in Table 6.1.

Architectures – In what follows, FCX refers to an FC layer with X hidden units, with layer normalization and ReLU activation. The actor of A_u is an MLP with two hidden FC100 layers. The output layer is FC and activated with tanh, predicting a vector of size 8 (the latent code z expanded by the VAE decoder D). Another

	Mean reward/ep.	Mean #steps/ep.	Nb. training steps
No Stacking	3.6±1.0	53±1	17.6M±0.4M
Stacking	0.5±1.5	56±4	24.6M±6.3M
Theoretical Opt.	5.0	40	N/A

Table 6.1 – Results on the 3D environment. The last line gives theoretical optimal results based on the interface action amplitude, without considering a naturalness constraint. The Mean reward/ep. is the mean cumulated reward per episode obtained in the best performing epoch. The Mean nb. steps/ep. is the mean number of timesteps needed to successfully finish an episode in the best performing epoch. The Nb. training steps is the number of environment steps that was needed to attain the best performing epoch. Stacking improves usability but *NOT* efficiency.

FC100 layer is plugged to the first hidden layer, with an output layer producing the estimate $\hat{a_i}$. The critic of A_u is an MLP with two hidden FC100 layers. The policy action is concatenated to the first hidden layer. A linear FC layer predicts the value Q.

The actor of A_i is an MLP with two hidden FC64, and an output layer with tanh activation. The output size is either 6 for the standard solution or 6x9=54 for the stacked solution. The critic of A_i has the same architecture as the critic of A_u, except hidden layers are FC64.

6.2.3 Results

Quantitative results are given in Table 6.1. Each setup was reproduced with 3 different random seeds. We consider that a run has converged whenever all 100 episodes of an epoch are successful. Once it has converged, it can still improve by solving episodes faster. This is measured as the mean number of steps needed to solve an episode. The mean reward per episode is also correlated to the quality of the interaction protocol, but should be interpreted differently. Indeed, a run with a lower mean step per episode but a higher mean reward per episode is most likely less satisfactory than a run with higher steps but lower reward. This is because the first type of solutions tend to be less continuous with harsher action changes, while the second is technically slower but goes in the direction of the objective more smoothly.

"Stacking" and usability — the interaction protocols must also be observed visually in order to assess their global quality: good interfaces should display distinctive characteristics, such as a similar curvature between 2D trajectories and 3D movements of the camera, or well defined classes for similar actions.

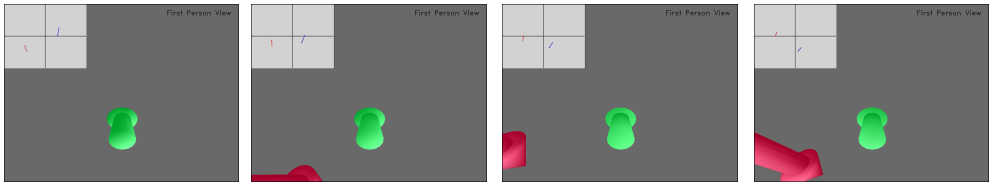

Figure 6.4 – An example of 4 back to back frames from the MARL setup learning the 3D navigation problem with instant stacking. We want to emphasize on the continuous aspect of user gestures (top-left) and the semantic. We can see that the user agent is currently performing a rotation-like gesture.

While stacking does *NOT* improve efficiency (as shown in Table 6.1), it makes the interaction protocol more usable. The continuous aspect of gestures using instant stacking is illustrated in Figure 6.4. The results of this experiments are published in Debard et al. 2019.

We cannot fully assess the quality of this learned interaction protocol without performing a user satisfaction study: while the efficiency and effectiveness of the interaction protocol are good, we have no measurment of the user satisfaction. Our setup however ensures that some properties of satisfying interfaces are provided: because we only learn the interaction protocol, we need only consider the gesture-related satisfaction, i.e. intuitivity and naturalness of gestures. Our setup favorizes naturalness but cannot infer intuitivity, and this is the point that requires human testing. Falling back to our step-by-step method for learning adaptive interaces described in Section 6.1.2, this human testing corresponds to the third step. After the pre-training of the interface, there are two possibilities:

- the interaction protocol is just not intuitive enough to be used correctly: it is discarded and another interaction protocol with a different initialization is tested.

- The interaction protocol is usable: it is further refined through human use to ensure maximum usability before being deployed to end users.

This process will complete the training process of adaptive interaction protocols while reducing the need for human supervision to later steps of interaction design.

6.3 Conclusions

We have described in this section an automatic learning process for complex interaction protocols, allowing for the automation of effectiveness and efficiency

maximization in interaction protocol design. This design method is meant to achieve three goals:

- reduce the workload of User Interface (UI) designers by only requiring human intervention during the intuitivity evaluation process,
- produce adaptive interaction protocols to better fit user expectations,
- and potentially discover new interaction protocols in complex environments where the initial design of an interface is a difficult task.

We believe that the progressive replacement of handcrafted algorithms by Machine Learning (ML) algorithms in UIs will gradually make these interfaces more flexible and allow users to better manipulate them to their needs. Our model is a proof of concept that can be enhanced and derived in many ways: define more specific constraints to produce better solutions, combine the learning of different parts of the UI, add an internal memory to the RL agent... The modularity of our setup can allow for the tackling of virtually any interaction protocol design problem given a proper training environment and correlated interaction logs.

CHAPTER 7

CONCLUSION: ACHIEVEMENTS, LIMITATIONS AND PERSPECTIVES

Contents

7.1 Contributions 99
7.1.1 Supervised Learning for Touch Gesture Recognition . . . 100
7.1.2 Interaction Protocol Design 100
7.2 Perspectives and Future Work 102
7.2.1 Real-Time Gesture Recognizer 102
7.2.2 More Human-Like User Models 102
7.2.3 Memory-Based Interface Agents 103
7.2.4 Adaptation During Real Use 103
7.2.5 A Final Point 103

This chapter closes three years of work between Machine Learning (ML) and Human-Computer Interaction (HCI). We can now summarize the contributions of this thesis and discuss their limitations and perspectives.

7.1 Contributions

User Interface (UI) design is a systematic challenge for new technologies and applications. As these applications are getting more complex, so does the UI design process. At some point, analyzing and processing user data becomes complicated for human specialists and ML starts to shine through automated feature learning and decision making. However, good ML solutions require data availability, a careful problem definition and a fitting architecture for the models to correctly generalize to real-life applications: these were the points adressed throughout this thesis to improve both the usability of ML in HCI and the quality of the ML solutions.

7.1.1 Supervised Learning for Touch Gesture Recognition

Gesture recognition is the first step toward user intent understanding: before interpreting a gesture, we must correctly identify it. We defined a proper pipeline to train such recognizers to be as generalizable as possible device- and user-wise.

- Because of the absence of public interaction touch gesture datasets, the first step was to create a challenging dataset containing as much natural variance as possible. We collected Itekube-7, composed of 6591 gestures distributed in 7 different classes and performed by 27 different users. Its goal was to propose a challenging classification task with generalization in mind. The test set for the classification task was made of gestures performed by users absent from the training set, requiring a solution to perform well on unseen users. Users were encouraged to show as much variety as possible while getting a minimal protocol. We believe this dataset is the most challenging touch gesture dataset publicly available as of now.

- We proposed a new feature preserving sampling method for touch gestures, allowing for reduced fixed-length representation without interpolation. This dynamic sampling method allowed us to reduce the number of datapoints by about a factor 10 while still preserving good performance.

- We then compared baseline Deep Learning (DL) architectures on the classification task using state-of-the-art models for sequential data. We observed through these experiments the superiority of convolutional features for our problem.

- Because we wanted our solution to be usable in a maximum number of contexts, the model needed not only be performing but also portable. This motivated us to condense the capabilities of recurrent and convolutional features into one cell: the Conv-MDGRU. We combined the local observation at a time of Convolutional Neural Networks (CNNs) with the multi-dimensional recurrence of Multi-Dimensional Long Short-Term Memorys (MDLSTMs) while further reducing the number of weights by generalizing Gated Recurrent Units (GRUs) to multi-dimensional recurrence. This architecture performed better than our baseline CNN while having 10 times less weights.

7.1.2 Interaction Protocol Design

The following work focused on the interpretation of user gestures to actions in a virtual environment, called the design of interaction protocols. This task is inherently hard because of a number of reasons:

- the protocol must allow for the performance of desired actions while minimizing the time required for the user to learn it,

- different users have different expectations and prior knowledge, making a unique interaction protocol hard to satisfy every user,

- user satisfaction is not directly quantifiable, making the conception and the evaluation of an interaction protocol a tedious process requiring user feedback.

We proposed a novel approach to mitigate these problems, using Reinforcement Learning (RL) to learn interaction protocols from synthesized training data. This part can be seen as our most innovative work, using ML to help overcome a recurring challenge in HCI.

- We cast the interaction protocol as a RL agent observing user gestures and taking actions in a virtual environment. The agent had to solve a Partially Observable Markov Decision Process (POMDP) problem, interpreting user intent through observed gestures. A simplified experiment was used to test this concept: the agent needed to re-learn an already known interaction protocol, namely Pinch-to-Zoom. This experience was done in a 2D environment requiring a user to superimpose an object over a target location. The object could be translated, rotated and rescaled. User data was synthesized in a deterministic fashion using our knowledge of the solution. The interface agent could only observe user gestures (without information about the position of the object and target). We showed that in these conditions, a RL agent is able to re-learn the optimal interaction protocol for this task.

- In order to synthesize user data for unknown interaction protocols, we needed to learn a user model. This meant learning knowledge representations usable by a model to decide which gesture to perform. Thus, we trained a Variational Auto-Encoder (VAE) on Itekube-7 to reconstruct gestures, illustrating the qualities of its latent space for our task. We compared different encoder and decoder architectures and proposed a novel sequential-VAE: instead of decoding each latent variable by unrolling the code with a Recurrent Neural Networks (RNN), the whole code is used for generation at each timestep. Firstly, this forces the code to condense both spatial and temporal properties of the gestures, secondly it allows us to control how long the decoder should unroll the code. Once trained, the decoder can then generate gestures of arbitrary length. Our user model was then a RL agent observing the environment and producing a code interpreted by the trained VAE decoder to produce gestures. This constrained our user model to produce gestures from the learned human gesture properties.

- As a final contribution, we made use of this user model to train an interface agent on a 3D navigation environment. This environment was a generalization of our earlier 2D environment, where the user had to move a camera in a 3D space to reach a target position; in this case, the optimal solution is unknown. We proposed a Multi-Agent Reinforcement Learning (MARL) setup to train the

user and the interface agent all together where the agents learned to cooperate in order to solve the task. We showed that the obtained solutions were getting satisfying performance results while the user agent was constrained to the learned human-like gestures.

7.2 Perspectives and Future Work

This thesis proposed new directions to design better and co-adaptive UIs. Every contribution can be further developed and improved to generalize to more complex contexts and tackle more real-life applications. We discuss in the following section the improvements we did not have time to develop during this thesis and their potential interest.

7.2.1 Real-Time Gesture Recognizer

During the first part of our work on gesture recognition, we limited ourselves to discrete gesture recognition. In real case scenarios, we want our model to handle real-time classification and early detection of the gestures. This could be done using:

- a variant of our dynamic sampling saving transitions and uniformly dropping the datapoint excess,
- and a hierarchical model combining short-sighted features on a low level with a higher level recurrent model aggregating these features temporally.

This approach would also allow for the regression of potential gesture parameters such as rotation angle or scaling coefficient. If we want this model to be used in a multi-user context, the model must be able to perform some clustering to group fingers by gesture. The higher level of our model should play a role in this decision process as temporal information can help sort out ambiguous situations (close fingers used for different gestures).

7.2.2 More Human-Like User Models

As of now, our user model is only constrained to produce its gestures from the trained latent space of our VAE. While this constrain is enough to display some human-like gestures, it does not take into account other important natural constraints. For example, a human user will tend to minimize the effort required to perform a gesture. This could be translated as minimizing the number of different gestures as well as their complexity. The user model also needs to model long-term dependencies if we are to work in environments requiring sequences of

gestures to perform an action (click on a menu then select an action). This lead will however be limited by the current state-of-the-art in RL: continuous control in partially observable environments are hard-to-solve problems, and the existing algorithms to learn such policies have trouble converging.

7.2.3 Memory-Based Interface Agents

The automatic design of co-adaptive interfaces still is a long way from being solved. Building upon our work, we can enhance our current interface modelization to better fit human expectations. The logical step to take as with the user agent is to model long-term dependencies. The interface agent should use this information to not only interpret sequences of gestures, but also propose more human-like actions. There are two temporal properties we can think of right now:

- continuity, meaning that a continuous gesture should be interpreted as a continuous sequence of actions,
- and correlation between the curvature of user gestures and the corresponding actions. It means that variations in the local curvature of a gesture should be reflected in the interpretation of the interface.

These conditions are important for the interface to be intuitive.

7.2.4 Adaptation During Real Use

Last but not least, the mendatory step to deploy these solutions in real conditions is to find out how to integrate user feedback during real use to tune the interface. Some signals can be used such as the repetition or the cancelling of a gesture (most likely indicating a bad interpretation of the agent), but we can also imagine direct feedback in the form of binary rewards: the possibility for the user to hit a “happy face” or a “sad face”.

7.2.5 A Final Point

There is a lot of exciting work to do to propose more flexible and co-adaptive interfaces. The most direct interest from this work may be the learning of more robust gesture recognizers in difficult contexts. With a reasonable amount of work, we believe there would be added value to re-learn current interaction protocols and replace them with their RL counterpart. By doing so, each interface could propose co-adaptation and fit individually to users. Eventually, with more robust RL models and with better constraints, we can expect the automatic design process

to help specialists design new interfaces for complex environments: the MARL setup could propose prototypes for the specialist to evaluate.

Completly modelling human users is nonsensical with the current capabilities of ML, thus the complete automation of UI design is impossible. We can however already automate part of this process, and in consequence optimize these parts with respect to collected data. This thesis aimed to take it a step further to gradually automate more of this design process. Our possible uses of electronic devices will grow with the quality of the interfaces: the more we blur the lines between the man and the machine, the more we can expect from it; and Artificial Intelligence is very likely to play an important role in this.

BIBLIOGRAPHY

Albawi, Saad and Osman Ucan (2018). "Social Touch Gesture Recognition Using Convolutional Neural Network". In: *Computational Intelligence and Neuroscience* (cit. on p. 22).

Alvarez-Cortes, Victor, Benjamín Zayas, Victor Silva, and Jorge Ramirez Uresti (2007). "Current Trends in Adaptive User Interfaces: Challenges and Applications". In: *Electronics, Robotics and Automotive Mechanics Conference* (cit. on p. 33).

Anthony, Lisa and Jacob Wobbrock (2012). "$N-protractor: a fast and accurate multistroke recognizer". In: *Proceedings of Graphics Interface 2012* (cit. on pp. 50, 56, 57).

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). "Layer Normalization". In: *arXiv* (cit. on p. 71).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *International Conference on Learning Representations* (cit. on p. 22).

Bailly, Gilles, Antti Oulasvirta, Timo Kötzing, and Sabrina Hoppe (2013). "MenuOptimizer: Interactive Optimization of Menu Systems". In: *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology* (cit. on p. 33).

Belongie, S., J. Malik, and J. Puzicha (2002). "Shape Matching and Object Recognition Using Shape Contexts". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (cit. on p. 46).

Beltramelli, Tony (2018). "Pix2code: Generating Code from a Graphical User Interface Screenshot". In: *ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (cit. on p. 35).

Benalcázar, M. E., A. G. Jaramillo, Jonathan, A. Zea, A. Páez, and V. H. Andaluz (2017). "Hand gesture recognition using machine learning and the Myo armband". In: *2017 25th European Signal Processing Conference (EUSIPCO)* (cit. on p. 4).

Bengio, Y., P. Simard, and P. Frasconi (1994). "Learning Long-term Dependencies with Gradient Descent is Difficult". In: *IEEE Transactions on Neural Networks and Learning Systems* (cit. on pp. 24, 43).

Boutilier, Craig (1996). "Planning, Learning and Coordination in Multiagent Decision Processes". In: *Theoretical Aspects of Rationality and Knowledge* (cit. on p. 95).

Bowman, Doug A., Ernst Kruijff, Joseph J. LaViola, and Ivan Poupyrev (2001). "An Introduction to 3-D User Interface Design". In: *Presence: Teleoperators and Virtual Environments* (cit. on p. 33).

Bowman, Doug, Jian Chen, Chadwick Wingrave, John Lucas, Andrew Ray, Nicholas Polys, Qing Li, Yonca Haciahmetoglu, Ji-Sun Kim, Seonho Kim, Robert Boehringer, and Tao Ni (2006). "New Directions in 3D User Interfaces". In: *The International Journal of Virtual Reality* (cit. on p. 16).

Breuel, Thomas M. (2017). "Robust, Simple Page Segmentation Using Hybrid Convolutional MDLSTM Networks". In: *International Conference on Document Analysis and Recognition* (cit. on p. 49).

Brin, Sergey and Lawrence Page (1998). "The Anatomy of a Large-scale Hypertextual Web Search Engine". In: *Computer Networks and ISDN Systems* (cit. on p. 33).

Brouet, Rémi (2014). "Multi-touch gesture interactions and deformable geometry for 3D edition on touch screen". PhD thesis. Université de Grenoble (cit. on p. 67).

Cashion, Jeffrey, Chadwick Wingrave, and LaViola Joseph J. Jr (2012). "Dense and Dynamic 3D Selection for Game-Based Virtual Environments". In: *IEEE transactions on visualization and computer graphics* (cit. on p. 16).

Cha, Jaekwang, JinHyuk Kim, and Shiho Kim (2019). "Hands-Free User Interface for AR/VR Devices Exploiting Wearer's Facial Gestures Using Unsupervised Deep Learning". In: *Sensors* (cit. on p. 35).

Chen, Xiuli, Gilles Bailly, Duncan P. Brumby, Antti Oulasvirta, and Andrew Howes (2015). "The Emergence of Interactive Behavior: A Model of Rational Menu Search". In: *ACM Conference on Human Factors in Computing Systems (CHI)* (cit. on pp. 33, 34).

Chen, Zhaoxin, Eric Anquetil, Harold Mouchere, and Christian Viard-Gaudin (2014). "A Graph Modeling Strategy for Multi-touch Gesture Recognition". In: *The International Conference on Frontiers in Handwriting Recognition (ICFHR)* (cit. on pp. 15, 33, 44).

Cho, Kyunghyun, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *The 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (cit. on pp. 24, 43).

Cirelli, Mauricio and Ricardo Nakamura (2014). "A Survey on Multi-touch Gesture Recognition and Multi-touch Frameworks". In: *The Ninth ACM International Conference on Interactive Tabletops and Surfaces* (cit. on p. 15).

Cooijmans, Tim, Nicolas Ballas, César Laurent, Caglar Gulcehre, and Aaron Courville (2016). "Recurrent Batch Normalization". In: *The International Conference on Learning Representations* (cit. on p. 53).

Cortes, Corinna and Vladimir Vapnik (1995). "Support-vector networks". In: *Machine Learning* (cit. on p. 13).

Debard, Quentin, Jilles Steeve Dibangoye, Stéphane Canu, and Christian Wolf (2019). "Learning 3D Navigation Protocols on Touch Interfaces with Cooper-

ative Multi-Agent Reinforcement Learning". In: *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD)* (cit. on pp. 9, 60, 75, 87, 97).

Debard, Quentin, Christian Wolf, Stéphane Canu, and Julien Arné (2018). "Learning to Recognize Touch Gestures: Recurrent vs. Convolutional Features and Dynamic Sampling". In: *The International Conference on Automatic Face and Gesture Recognition (FG)* (cit. on pp. 9, 38, 51).

Degris, Thomas, Martha White, and Richard S. Sutton (2012). "Linear Off-Policy Actor-Critic". In: *International Conference on Machine Learning* (cit. on p. 30).

Dehghani, Mostafa, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft (2017). "Neural Ranking Models with Weak Supervision". In: *The 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (cit. on p. 3).

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL-HLT* (cit. on p. 26).

Doll, William J. and Gholamreza Torkzadeh (1988). "The Measurement of End-User Computing Satisfaction". In: *MIS Quarterly* (cit. on p. 65).

Doll, William, Xiaodong Deng, T.S. Ragu-Nathan, Gholamreza Torkzadeh, and Weidong Xia (2004). "The Meaning and Measurement of User Satisfaction: A Multigroup Invariance Analysis of the End-User Computing Satisfaction Instrument". In: *Journal of Management Information Systems* (cit. on p. 66).

Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* (cit. on p. 19).

Ferretti, S., S. Mirri, C. Prandi, and P. Salomoni (2014). "Exploiting Reinforcement Learning to Profile Users and Personalize Web Pages". In: *International Computer Software and Applications Conference Workshops* (cit. on p. 33).

Al-Fraihat, Dimah, Mike Joy, Ra'Ed Masa'deh, and Jane Sinclair (2019). "Evaluating E-learning Systems Success: An Empirical Study". In: *Computers in Human Behavior* (cit. on p. 66).

Fukushima, Kunihiko (1980). "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* (cit. on p. 20).

Gehring, Jonas, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin (2017). "Convolutional Sequence to Sequence Learning". In: *The International Conference on Machine Learning* (cit. on p. 22).

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). "Generative Adversarial Nets". In: *Neural Information Processing Systems* (cit. on pp. 34, 77).

Graves, Alex, S Fernández, and J Schmidhuber (2007). "Multi-dimensional recurrent neural networks". In: *Artificial Neural Networks–ICANN* (cit. on pp. 25, 53).

Grudin, Jonathan (2016). *From Tool to Partner: The Evolution of Human-Computer Interaction* (cit. on p. 12).

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *International Conference on Machine Learning* (cit. on p. 31).

Halim, A., Christel Dartigues-Pallez, Frederic Precioso, Michel Riveill, A. Benslimane, and Salma Ghoneim (2016). "Human action recognition based on 3D skeleton part-based pose estimation and temporal multi-resolution analysis". In: *International Conference on Image Processing* (cit. on p. 4).

Higgins, Irina, Arka Pal, Andrei A. Rusu, Loïc Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner (2017). "DARLA: Improving Zero-Shot Transfer in RL". In: *The International Conference on Machine Learning* (cit. on p. 35).

Hinckley, Ken and Daniel Wigdor (2012). *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications* (cit. on p. 15).

Hinton, Geoffrey (2012). *Neural Networks for Machine Learning - Lecture 6e*. URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (cit. on p. 19).

Hoai, Minh and Fernando De la Torre (2014). "Max-Margin Early Event Detectors". In: *The International Journal of Computer Vision* (cit. on p. 58).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* (cit. on pp. 24, 43, 53).

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *The International Conference on Machine Learning* (cit. on pp. 53, 81).

Jiang, Nan, Alex Kulesza, Satinder Singh, and Richard Lewis (2016). "The Dependence of Effective Planning Horizon on Model Accuracy". In: *International Joint Conference on Artificial Intelligence (IJCAI)* (cit. on p. 27).

Johnson, E. A. (1965). "Touch display–a novel input/output device for computers". In: *Electronics Letters* (cit. on p. 12).

Kalchbrenner, N., I. Danihelka, and A. Graves (2016). "Grid Long Short-Term Memory". In: *The International Conference on Learning Representations* (cit. on p. 25).

Karpathy, A., G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei (2014). "Large-Scale Video Classification with Convolutional Neural Networks". In: *Conference on Computer Vision and Pattern Recognition* (cit. on p. 22).

Kin, Kenrick, Björn Hartmann, Tony DeRose, and Maneesh Agrawala (2012). "Proton++: A Customizable Declarative Multitouch Framework". In: *User Interface Software and Technology Symposium* (cit. on p. 13).

Kingma, Diederik and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (cit. on p. 19).

Kingma, Diederik and Max Welling (2013). "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations* (cit. on pp. 34, 77, 78).

Krafka, Kyle, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra M. Bhandarkar, Wojciech Matusik, and Antonio Torralba (2016). "Eye Tracking for Everyone". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 4).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25* (cit. on pp. 3, 53).

Lacoche, Jérémy, Thierry Duval, Bruno Arnaldi, Eric Maisel, and Jérôme Royan (2019). "Machine Learning Based Interaction Technique Selection for 3D User Interfaces". In: *Virtual Reality and Augmented Reality* (cit. on p. 33).

Langley, Pat (1997). "Machine learning for adaptive user interfaces". In: *KI-97: Advances in Artificial Intelligence* (cit. on pp. 32, 33).

Lecun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*, pp. 2278–2324 (cit. on p. 20).

Li, Pan, Zhen Qin, Xuanhui Wang, and Don Metzler (2019). "Combining Decision Trees and Neural Networks for Learning-to-Rank in Personal Search". In: *Conference on Knowledge Discovery and Data Mining (SIGKDD)* (cit. on p. 3).

Lillicrap, Timothy, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2015). "Continuous control with deep reinforcement learning". In: *CoRR* (cit. on pp. 31, 71, 72, 95).

Liu, Jun, Amir Shahroudy, Dong Xu, and Gang Wang (2016). "Spatio-Temporal LSTM with Trust Gates for 3D Human Action Recognition". In: *The European Conference on Computer Vision (ECCV)* (cit. on pp. 26, 53).

Lü, Hao and Y Li (2012). "Gesture Coder: A tool for programming multi-touch gestures by demonstration". In: *SIGCHI Conference on Human Factors in Computing Systems* (cit. on pp. 4, 14, 33, 44).

Machmud, Rizan (2018). "Study of Satisfaction of Information System Users in Study Program (SIMPRODI) in Gorontalo State University". In: *International Journal of Applied Business and International Management* (cit. on p. 66).

MacKay, David JC (2001). "Local Minima, Symmetry-breaking, and Model Pruning in Variational Free Energy Minimization". In: (cit. on p. 80).

Malik, Shahzad, Abhishek Ranjan, and Ravin Balakrishnan (2005). "Interacting with Large Displays from a Distance with Vision-tracked Multi-finger Gestural

Input". In: *The 18th Annual ACM Symposium on User Interface Software and Technology* (cit. on p. 13).

Martín, Abadi, Agarwal Ashish, Barham Paul, Brevdo Eugene, Chen Zhifeng, Citro Craig, Corrado Greg S., Davis Andy, Dean Jeffrey, Devin Matthieu, Ghemawat Sanjay, Goodfellow Ian, Harp Andrew, Irving Geoffrey, Isard Michael, Yangqing Jia, Jozefowicz Rafal, Kaiser Lukasz, Kudlur Manjunath, Levenberg Josh, Mané Dan, Monga Rajat, Moore Sherry, Murray Derek, Olah Chris, Schuster Mike, Shlens Jonathon, Steiner Benoit, Sutskever Ilya, Talwar Kunal, Tucker Paul, Vanhoucke Vincent, Vasudevan Vijay, Viégas Fernanda, Vinyals Oriol, Warden Pete, Wattenberg Martin, Wicke Martin, Yu Yuan, and Zheng Xiaoqiang (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. URL: http://tensorflow.org/ (cit. on p. 52).

Matthey, Loïc, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner (2017). "β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning representations* (cit. on p. 78).

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *International Conference on Machine Learning* (cit. on p. 31).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller (2013). "Playing Atari with Deep Reinforcement Learning". In: *ArXiv* (cit. on p. 31).

Moran, K. P., C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk (2018). "Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps". In: *IEEE Transactions on Software Engineering* (cit. on p. 3).

Moysset, Bastien, Pierre Adam, Christian Wolf, and Jérôme Louradour (2015). "Space Displacement Localization Neural Networks to locate origin points of handwritten text lines in historical documents". In: *ICDAR 2015 Workshop on Historical Document Imaging and Processing* (cit. on p. 49).

Moysset, Bastien, Christopher Kermorvant, and Christian Wolf (2018). "Learning to detect, localize and recognize many text objects in document images from few examples". In: *International Journal on Document Analysis and Recognition (IJDAR)* (cit. on p. 49).

Nair, Ashvin, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine (2018). "Visual Reinforcement Learning with Imagined Goals". In: *Neural Information Processing Systems* (cit. on p. 35).

Nair, Vinod and Geoffrey E. Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *International Conference on Machine Learning* (cit. on p. 17).

Naz, Saeeda, Arif Umar, Riaz Ahmad, Imran Siddiqi, Saad Ahmed, Muhammad Razzak, and Faisal Shafait (2017). "Urdu Nastaliq Recognition using Convolutional-Recursive Deep Learning". In: *Neurocomputing* (cit. on p. 49).

Nguyen, T., P. Vu, H. Pham, and T. Nguyen (2018). "Deep Learning UI Design Patterns of Mobile Apps". In: *International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)* (cit. on p. 35).

Oord, Aäron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu (2016). "WaveNet: A Generative Model for Raw Audio". In: *The ISCA Speech Synthesis Workshop* (cit. on p. 4).

Ortega, Francisco R., Fatemeh Abyarjoo, Armando Barreto, Naphtali Rishe, and Malek Adjouadi (2016). *Interaction Design for 3D User Interfaces: The World of Modern Input Devices for Research, Applications, and Game Development*. A. K. Peters, Ltd. (cit. on pp. 15, 33).

Park, J. W., Ju-Hwan Seo, Young-Hoon Nho, and Dong-Soo Kwon (2019). "Touch Gesture Recognition System based on 1D Convolutional Neural Network with Two Touch Sensor Orientation Settings". In: *2019 16th International Conference on Ubiquitous Robots (UR)* (cit. on p. 22).

Pazzani, Michael, Jack Muramatsu, and Daniel Billsus (1996). "Syskill & Webert: Identifying Interesting Web Sites". In: *The 13th National Conference on Artificial Intelligence - Volume 1* (cit. on p. 32).

Perlich, Claudia, Brian Dalessandro, O. Stitelman, T. Raeder, and F. Provost (2013). "Machine Learning for Targeted Display Advertising: Transfer Learning in Action". In: *Machine Learning* (cit. on p. 3).

Radford, Alec (2018). "Improving Language Understanding by Generative Pre-Training". In: *OpenAI* (cit. on p. 26).

Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar (2018). "On the Convergence of Adam and Beyond". In: *The International Conference on Learning Representations* (cit. on p. 19).

Ropinski, Timo, Frank Steinicke, and Klaus Hinrichs (2005). "A Constrained Road-based VR Navigation Technique for Travelling in 3D City Models". In: *The 2005 International Conference on Augmented Tele-existence* (cit. on p. 16).

Rubine, Dean (1991). "Specifying Gestures by Example". In: *The 18th Annual Conference on Computer Graphics and Interactive Techniques* (cit. on p. 14).

Sanabria, Melissa, Sherly, Frédéric Precioso, and Thomas Menguy (2019). "A Deep Architecture for Multimodal Summarization of Soccer Games". In: *International Workshop on Multimedia Content Analysis in Sports* (cit. on p. 4).

Scholliers, Christophe, Lode Hoste, Beat Signer, and Wolfgang De Meuter (2011). "Midas: A Declarative Multi-touch Interaction Framework". In: *The Fifth International Conference on Tangible, Embedded, and Embodied Interaction* (cit. on p. 13).

Semeniuta, Stanislau, Aliaksei Severyn, and Erhardt Barth (2016). "Recurrent Dropout without Memory Loss". In: *CoRR* (cit. on p. 53).

Seo, Young-Woo and Byoung-Tak Zhang (2000). "A Reinforcement Learning Agent for Personalized Information Filtering". In: *International Conference on Intelligent User Interfaces* (cit. on p. 33).

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller (2014). "Deterministic Policy Gradient Algorithms". In: The International Conference on Machine Learning (cit. on p. 31).

Simonyan, Karen and Andrew Zisserman (2014). "Two-Stream Convolutional Networks for Action Recognition in Videos". In: *Advances in Neural Information Processing Systems* (cit. on p. 22).

Sønderby, Casper Kaae, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther (2016). "Ladder Variational Autoencoders". In: *International Conference on Neural Information Processing Systems* (cit. on p. 80).

Song, Jiule, Giovanni Migliaccio, Guangbin Wang, and Hao Lu (2017). "Exploring the Influence of System Quality, Information Quality, and External Service on BIM User Satisfaction". In: *Journal of Management in Engineering* (cit. on p. 66).

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *The Journal of Machine Learning Research* (cit. on p. 53).

Sun, Huanbo and Georg Martius (2019). "Machine Learning for Haptics: Inferring Multi-Contact Stimulation From Sparse Sensor Configuration". In: *Frontiers in Neurorobotics* (cit. on p. 4).

Sutherland, Ivan E. (1964). "Sketch Pad a Man-machine Graphical Communication System". In: *The SHARE Design Automation Workshop* (cit. on p. 12).

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press (cit. on p. 28).

Szalavári, Zsolt and Michael Gervautz (1997). "The Personal Interaction Panel – a Two-Handed Interface for Augmented Reality". In: *Computer Graphics Forum* (cit. on p. 16).

Tan, Desney, George Robertson, and Mary Czerwinski (2001). "Exploring 3D Navigation: Combining Speed-coupled Flying with Orbiting". In: *CHI 2001, Human Factors in Computing Systems* (cit. on p. 16).

Taylor, Graham W., Rob Fergus, Yann Lecun, and Christoph Bregler (2010). "Convolutional Learning of Spatio-temporal Features". In: *The European Conference on Computer Vision* (cit. on p. 22).

Van Den Oord, Aäron, Nal Kalchbrenner, and Koray Kavukcuoglu (2016). "Pixel Recurrent Neural Networks". In: *International Conference on Machine Learning* (cit. on p. 48).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). "Attention is All You

Need". In: *International Conference on Neural Information Processing Systems* (cit. on p. 26).

Vatavu, Radu-Daniel, Lisa Anthony, and Jacob O. Wobbrock (2018). "$Q: a super-quick, articulation-invariant stroke-gesture recognizer for low-resource devices". In: *MobileHCI* (cit. on p. 14).

Watkins, Christopher J. C. H. and Peter Dayan (1992). "Q-learning". In: *Machine Learning* (cit. on p. 30).

Weir, Daryl, Simon Rogers, Roderick Murray-Smith, and Markus Löchtefeld (2012). "A User-specific Machine Learning Approach for Improving Touch Accuracy on Mobile Devices". In: *ACM Symposium on User Interface Software and Technology* (cit. on p. 33).

Weld, Daniel S., Corin Anderson, Pedro Domingos, Oren Etzioni, Krzysztof Gajos, Tessa Lau, and Steve Wolfman (2003). "Automatically Personalizing User Interfaces". In: *International Joint Conferences on Artificial Intelligence* (cit. on p. 33).

Williams, Ronald J. (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* (cit. on p. 29).

Wobbrock, Jacob O, Andrew D Wilson, and Yang Li (2007). "Gestures without libraries, toolkits or training: a 1 recognizer for user interface prototypes". In: *The 20th annual ACM symposium on User interface software and technology UIST 07* (cit. on p. 44).

Wu, Mike and Ravin Balakrishnan (2003). "Multi-finger and Whole Hand Gestural Interaction Techniques for Multi-user Tabletop Displays". In: *The 16th Annual ACM Symposium on User Interface Software and Technology* (cit. on p. 13).

Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc Le, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, ukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, and Jeffrey Dean (2016). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *arXiv* (cit. on p. 22).

Wu, Yuhuai, Elman Mansimov, Shun Liao, Roger B. Grosse, and Jimmy Ba (2017). "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". In: *Neural Information Processing System* (cit. on p. 31).

Yu, Shun-Zheng (2010). "Hidden Semi-Markov Models". In: *Artificial Intelligence* (cit. on p. 45).

Zhu, Yi, Zhenzhong Lan, Shawn Newsam, and Alexander Hauptmann (2018). "Hidden Two-Stream Convolutional Networks for Action Recognition". In: *Asian Conference on Computer Vision* (cit. on p. 4).