SORBONNE UNIVERSITÉ/ TÉLÉCOM PARISTECH/ IRCAM
Faculty of Computer Science
SAR / ATIAM

MASTER'S THESIS

Deep Reinforcement Learning for the Design of Musical Interaction

---

Author:
Bavo VAN KERREBROECK

Supervisor:
Frédéric BEVILACQUA
Hugo SCURTO

Thesis written during an internship at

IRCAM, UMR STMS 9912
Paris 75004, France
ISMM team
(12/02/2018 - 10/08/2018)
20th August 2018

Abstract

This research is situated on the cross-roads of several broad and widely interpreted terms such as exploration, artificial intelligence, interaction and music. It investigates to what extent an intelligent system can aid a user in the exploration of a musical space. The research took place in the context of the PhD thesis of Hugo Scurto who proposes to use the technique of reinforcement learning to address this challenge. It allows to create an adaptive agent that evolves over time and does not require labels or representative examples to learn through the use of a feedback signal. In addition, learning takes into account the environment's state transitions which brings the learning closer to the dynamics inherent in the exploration of a musical space.

More specific, this research adds the use of deep reinforcement learning to the agent. It allows to generalise learning to unexplored areas and thus to extend the application of reinforcement learning to high-dimensional spaces. In this report, it is shown that simple methods such as DTAMER outperform more complicated ones on a generic problem setting. Tile coding is used as a simple and efficient new technique for density estimation to construct an exploration bonus [6] and improve the exploration strategy of the agent.

Also, this research proposes several novel user controls to interact with the agent to improve learning and the user experience. Evaluative, instructive and exploration controls are formalised extending existing interactive deep reinforcement learning methods and are shown to bring new, useful, usable and attractive affordances to the user. The variation of uses and interaction modes with the agent is demonstrated, as well as a quantitative evaluation of user feedback consistency and exploration density. While the application was positively received by users, the agent's learning during user interaction remained inconclusive. This was mainly due to new parameter settings to improve the agent's reactivity, user feedback inconsistencies and variance in user exploration strategies. Future work could entail the search for proper parameter settings based on earlier provided user feedback, testing the exploration of different musical spaces, new exploration visualisations and other user controls.

This report starts with a technical analysis of specific algorithms followed by several propositions for the human-computer interaction and an experimental evaluation involving 14 participants. Overall, it was shown that the use of interactive machine learning techniques such as interactive deep reinforcement learning provide exciting new directions for research and have an added and complementary value next to standard ways of manual exploration. Also, using an intelligent partner entails new modes of interaction and offers opportunities to shape and create an improved user experience. A final version of the application will be used for an installation at the upcoming ISMIR 2018[1] conference in Paris.

Keywords: deep reinforcement learning, music, exploration, human-computer interaction

[1] http://ismir2018.ircam.fr/

Résumé

Cette recherche se situe aux intersection de plusieurs termes largement interprétés tels que l'exploration, l'intelligence artificielle, l'interaction et la musique. Il étudie dans quelle mesure un système intelligent peut aider un utilisateur à explorer un espace musical. La recherche a eu lieu dans le cadre de la thèse de doctorat de Hugo Scurto qui propose d'utiliser la technique de l'apprentissage par renforcement pour relever ce défi. Cette technique permet de créer un agent adaptatif qui évolue avec le temps et qui ne nécessite pas d'étiquettes ou d'exemples représentatifs à apprendre en utilisant un signal de récompense. En outre, l'apprentissage prend en compte les transitions d'état de l'environnement, ce qui rapproche l'apprentissage de la dynamique inhérente à l'exploration d'un espace musical.

Plus précisément, cette recherche ajoute l'utilisation d'un apprentissage par renforcement profond à l'agent. Il permet de généraliser l'apprentissage à des domaines inexplorés et d'étendre ainsi l'application de l'apprentissage par renforcement aux espaces de grande dimension. Dans ce rapport, il est montré que des méthodes simples telles que DTAMER sont plus performantes que les méthodes plus complexes sur un problème générique. Le codage en mosaïque est utilisé comme une nouvelle technique simple et efficace pour l'estimation de la densité afin de construire un bonus d'exploration [6] et d'améliorer la stratégie d'exploration de l'agent.

En outre, cette recherche propose plusieurs nouveaux contrôles utilisateur pour interagir avec l'agent afin d'améliorer l'apprentissage et l'expérience utilisateur. Des contrôles d'évaluation, d'instruction et d'exploration sont formalisés pour étendre les méthodes interactives d'apprentissage renforcement profond existantes. Ils sont montrés pour apporter de nouvelles affordances utiles, utilisables et attrayantes à l'utilisateur. La variation des utilisations et des modes d'interaction avec l'agent est démontrée, de même qu'une évaluation quantitative de la cohérence des récompenses des utilisateurs et de la densité d'exploration. Bien que l'application ait été positivement reçue par les utilisateurs, l'apprentissage de l'agent pendant l'interaction avec l'utilisateur n'a pas été concluant. La principale raison de cette mise en garde est l'utilisation d'un nouveau paramétrage pour améliorer la réactivité de l'agent lors de l'interaction avec l'utilisateur. Futurs travaux pourraient impliquer la recherche de paramètres appropriés basés sur les récompenses des utilisateurs précédents, le test de différents espaces musicaux à explorer, de nouvelles visualisations d'exploration et d'autres contrôles utilisateurs.

Ce rapport commence par une analyse technique d'algorithmes spécifiques suivie de plusieurs propositions pour l'interaction homme-machine et une évaluation expérimentale impliquant 14 participants. Globalement, il a été démontré que l'utilisation de techniques d'apprentissage par renforcement interactif et profond fournit de nouvelles pistes de recherche intéressantes et offre une valeur ajoutée et complémentaire aux méthodes classiques d'exploration manuelle. En outre, l'utilisation d'un partenaire intelligent implique de nouveaux modes d'interaction et offre des possibilités de façonner et de créer une expérience utilisateur améliorée. Une version finale de l'application sera utilisée pour une installation lors de la prochaine conférence ISMIR 2018 [2] à Paris.

Mots clés: apprentisage par renforcement profond, musique, exploration, interaction homme-machine

[2] http://ismir2018.ircam.fr/

Acknowledgements

My biggest thanks goes out to my supervisors Frédéric Bevilacqua and Hugo Scurto for giving me the opportunity of this internship. Thank you also for the hours discussing ideas and abstract concepts such as exploration, creativity and interaction. I believe these discussions will continue to have an impact on my work. A very big thanks also to Benjamin Matuszewski whom I have often called my hidden third supervisor. Thank you for all our talks and looking forward to bundle forces on something completely useless one day. Finally, a very big thanks to the ATIAM master coordination and my other classmates. You made this year at this extraordinary place possible and for this I will remain forever grateful.

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
DRL	**D**eep **R**einforcement **L**earning
DTAMER	**D**eep **TAMER**
DQN	**D**eep **Q**-**N**etwork
FM	**F**requency **M**odulation
HL	**H**idden **L**ayer
HCI	**H**uman **C**omputer **I**nteraction
IRL	**I**nteractive **R**einforcement **L**earning
IML	**I**nteractive **M**achine **L**earning
ISMM	{ **S**ound **M**usic **M**ovement } **I**nteraction
MSE	**M**ean **S**quared **E**rror
NIME	**N**ew **I**nterfaces for **M**usical **E**xpression
RL	**R**einforcement **L**earning
SGD	**S**tochastic **G**radient **D**escent
VST	**V**irtual **S**tudio **T**echnology

List of Figures

List of Tables

Contents

To my little sister,
who loved discovering music

Introduction

Exploration is something inherent to our lives and allows us to discover new things. However, when confronted with the sometimes overwhelming complexities that surround us, we might lose sight of the bigger picture and miss out on interesting new discoveries. Also, starting an exploratory journey, it is often difficult to structure our exploration, be efficient and keep track of past experience. Consider the case of a digital instrument. Exploring such a large and combinatorial possibility space with non-linear dimensional relations using an often complex interface can be a daunting task. Especially when a user is inexperienced or when dimensional relations change with every new sound or preset mapping. Even more, in such a context, a user is often more occupied with a parametric understanding and more analytical approach instead of focusing on the original result of the exploration namely sound. We can postulate that such an interaction and associated mindset limits discovery, innovative thinking and creativity.

This research investigates to what extent an intelligent system can help a user in the exploration of a timbral space. The objectives of this thesis are to apply and analyse state-of-the-art reinforcement learning models to construct these intelligent systems and design and investigate the human-machine interaction. To this end, this thesis proposes several improvements to an existing reinforcement learning model using interaction controls specifically aimed towards the application of musical exploration.

Two chapters will start this report providing a background and the state-of-the-art. Next, a chapter will focus on the technical and algorithmic side followed by a chapter on the user and the human-machine interaction. Finally, a chapter will present an experiment and its results to evaluate an interactive reinforcement learning model. Specifically, the outline of this report is as follows:

Chapter 1: This chapter presents the context of this work as well as first arguments for the methods used. The research question will be presented followed by an extended motivation for this work based on the existing literature.

Chapter 2: This chapter presents the state-of-the-art in the field of reinforcement learning. It will present new developments such as deep and interactive reinforcement learning as well as discuss current challenges and characteristics. Methods presented here will be used in the next chapter to perform the first analyses related to the musical application.

Chapter 3: The actual use-case is formalised here followed by two technical analyses. First, an analysis of three deep reinforcement learning models is presented to compare their strengths and weaknesses related to our musical application. Second, three interactive models are evaluated leveraging insights gained from analyses earlier on.

Chapter 4: This chapter presents and discusses user controls added to a final model from the state-of-the-art. These controls represent the main contribution of this research to the existing literature.

Chapter 5: Finally, this chapter presents the experimental set-up used to evaluate the model obtained so far. It presents an analysis of the user experience, the human-computer interaction as well as the technical model. The analysis is based on the sounds and trajectories obtained during exploration, the user feedback consistency and the system's learning overall.

This thesis is written during an internship in the {Sound Music Movement} Interaction (ISMM) team at IRCAM under the supervision of Hugo Scurto (PhD candidate in ISMM) and Frédéric Bevilacqua (head of ISMM). It took place in the context of the PhD thesis of Hugo Scurto entitled 'Design of musical interaction by co-exploration'.

Chapter 1

Background

This chapter presents the context, explains the research question and motivates the relevance of this work. After reading this chapter, it should be clear that interactive deep reinforcement learning methods have a validity and unexplored potential as a tool in the (creative) exploration of a musical space.

1.1 Context

Computer science and more specifically machine learning represent the domain for this research. This work will link into the domain of Human-Computer Interaction (HCI), New Interfaces for Musical Expression (NIME) and Sound and Music computing as well.

A use case Consider a user wanting to play a digital music instrument or VST with over 50 knobs, sliders and input screens. One application of this research would be to have the resulting model and algorithm act as an 'artificial partner' next to the user and instrument. This in the sense of recently proposed paradigms 'computer-as-partner' [5] or 'creative tool' [19]. The partner would have control over and explore various knob, slider and input screen settings while the user would provide positive or negative feedback over the resulting sound or timbre and potentially control some of the partner's settings. User's feedback would be incorporated into the partner's exploration strategy and the possible timbral space would be explored more efficiently while also improving the user experience.

Interactive machine learning Recently, new interactive approaches based on machine learning techniques have been introduced and studied in the domain of NIME. Approaches such as 'learning by demonstration' introduced by Caramiaux and Françoise have shown how it is possible to allow users to create new gestural controls based on machine learning techniques using the interactive recording of gestural templates [7][23]. However, while these approaches have proven the usefulness of machine learning techniques in the development of NIME, they also showed the interest of supporting users during both the demonstration (training the model) as well as exploration phase (playing with the model). Learning would then occur from the first moment of interaction and users would be provided with feedback on the learned model during for example the learning of a gesture-sound relationship. These artificial partners have shown to have more capacity than purely in an instructive sense or as a tool to recognise, map or represent data and could also serve as a partner in the creative exploration of a musical space [19] [66].

Reinforcement learning To address the exploration task of a musical space, the PhD thesis of Hugo Scurto proposes to use the reinforcement learning class of machine learning techniques. In this class, an agent learns from feedback received during interactions (taking actions) with an environment in a certain state [73]. The use of reinforcement learning allows to create an adaptive system through the use of the agent that is learning and evolving over time. Next, a user is introduced that interacts with the agent and environment providing for example the feedback for a machine exploring a musical environment. As such, a new co-exploratory relation between user and machine appears, where the first is exploring a possible timbral space and the latter exploring a parameter space. For example, the user

could explore possible sound timbres while the machine explores a parameter space of a digital instrument. Applications of this interactive reinforcement learning paradigm have already shown to be promising in a simple environment with well-defined non-musical, non-creative rules [38], in a simple musical environment focused on rhythm [16] or on musical tension [41]. Non-interactive reinforcement learning has also been used to construct co-improvisers next to a musician [12] [25] or for musical melody generation [35].

Supervised learning One of the biggest difficulties for reinforcement learning remains the so-called 'curse of dimensionality'. As the algorithm learns from interactions with the environment, it becomes infeasible to perform a full exploration of an environment with increasing parameter space dimensions. The algorithm thus needs to generalise knowledge of past experience to unexplored areas of the environment. It is here that supervised learning can be of use as generalisation from training data is one of its main goals. In addition, the combination of supervised and reinforcement learning or so-called deep reinforcement learning also allows automatic feature construction of the input space, better scalability (higher dimensional input data or continuous input), non-linear properties, more flexibility regarding the input data (e.g. the use of images) and to leverage research from a very active research community. As such, supervised learning has obtained a significant improvement of results for reinforcement learning in high-dimensional spaces [49] as well as in an interactive context [10] [77].

1.2 Research Question

With the previous user story in mind, this internship attempts to answer the following high-level question: To what extent can an 'intelligent' system function as partner for a user in the exploration of a musical space? This question is broken down in subcomponents below.

Learning

- Which supervised reinforcement learning model is most suitable for user interaction?

First, as the aim is to construct an 'intelligent' system, machine learning represents the key component in this research. The combination of two machine learning techniques will be used and analysed which are reinforcement and supervised learning. Quantitative metrics such as convergence, generalisation and speed will be used to evaluate their performance. Furthermore, I aim to implement different learning models and assess their flexibility in adapting to a context with user interaction. While supervised reinforcement learning algorithms currently exist in a wide variety of implementations, little research exists with human interaction. Given this gap, a first contribution of this internship will be insights gained of applying user interaction in a supervised reinforcement learning context.

A second contribution concerns the application of supervised reinforcement learning using a generic state representation. More specifically, most reinforcement learning problems integrate a specific coupling between states, agent's actions and feedback. This is the case in for example the cartpole balancing problem where a physical model determines the optimal policy to be learned [4]. In contrast, in a musical setting, an agent tries to learn a policy based on varying sound generation parameters using a feedback signal received from a (probably inconsistent) user. Here, the optimal policy can be much harder to define due to the human factor as well as the distance between sound perception, sound appreciation and the actual parameter space exploration. The state representation and actions in this research are generic in the sense that they directly equal the sound generation parameters with their increments or decrements as possible agent's actions.

Exploration

- How to define and implement an algorithmic exploration strategy suitable for user interaction?

The concept of exploration, learning new things, and exploitation, exploiting past knowledge, for both user and agent represent an important aspect of the research question. Questions related to the user concern whether user attitude is more goal-driven versus more discovery driven. One can also make the distinction between a trajectory and a goal or consider the fact that users might have changing or no goals at all. Questions related to the agent concern the choice of the starting state for exploration, modelling exploration with episodes or with an infinite horizon and the exploration-exploitation trade-off which will be detailed in Chapter 2.

To limit the scope of the research question, I only explore the timbral part of a musical space referring to that specific quality of sounds independent of their pitch, intensity and duration. As such I will not include temporal aspects such as amplitude and pitch envelope or rhythm. This in contrast to research by Derbinsky [16] which investigates a small and non-supervised reinforcement learning model on rhythm. Also, I will model the system's state using a simple parametric representation of a digital music instrument (e.g. using pitch, filter frequency, FM index parameters) and for example not use spectra, spectrograms or any perceptual sound descriptors.

Interaction

- How to design and implement the user feedback signal and user interface?

During development of the artificial partner, I will keep questions concerning the user experience in mind. Combining such a partner with user interaction entails many choices dependent on the user profile and user needs as well as has consequences for the technical implementation.

More specifically, the feedback signal from the user must be defined. Also, I will experiment with the amount and areas of control of the user over the system. Other broader questions remain pertinent during the implementation as well. Should a user intervene or set preferences for exploration? What does the interface look like and how does the environment presents itself to the user (for example only through sound or visual as well)?

1.3 Motivation

This section will take a broad view and argument the validity and opportunity of applying machine learning techniques in an interactive musical setting. It starts with a presentation of the domain of Human-Computer Interaction (HCI), gradually move towards more recent advances such as Interactive Machine Learning (IML) and argue for its application to musical expression.

1.3.1 Human-Computer Interaction

Human-Computer Interaction refers to the domain studying the design and use of computer technology, focusing on the interfaces between people i.e. users and computers[1]. It represents a cross-disciplinary field integrating computer science, behavioural sciences, design as well as cognitive psychology and human factors such as user satisfaction. The term was popularized in the book *The Psychology of Human-Computer Interaction* by Newell et al. published in 1983 and thus entered into the research domain around the time of the introduction of the personal computer and the internet.

While interaction was the research focus, most of the work done focused primarily on the interface. However, as argued by Beaudouin [5], an HCI designer or engineer should design an interaction and not an interface. Focusing on the interaction allows a proper inclusion of the user in both the analysis as evaluation of an HCI. Moreover, it allows the notion of 'computer-as-partner' as opposed to 'computer-as-tool'. This distinction becomes all the more relevant when placed in a larger historical context where we can see interaction evolving alongside human skills and abilities. Looking at interaction evolving from an electrical or mechanical to symbolic, textual and finally graphical nature, we see new skills such

[1] https://en.wikipedia.org/wiki/Human-computer_interaction

as attention and pattern recognition appear. In the same spirit, Dourish introduces the notion of 'embodied interaction' in which tangible computing, with distributed computation and reality augmentation, and social computing, with collaboration and cross-user interaction enhancement, play a central role [17]. Jacucci [33] takes another approach investigating the 'interactive experience' and introduces interaction as performance. While a plethora of terminologies does not support a clear view on HCI, common traits such as embodiment, action, experience and the evolution from passive user to embodied actor illustrate the fact that concentrating on the interface is not sufficient.

Discussing HCI, one should address the notion of exploration or interactive intent. When designing a system with user interaction, one decides over and balances control and flexibility between user and machine. For example, in an information discovery application, a HCI can be designed as to answer search queries by a closest matching example or allow a user to provide feedback and have the system adapt and try to model user intent [61]. While providing more room for discovery, it introduces a new complexity to define the balance between exploration of the information space and exploitation of user feedback while also providing more room for the definition of exploration strategies. Such a system broadens the search problem as well as the possible solution space and takes the perspective of an active user with information needs, information skills and evolving, connected resources [47]. An important metaphor in HCI or in discussions of exploration is the need for 'low floors, high ceilings and wide walls' [59]. It stresses the importance of having easy access, allowing increasingly sophisticated interactions while offering multiple ways to achieve a solution. Exploration as the discovery of new, possible solutions with or without a solution space in mind is a theme that will play a prominent role in this research and will come back into analysis.

Research in design of interactions also links into research of creativity as illustrated by the leading psychologist on creativity Csikszentmihalyi in [13]. He states how interaction is crucial for creativity. To support this argument, in [59], several design principles are listed to support creative thinking. In addition to exploration and the floor, ceiling and wall metaphor, Resnick also lists the need for support of collaboration, open interchange, simplicity and warns for the blind use of black boxes. The last comment is an important one as it has direct consequences for an interactive machine learning application. As such, the choice of a black box or primitive element defines what ideas users can explore with a tool, and what ideas remain hidden. Applied to an interactive machine learning application, the exploration-exploitation parameter is a crucial algorithmic parameter but could be crucial for a user interaction as well. Design principles proposed by Shneiderman [67] confirm the ones previously presented and specifies the need to increase the capacity to locate, study, review and revise solutions as well as the capacity to rapidly generate alternatives and their implications. Finally, design principles by Cockburn can be mentioned here as they allow to avoid the tendency of interfaces to trap users into a 'beginning mode' [11]. He argues how skill acquisition is important for HCI and distinguishes a cognitive, associative and autonomous phase in the interaction. To successfully and lastingly support a user, one should keep these principles in mind already during the development of the system.

1.3.2 Interactive Machine Learning

Interactive Machine Learning (IML) represents a recent research field gaining popularity together with other machine learning techniques such as for example the increased popularity in neural network research. Where the previous section argued for an integrated approach to interaction more from a user perspective, this section takes the argument from the machine's perspective. Its main argument for including user interaction in machine learning is to take advantage of user expertise to improve the generalisation or efficiency of learning systems. While machine learning techniques beat humans at finding patterns or correlations in data, machines remain limited to inductive predictions requiring human interaction for interpretation, comparison or turning data into actionable insights.

It is for this reason Amershi stresses the importance of studying users alongside the development of machine learning algorithms [2]. According to him, users should be included in both the exploratory development stage and later refinement stages. Doing so, it is important to keep the user experience in mind and remember how users want to guide and

demonstrate and not just provide simple feedback. In addition, end users may value transparency over a black-box and may want to specify preferences on errors or ask why something happened the way it did. In a concrete example of having expert users experiment with an interactive, musical machine learning system, other user requirements such as the need to reduce barriers to speed and exploration, the appreciation of abstraction (i.e. black-boxes) and the access to surprise and discovery arose [21]. In addition and as a specific argument for this research, the experiment also showed how goals and user requirements evolve during interaction and showed user frustration when an understanding of a static system later proved to be false.

In a different application, both Fiebrink and Scurto argue for the potential of a human centred view of machine learning and its impact as a creative tool [19][66]. Fiebrink distinguishes musical applications between recognising, mapping, tracking, representing and collaborating and adds the use of machine learning as creative tool. As the latter, a machine learning tool can be trained to represent an 'imperfect mirror' of a user [55] allowing him or her to engage in creative expression without being obstructed by limited expertise of an instrument and to enter a state of flow as describe by Csikszentmihalyi [14].

When developing and designing an IML system, new methods for evaluation must go along with it. While quantitative metrics remain valid from an algorithmic perspective, subjective criteria and complementary exploratory and objective metrics are needed as well [20]. In the context of evaluating a creative tool, in [44], Lowgren argues for the need of an extended vocabulary including terms as playability, seductivity, fluency, ambiguity (as a positive property) and engagement instead of usability.

1.3.3 New Interfaces for Musical Expression

Having discussed interaction from a user and machine perspective, this section introduces the musical element. New Interfaces for Musical Expressions or NIME is the cross-disciplinary field dedicated to research on the development of new technologies and their role in musical expression and artistic performance[2]. It encompasses both technological disciplines, such as electronics, sound synthesis and processing, software engineering, as human behavioural disciplines, such as psychology, physiology, ergonomics, HCI among others.

Developing and designing new musical instruments addresses both its control, sonic and musical properties as its essence in the form of its implications and experience [36]. Developing and designing NIME requires a focus on sound, interactivity, streamed-based and dynamic spectro-morphology ([70]) as opposed to notes, reactivity, event-based and static sound descriptors [56]. Considering the computer as a meta-instrument and decoupling controller from sound engine, many opportunities in the diversity, improvisation and expressiveness arise. However, as great as the opportunities in new instrument development are as small as their adoption among new users is [36].

Learning from past experience, in [36] Jorda lists 25 guidelines that can aid in the design and evaluation of NIME. While an extended discussion is not possible here, simple guidelines such as 'wider rather than deeper' and 'interfaces with representation and control' are very powerful and to be kept in mind. Other useful observations from research are the clear relation between the degrees of freedom of an instrument and user exploration [81] and the difference between generatively and explicitly defined mappings in [31]. Finally, the broad exploration and later fine-tuning or evaluation of alternatives in the design of a mixer interface in [9] and mentioned in [66] could present a needed feature in the development of an 'intelligent partner'.

Finally, other work has been done combining NIME with machine learning techniques. In [8], a comprehensive overview is given of the different classification and regression methods used to model musical gestures and map gestures to sound. Regression techniques are mentioned to perform dimensionality reduction useful in for example instrument parameter mapping or visualisation, or to perform gesture representation, cross-modal analysis and control. Several classification techniques are listed on different neural network, Hidden Markov or Gaussian Mixture Models to perform multi-parametric control, gesture analysis or recognition. In [24], complex probabilistic models are presented to design motion

[2] https://en.wikipedia.org/wiki/New_Interfaces_for_Musical_Expression

sound relationships or define mappings by demonstration with example applications such as gesture-based sound mixing or interactive vocalisation. Schacher in [62] mentions machine learning techniques as a generative and interactive tool for recognition of key musical moments, adaptive control mapping and high-level feature classification for autonomous music generation. All these techniques however powerful still have common weaknesses of which some this research will try to address. More specifically, while supervised methods remain good for customisation and retraining, their slow learning curve obstructs free improvisation. While they are strong in pattern recognition, obtaining understandable representations often remains challenging. In general, real-time training with few examples and high adaptability are still out of reach. An advantage of reinforcement learning on the other hand is that it does not require labelled examples from some knowledgeable external supervisor. It does not learn from examples of desired behaviour that are both correct and representative for the full set of situations in which an agent has to act. It learns from its own experience and it is this last trait that this research will leverage to investigate in a musical interactive application.

Chapter 2

Reinforcement Learning: State-of-the-Art

One of the first ideas that might come up when thinking about the nature of learning is the idea of interacting with our environment. Reinforcement learning takes this idea as its base. It is learning what to do (the policy), how to map situations or states to actions, to maximise a reward signal. A general model is shown in figure 2.1.

Reinforcement learning differs from other branches of learning such as supervised and unsupervised learning. Where supervised learning focuses on generalising from representative and correct examples, the goal of reinforcement learning is to learn from its own experience. Unsupervised learning on the other hand typically tries to find hidden structure in unlabelled data.

Also, reinforcement learning is part of a larger trend towards simple, general principles in artificial intelligence. It considers the whole problem and has to address the problem of planning and real-time action selection as well as how environment models are acquired and improved.

This section constitutes a large part of this thesis and for good reason. It will introduce the theory, the models and algorithms that will be used to develop and test our own implementations later on. The majority of this section is based on the work of Sutton and Barto [73] while gradually building upon more recent research as cited in the text.

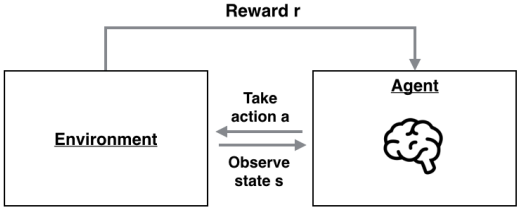


FIGURE 2.1: General reinforcement learning model

2.1 Theoretical Basis

The theory of reinforcement learning has close links with dynamical systems theory and more specifically, the optimal control of incompletely-known Markov decision processes. Informally, it concerns a learning agent interacting over time with its environment to achieve a goal. The agent must be able to (partly) sense the state of its environment and must be able to take actions that affect the state. The goal or goals must be defined in relation to the state of the environment. Any method to solve such a problem can be considered a reinforcement learning method.

Markov decision process A Markov Decision Process (MDP) is a classical formalisation of sequential decision making, where actions influence not just immediate rewards, but also subsequent states and through those future rewards. As such, MDPs involve delayed reward and the need to trade-off immediate and delayed reward. Specifically, a Markov decision process is a discrete time stochastic control process. At each time step, the process is in some state s, and the decision maker ,from hereon called agent, may choose any action a that is

available in state s. The process responds at the next time step by moving into a new state s', and giving the agent a corresponding reward $R_a(s)$.

In a finite MDP, the sets of states, actions and rewards ($\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$), all have a finite number of elements. In this case, the random variables R_t and S_t have well defined discrete probability distributions dependent only on the preceding state and action. As such, for particular values of these random variables, there is a probability of those values occurring at time t, given particular values of the preceding state and action:

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{2.1}$$

for all $s', s \in \mathcal{S}, r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$. From this, one can obtain knowledge of the environment such as the state-transition probabilities

$$p(s'|s, a) = Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a) \tag{2.2}$$

or the expected rewards for state-action pairs:

$$r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a) \tag{2.3}$$

Value function and policy Almost all reinforcement learning algorithms involve estimating functions of states (or state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state). These functions are called value functions with the notion of good determined in terms of expected return. Expected return G can be defined as a discounted sum of future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.4}$$

where gamma, $0 \leq \gamma \leq 1$, as discount rate determines the present value of future rewards.

As these rewards depend on what actions an agent will take in the future, value functions are defined with respect to particular ways of acting, called policies. A (stochastic) policy π is a mapping from states to probabilities of selecting each possible action and noted as $\pi(a|s)$. The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter. Formally, the state-value function is defined as

$$v_\pi(s) = \mathop{\mathbb{E}}_{\pi}[G_t|S_t = s] = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s\right], \quad \text{for all } s \in \mathcal{S} \tag{2.5}$$

Similarly, the action-value function for policy π is defined as

$$q_\pi(s, a) = \mathop{\mathbb{E}}_{\pi}[G_t|S_t = s, A_t = a] = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s, A_t = a\right], \quad \text{for all } s \in \mathcal{S} \tag{2.6}$$

Bellman equation and optimality Value functions satisfy recursive relationships expressed by the Bellman equation and this fundamental property is used throughout reinforcement learning. It states that the value of a start state must equal the discounted value of the expected next state, plus the reward expected along the way.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)[r + \gamma v_\pi(s')], \quad \text{for all } s \in \mathcal{S} \tag{2.7}$$

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')] \tag{2.8}$$

Now, solving a reinforcement learning task means finding a policy that achieves maximum reward over time. For every problem there exists at least one optimal policy that is better than or equal to all other policies. They all share the same optimal state-value and

action-value function which formulated as the Bellman optimality equation become:

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s']] \tag{2.9}$$

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \tag{2.10}$$

Once one has v_*, the optimal policy follows automatically as actions that appear best after a one-step search will be optimal actions. With q_*, one can simply take any action that maximizes $q_*(s, a)$ to find the optimal action. The latter is called greedy action selection.

Value based methods Out of the previous definitions, one can now concentrate on finding the optimal policy. One approach called 'value-based methods' do this alternating between two simultaneous, interacting processes. One making the value function consistent with the current policy called policy evaluation, and the other making the policy greedy with respect to the current value function called policy improvement. The majority of reinforcement learning algorithms differ in the way they approach each of these processes.

Two extreme cases of policy evaluation are Monte Carlo and dynamic programming methods which are respectively averaging sample based or model-based. Without any knowledge of the environment, one can find the value function for a specific policy averaging returns for every state as the number of visits for every state goes to infinity. Model-based methods assume the dynamics of the environment are known and value functions can be found using the Bellman equation for v_π as an update rule:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]], \quad \text{as } k \text{ goes to infinity} \tag{2.11}$$

Policy improvement helps to find better policies which should eventually lead to the optimal one. This is done using the policy improvement theorem for which a proof can be found in [73] and copied in Appendix A. It shows that one improves an original policy by making it greedy with respect to the value function of the original policy. Formally, this means

$$\pi'(s) = argmax_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big] = argmax_a q_\pi(s, a) \tag{2.12}$$

One now has all the tools needed to develop an iterative reinforcement learning algorithm. An example of a full Monte-Carlo method would be to average returns for every visited state-action pair, update its estimate of the action-value function and update the policy based on greedy action selection with reference to its new action-value function. A fully incremental dynamic programming method called value iteration combines the policy evaluation and policy improvement steps in one update rule:

$$v_{k+1}(s) = max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big] \tag{2.13}$$

In between dynamic programming and Monte Carlo methods, there exist temporal difference methods. Like Monte-Carlo methods, they can learn directly from raw experience without a model of the environment. Like dynamic programming, they update their estimates based in part on other learned estimates, without waiting for a final outcome. This process of so-called bootstrapping provides the advantage of on-line learning but also increases variance and affects convergence towards an optimal policy. One example, called TD(0) makes the update

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big] \tag{2.14}$$

immediately on transition to S_{t+1} and receiving R_{t+1} with $V(S_t)$ the estimated value for state S_t. While for Monte Carlo methods, the update target would be G_t, here it is $R_{t+1} + \gamma V(S_{t+1})$. This policy evaluation step is then followed by a policy improvement by making

action selection greedy with reference to the new value function. It is these methods that will become the focus of the discussion as well as the basis for implementations later on.

Policy gradient methods Next to value-based methods, another approach to improve a policy is to compute a gradient to increase the probability of taking a good action $\pi(s) = a$. Informally, to improve a policy, one must modify $\pi(a|s)$ to increase the probability of a when $q_{\pi'}(s, a) > v_\pi(s)$. By successively applying this process, one eventually obtains the optimal policy. For the simplest policy gradient algorithm, improvement of a policy is guaranteed for sufficiently small step sizes and convergence to a local optimum for decreasing step size. However, updates can have a large variance and slow down learning significantly. As such, improved policy gradient methods called 'actor-critic' methods will be introduced later on to speed up learning while weakening theoretical convergence properties.

2.2 Characteristics

There are many reinforcement learning algorithms because of their different trade-offs (sample efficiency, stability, ease of use) and assumptions (stochastic or deterministic policies, continuous or discrete state-action spaces, episodic or continuing tasks). This section will present some important characteristics of these algorithms.

Rewards The feedback or reward signal represents a key component of reinforcement learning. It is represented using a reward function, can be sparse or frequent, (in-)consistent over time and be provided by the environment or a user. If the reward function is assumed to be dependent on a whole sequence of state-action pairs, one can implement memory-based policies with recurrent neural networks [29]. Also, one can include a specific reward distribution in the model to allow for user feedback to be distributed over multiple state-action pairs in time.

Several approaches have been taken to deal with different reward signals or depending on if we wish to include or to learn a reward function. One approach called 'imitation learning' tries to learn to perform a task from expert demonstrations. This approach includes 'inverse reinforcement learning' that attempts to learn an unknown reward function [53]. Another approach called 'apprenticeship learning' does not learn a reward function and directly maps expert trajectories to a policy [1]. Other approaches leverage user interaction and use user preferences over state trajectories [10], use only user feedback [77], or merge user and environmental feedback [58]. Interestingly, the latter concatenates information about the environment next to information about the user providing a tight environment-user-control 'state'-representation.

On- versus off-policy The goal of a reinforcement learning algorithm is to find a (sub)optimal policy to maximize reward. However, the policy used to explore the state space, the so-called behaviour policy, can be different from the policy to be learned or target policy. Algorithms with identical or different behaviour and target policy are called respectively on-policy and off-policy algorithms. For example, an off-policy algorithm could for its behaviour policy randomly take actions for a certain percentage of the time while storing new and improved estimates in its target policy that does not include any randomness.

Off-policy algorithms often have greater variance and slower convergence as their training data is due to a different policy. They are however more powerful and general as they allow to separate exploration from learning. They also include the on-policy algorithms as a special case when policies are the same. With off-policy methods a first challenge is related to the fact of having two different data distributions (behaviour and learned policy). A second challenge is related to the fact that the off-policy distribution of the updates is not according to the on-policy distribution when using function approximation. Techniques such as importance sampling deal with the different distributions, while techniques such as experience replay deal with the high variance in updates and will be presented with the DQN algorithm.

Exploration and Exploitation A central topic in reinforcement learning is the exploration-exploitation trade-off. It refers to the balance between taking 'risky' actions to search for new information and taking 'safe' actions by exploiting obtained information about past actions. The dilemma for the learning methods presented here is that they seek to learn action values conditional on subsequent optimal behaviour, but they need to behave non-optimally in order to explore all actions or to maintain exploration.

One simple solution to the problem of maintaining exploration is the assumption of exploring starts. This assumption is valid when training episodes start in a state-action pair and that every pair has a nonzero probability of being selected at the start. Another solution is to define *ϵ-soft* policies which are policies with $\pi(a|s) \geq \frac{\epsilon}{|\mathcal{A}(s)|}$ for all states and action and for some $\epsilon > 0$. A specific example is the *ϵ-greedy* policy that most of the time chooses an action that has maximal estimated action value (i.e. the greedy action), but with probability ϵ they select an action at random. One often used technique using ϵ-greedy exploration strategies is the technique of ϵ-decay which takes an ϵ_d, a start ϵ_s and end value ϵ_e and lets ϵ evolve according to

$$\epsilon = \epsilon_e + (\epsilon_s - \epsilon_e)e^{-\frac{t}{\epsilon_d}} \quad \text{for t} > 0 \tag{2.15}$$

2.3 Reinforcement Learning with Function Approximation

With increasingly large and combinatorial state spaces, finding an optimal policy or value function becomes infeasible even with infinite time or data. In such cases, the goal becomes to find a good approximate solution using limited computational resources. Using function approximation, one tackles the issue of generalisation which allows to infer knowledge from previously visited states to unencountered states.

The goal of our function approximation is represented by parametric versions of earlier introduced value or policy functions from hereon referred to as $q(s, a, w) \approx q_\pi(s, a)$ and $\pi(a|s, \theta) \approx \pi(a|s)$. We focus in this research on non-linear function approximation using Artificial Neural Networks (ANNs) although linear methods can be used as well. ANNs have several advantages over the use of linear methods as they allow for automatic feature construction, have non-linear basis functions and extend naturally to larger state spaces. For example, with linear methods one has to define features as basis functions for the linear weight vector without taking into account interaction between features.

Depending on whether we are approximating value or policy functions, different objectives and optimisation schemes are used. However, the previously introduced policy evaluation and improvement schemes remain the same. In addition, with function approximation and the assumption to have many more states than function parameters, an update at one state affects many others. As such, we must specify a state distribution $\mu(s) \geq 0, \sum_s \mu(s) = 1$, representing how much we care about the error in each state.

Value based methods Value based methods try to approximate the state-action value function q using a Mean Squared Error (MSE) objective function:

$$MSE(w) = \sum_{s \in \mathcal{S}} \mu(s) \Big[q_\pi(s, a) - q(s, a, w) \Big]^2 \tag{2.16}$$

Assuming states appear in examples with the same distribution μ over which we are trying to minimize the MSE, we can use stochastic gradient-descent method to reduce the error:

$$w_{t+1} = w_t + \alpha \Big[q_\pi(S_t, A_t) - q(S_t, A_t, w_t) \Big] \nabla q(S_t, A_t, w_t) \tag{2.17}$$

Finally, we can use any approximation for $q_\pi(S_t, A_t)$ as the update target. For example, using the TD(0) method introduced earlier but applied to the state-action function, one can complete the update with:

$$w_{t+1} = w + \alpha \Big[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}, w_t) - q(S_t, A_t, w_t) \Big] \nabla q(S_t, A_t, w_t) \tag{2.18}$$

This equation is an important one and we will use it later to obtain the so-called SARSA algorithm. Inspecting the previous equation, one can see that the update target depend on the current value of the weight vector w_t. This implies that they will be biased and will not produce a true gradient-descent method. They take into account the effect of changing the weight vector w_t on the estimate, but ignore its effect on the target. They include only a part of the gradient and because of this are called semi-gradient methods.

Policy gradient methods Policy based methods directly learn a parametrised policy allowing them to select actions without consulting a (action or state) value function. They have the advantage of being able to approach a deterministic as well as find a stochastic optimal policy and offer the possibility of injecting prior knowledge about the desired form of the policy. They are based on the gradient of some performance measure $J(\theta)$ dependent on the policy parameter θ:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t). \tag{2.19}$$

The performance measure J can be based on the value of the starting state or the average rate of reward per time step (see 'Continuing tasks' below). In both cases, methods are based on the policy gradient theorem which is found in [73] (and copied in Appendix B). The theorem provides an analytical expression for the gradient of performance with respect to the policy parameter

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_\theta \pi(a|s,\theta) \tag{2.20}$$

with $\mu(s)$ the on-policy state distribution under π.

Also in Appendix B, this expression can then be converted in an expectation that allows sampling and consequently an iterative update:

$$\nabla J(\theta) = \mathbb{E}_\pi \left[G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \right] \tag{2.21}$$

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)} \tag{2.22}$$

Actor-Critic methods Previous value and policy based methods can also be combined to reduce variance and speed up learning. These actor-critic methods require the approximation of both the value (critic) as policy function (actor).

Specifically, one can include a baseline $v(S_t, w)$ to reduce variance in updates and bootstrap to approximate G_t as well as allow incremental and online updates in 2.22.

$$\theta_{t+1} = \theta + \alpha \Big(R_{t+1} + \gamma v(S_{t+1}, w) - v(S_t, w) \Big) \frac{\nabla_\theta \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)} \tag{2.23}$$

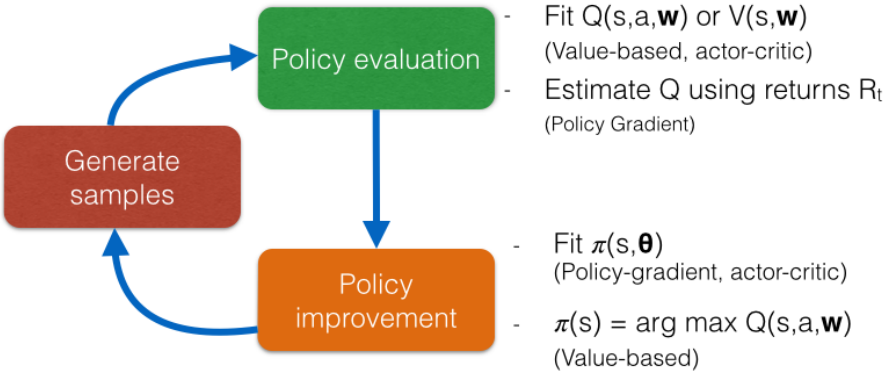


FIGURE 2.2: Reinforcement learning methods

Episodic versus continuing One important distinction between reinforcement learning algorithms is based on the choice of working with a finite or infinite horizon model. A finite horizon or episodic model assumes the existence of a terminal state and makes sense when the agent-environment interaction breaks naturally into subsequences such as for example with plays of a game. An infinite horizon or continuing model assumes tasks to go on forever and is useful for example in an application with a robot with a long life span.

When using approximate methods in reinforcement learning for continuing tasks i.e. a tasks that do not break down in identifiable episodes, one needs to adapt the objective functions for both value as policy based methods. Now, the goal is not to maximize the value of a starting state but to maximize the average reward over time. To transform the results from the previous section to the continuing case, one simply removes all discounting factors and replaces all rewards by the difference between reward and the true average reward over time. As such, the update rule for value-based functions 2.18 becomes:

$$w_{t+1} = w + \alpha \Big[R_{t+1} - \bar{R}_{t+1} + q(S_{t+1}, A_{t+1}, w_t) - q(S_t, A_t, w_t) \Big] \nabla q(S_t, A_t, w_t) \tag{2.24}$$

with $\bar{R}_t$ the average reward over time at time t. For policy-gradient methods, the rule 2.23 becomes:

$$\theta_{t+1} = \theta + \alpha \Big(R_{t+1} - \bar{R}_{t+1} + v(S_{t+1}, w_t) - v(S_t, w) \Big) \frac{\nabla_\theta \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \tag{2.25}$$

2.4 Deep Reinforcement Learning

While Reinforcement Learning (RL) can reason about decision making, it is deep models that allow reinforcement learning algorithms to learn and represent complex input-output mappings. Deep models in this research refer to ANNs with multiple hidden layers between the in- and output layers and are used as function approximators in reinforcement learning algorithms. As mentioned earlier, many approximators can be used to learn the policy and state(-action) value functions. The very big majority of the Deep Reinforcement Learning (DRL) community today however prefers ANNs for their automatic feature construction, fast implementation, non-linear approximation and closer workings to human and behavioural learning.

Having introduced the theoretical basis of reinforcement learning with the use of function approximation, this section will present a concise overview of specific algorithms. These algorithms represent the state-of-the-art and are distinguished whether their optimisation method is value- or policy-gradient based.

Value-based methods Earlier introduced temporal-difference learning [72] represents a class of classical algorithms in reinforcement learning. Implementing temporal-difference learning using a state-action function, one obtains the so-called SARSA algorithm [60]. The update rule for SARSA with function approximation was presented with equation 2.18 and can be implemented using deep ANNs [82]. Deep SARSA (DSARSA) will be the first to be used in later implementations. Its algorithm can be found in appendix C and its model in figure 2.3.

A generalisation of SARSA as an off-policy algorithm is called Q-learning [78]. As discussed earlier, off-policy methods use a different policy for exploration than the one being learned. Instead of using a state-action value approximation in 2.18, Q-learning maximizes over the full one-step action space:

$$w_{t+1} = w + \alpha \Big[R_{t+1} + \gamma max_a q(S_{t+1}, a, w_t) - q(S_t, A_t, w_t) \Big] \nabla q(S_t, A_t, w_t). \tag{2.26}$$

Q-learning and its ANN implementation deep Q-learning (DQN) [50] have been responsible for some of the greatest advances in (deep) reinforcement learning [49]. Several algorithmic improvements have been added to DQN and have improved performance further. A technique called 'experience replay' stores experience (state-action-reward pairs) and retrains

the network using randomly sampled mini-batches of previous data [49]. This resampling decorrelates the input data and improves the algorithm's sample efficiency. Double DQN (DDQN) duplicates the value-function network to separately use a network for the action selection and the value estimation update [74]. This avoids over-estimation of values through the maximisation operation in 2.26.

$$w_{t+1} = w + \alpha \Big[R_{t+1} + \gamma q(S_{t+1}, argmax_a q(S_{t+1}, a, w'_t), w_t) - q(S_t, A_t, w_t) \Big] \nabla q(S_t, A_t, w_t) \tag{2.27}$$

Finally, other research decomposed the state-action value function $q(s,a)$ in a state-value function $v(s)$ and an advantage function $a(s,a)$ to construct a so-called duelling neural network architecture [75]. The advantage function quantifies how good an action is as compared to the average reward over all actions. This allowed to include a notion of the state-value independent of an action taken and showed to improve performance of DDQN. Specifically, it means decomposing the final neural network layer in two separate outputs and recombining them afterwards using the following update rule:

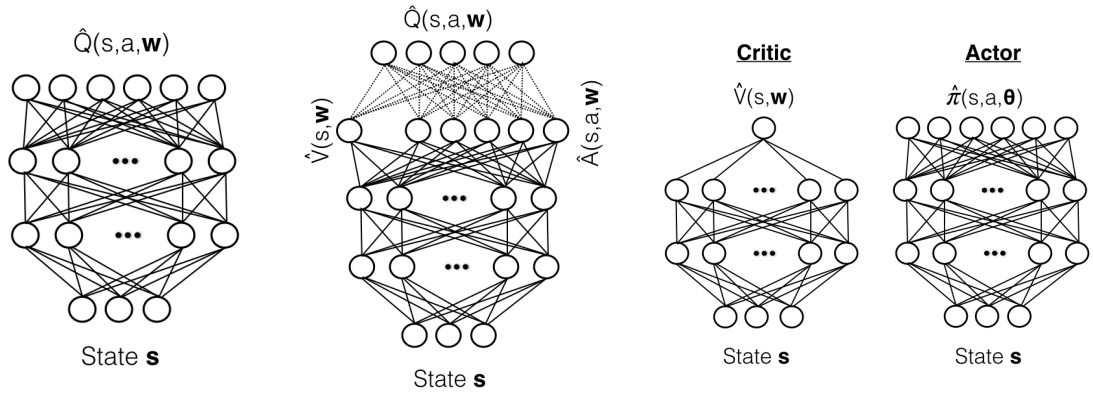
$$Q(s,a,w) = V(s,w) + (A(s,a,w) - \frac{1}{\mathcal{A}} \sum_{a'} A(s,a',w)) \tag{2.28}$$

It is this last implementation called the Duelling Double Deep Q-network (DDDQN) that will represent the second algorithm to be used later on. The implementation follows the DSARSA algorithm with a new update rule as in 2.27 and using slower updating of the second or double ANN. An illustration of the ANN for DDDQN is shown in figure 2.3.

Policy-gradient methods Policy gradient methods have the advantage of directly optimising towards the goal, the optimal policy. However, these methods remain very sample inefficient, have high variance in their updates and have a high sensitivity to the step-size in its gradient-descent optimisation. One of the first popular policy-gradient algorithms is called REINFORCE [80] and is a direct implementation of equation 2.22. Later implementations extended policy gradient methods to continuous action spaces [68], off-policy methods [42] and stochastic policy methods [28]. A popular method called 'Trust Region Policy Optimisation' to adapt and limit the gradient updates was developed to address the step-size sensitivity [64].

Actor-Critic methods As mentioned before and shown in 2.23, Actor-Critic methods introduce a baseline with policy-gradient methods to reduce the variance of its updates and accelerate learning [15]. These represent an important and popular class of algorithms for deep reinforcement learning. With considerable problem-specific adjustments, it is this method that was used as the first implementation to beat an expert player in the board game Go [69]. Other improvements include the addition of a replay memory [76] and asynchronous Actor-Critic methods [48] that run multiple agents in parallel to train on decorrelated data inputs. In the latter, the author also introduced the use of an advantage function approximated by a temporal-difference error as a baseline. It is a synchronous version of this method called 'Advantage Actor-Critic' (A2C) that will be used as a third implementation later on. The A2C algorithm again follows the DSARSA algorithm from before but with a new update rule as in 2.23 and using two ANNs. A first ANN approximates the value-function $V(s,w)$ using the value-error (shown between brackets in 2.23). A second network approximates the policy $\pi(s,\theta)$ using a logarithmic softmax[1] ANN output and using the identity $\nabla ln(x) = \frac{\nabla x}{x}$.
An overview of the ANNs structure is shown in the figure below:

[1] A normalised exponential function to transform the network output into normalised probabilities (see https://en.wikipedia.org/wiki/Softmax_function)



(A) DSARSA (B) DDDQN (C) A2C

FIGURE 2.3: Deep reinforcement learning ANN overview

Exploration bonus The exploration strategies seen so far, specifically ϵ-greedy exploration, represent the most basic of exploration strategies for the state(-action) space and more sophisticated techniques with and without deep ANNs have been developed. A method that will be used in implementations later on is based on the notion of optimistic exploration (the assumption that the unknown is good) and the addition of an 'exploration bonus' to the reward. As shown in [6], this exploration bonus can be based on an estimation of state-visitation counts called the pseudo-count $\hat{N}(s)$ and a total visit pseudo-count n using a density model $\hat{p}_\phi(s_i)$. Formally, one calculates the total reward with bonus r_i^+ as

$$r_i^+ = r_i + \beta \sqrt{\frac{1}{\hat{N}(s_i) + C}} \tag{2.29}$$

with

$$\hat{N}(s_i) = \hat{n} p_\phi(s_i) \tag{2.30}$$

$$\hat{n} = \frac{1 - p_{\phi'}(s_i)}{p_{\phi'}(s_i) - p_\phi(s_i)} p_\phi(s_i) \tag{2.31}$$

with β and c pre-defined constants.

Also in [6], it was shown that the pseudo-counts have close links to another class of methods based on 'intrinsic motivation' to improve the exploration strategy. They suggest to explore what is surprising, typically based on a prediction error using the target estimates from before or based on information gain. Other methods are based on Thompson sampling [71], bootstrapping neural networks for deep exploration [54] or by adding parametric noise to network weights [22]. Approaches such as Thompson sampling have been used to find an appropriate exploration-exploitation balance but require a prior distribution. Bayesian methods can even be used to compute an optimal exploration-exploitation balance but often require much too great computation resources for the combinatorial problems considered in reinforcement learning. These approaches could benefit from approximate methods and are the topic of current research.

2.5 Interactive Reinforcement Learning

Interactive Reinforcement Learning (IRL) or human-centred reinforcement learning refers to the reinforcement learning problem where rewards to teach an agent are generated by a human trainer instead of a (stationary) environmental reward function. This section will focus on state-of-the-art methods involving only human feedback without environmental reward as these are the methods that will be used in our implementations later on.

A first simple approach to IRL is to directly include human feedback as the reward for traditional RL algorithms. This approach can achieve good results if the feedback is stationary and intended to be maximized [32] [57]. Unfortunately, the assumption of stationarity is too strong for our application and in addition can lead to unintended agent behaviours [39] [30].

A special IRL case applied to a musical setting is by Derbinsky [16]. He trains agents to learn user rhythms and represents a state as the presence or absence of a beat. Unfortunately his model does not scale well without deep ANNs and has the need for a demonstration phase.

One of the main challenges in reinforcement learning is the 'delayed reward' challenge which represents the difficulty of assigning a reward to actions arbitrarily far in the past. Instead of modelling the human feedback as a reward, one can also model the feedback as a direct comment on agent's behaviour. Algorithms such as TAMER [38], Deep TAMER [77] and COACH [45] use human feedback as a direct label of respectively the action-value or advantage function. Algorithms such as SABL [43] and Policy Shaping [27] directly label the optimality of actions of the policy. In this way these algorithms circumvent the delayed reward challenge assuming a human trainer can judge the overall optimality or quality of agent's actions.

(Deep) TAMER The TAMER and Deep TAMER algorithm can be seen as value-based algorithms. They have been applied in settings that allow to quickly learn a policy on episodic tasks (small game environments or physical models) and aim to maximise direct human reward. This opposed to the traditional RL training objective to maximise the discounted sum of future rewards. These algorithms learn the human reward function R using an ANN and construct a policy from R taking greedy actions. In addition, to accommodate sparse and delayed rewards from larger user response times, the algorithms include a weighting function $u(t)$ to past state trajectories and a replay memory in the case of Deep TAMER. Specifically, while traditional RL algorithms aim to optimise the loss

$$MSE = \Big[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}, \mathbf{w}_t) - q(S_t, A_t, \mathbf{w}_t)\Big]^2, \tag{2.32}$$

(Deep) TAMER aims to optimise

$$MSE = u_t(t_f)\Big[r_t(t_f) - \hat{R}(S_t, A_t))\Big]^2 \tag{2.33}$$

with $r(t_f)$ and $u_t(t_f)$ respectively the user-provided feedback and weighting function at time t_f. One drawback of this method is the fact that exploration is now implicitly defined as we only learn a reward function on which to base our exploration strategy. In the TAMER implementations, exploration is further simplified to a simple greedy action selection based on the reward function leaving exploration only to erroneous state-action reward estimates. As a concrete example and to get an idea of parameters in the problem setting of [77], Deep TAMER achieved above human player's performance in the Atari Bowling game with following specifications:

Training time	15 minutes
Data frequency	20 frames/second
Input data	160×160 images
ANN	Auto-encoder outputsize = 100 and fully-connected 2-layer ANN of size 16 and 4

$\lvert$Feedback signals$\rvert$	1000
$\lvert\mathcal{A}\rvert$	4
$\lvert\mathcal{S}\rvert$	100
$\lvert$Training cycles$\rvert$ (feedback + replay mini-batches)	3500

TABLE 2.1: Deep TAMER Atari Bowling game specifications

COACH A different method was developed with the COACH algorithm [45]. The algorithm directly learns and approximates the policy function $\pi(.)$. It removed an assumption of policy-independent feedback (see [45]) and modelled feedback using the estimation

error in equation 2.23. This feedback is then considered as an unbiased estimator of the advantage function [63] in an actor-critic algorithm. One can thus replace the error by the user provided feedback r_t:

$$\theta_{t+1} = \theta + \alpha r_t \frac{\nabla_\theta \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \tag{2.34}$$

With policy-dependent feedback, COACH also affords new training strategies:

- Diminishing returns: decrease positive feedback for good actions as agent adopts those actions
- Differential feedback: Vary the magnitude of feedback w.r.t. the degree of improvement or deterioration in behaviour
- Policy shaping: provide positive feedback for suboptimal actions that improve behaviour and then negative feedback after the improvement has been made

Others Other algorithms attempt to allow user feedback on the agent's actions in the sense of 'that was right' or 'that was wrong'. In the Policy Shaping algorithm [27], feedback is first converted into a policy and then combined with an underlying reinforcement learning policy. The model developed in [43] allows taking into account a trainer's feedback strategy. It's SABL algorithm can adapt to different users and learn in the absence of feedback. Because in our problem setting I assume actions of an agent can not be evaluated as 'optimal' by a user and the user can be inconsistent, these algorithms will not be detailed any further.

2.6 Challenges

The reason for the many implementations presented so far are the difficulties posed by (deep) reinforcement learning and the use of various techniques to stabilise, optimise and generalise learning. The root cause for these difficulties is the dynamical nature of deep reinforcement learning. This means its online data-collection with as only supervision a single scalar as reward. Its methods use random chance for data-collection which adds a great difficulty next to already existing machine learning challenges such as model issues, implementation issues and algorithmic issues. Sample inefficiency is another big challenge because of which on-policy methods among others perform very poorly. Briefly mentioned so far is the problem of stability of algorithms because of a high variance of gradient updates and a high sensitivity to parameter initialisation. The latter could be determining convergence and thus poses another challenge in debugging implementations. Local optima pose a challenge as well. If one gets the exploration-exploitation trade-off wrong, an algorithm may explore too much, get junk data and learn nothing or may exploit too much and burn-in non-optimal behaviours. Finally, this trade-off also has large consequence for the generalisation capacities of an algorithm. It must be noted that next to generalisation, overfitting may not always pose a problem for reinforcement learning as an agent can always go collect more samples to train on. The difficulties can be summarised as follows:

- Random chance for data-collection
- Sample inefficiency
- Stability
- Local optima

Several important topics such as the ones above have only been included briefly in the discussion so far. While important results do exist in the literature, these topics still remain a big area for current research. They are not further discussed here too focus on providing the reader a theoretical base without too many in-depth analysis. Additional details on these topics will be given with results of algorithmic implementations later on.

Chapter 3

Model Development

This chapter will present deep reinforcement learning implementations and some early experimental results to address the research question from section 1.2.

The chapter's outline is as follows:

3.1 Problem Formalisation: A first section formalises the question of musical exploration into a reinforcement learning problem. It aims to clarify the problem's constraints.

3.2 Analysis of Deep Reinforcement Learning: This section presents three deep reinforcement learning models with distinct properties (DSARSA, DDDQN, A2C respectively on-policy, off-policy value-based and actor-critic) applied to the problem formalised in the first section. It first presents analyses using a synthetic reward function and synthetic goal followed by results of a hyperparameter search. It aims to provide insights into each models' strengths and weaknesses and properties such as convergence, generalisation and parameter sensitivity with regards to our problem setting.

3.3 Analysis of Interactive Reinforcement Learning: This section first presents remaining challenges for our problem setting. Next, it presents the evaluation of three interactive deep reinforcement learning models which can be considered as interactive versions of the ones in the previous section (interactive DSARSA, DTAMER, COACH).

3.4 Discussion and Future Work

3.1 Problem Formalisation

Figure 3.1 shows a schematic overview of both the non-interactive as interactive problem setting involving a user. For both cases, the environment's state s_t is first communicated to the agent. The agent consults its policy π and decides on taking action a. The environment transitions to a new state s_{t+1} and the agent receives this new state together with reward r. Next, the agent stores the action, state and reward couple in its replay memory and learns from this new information by updating its estimated reward function $\hat{R}$. The only difference between the non-interactive and interactive case is the way the reward is provided. In this chapter, I will work with a synthetic reward function $R(x, s_t, s_{t+1})$ even when analysing interactive deep reinforcement learning models. The user and its feedback will be included and discussed in chapter 3 and 4. A video used for a submission at the ISMIR conference and demonstrating the application can be found at `https://youtu.be/z7-3ftBMb1I`.

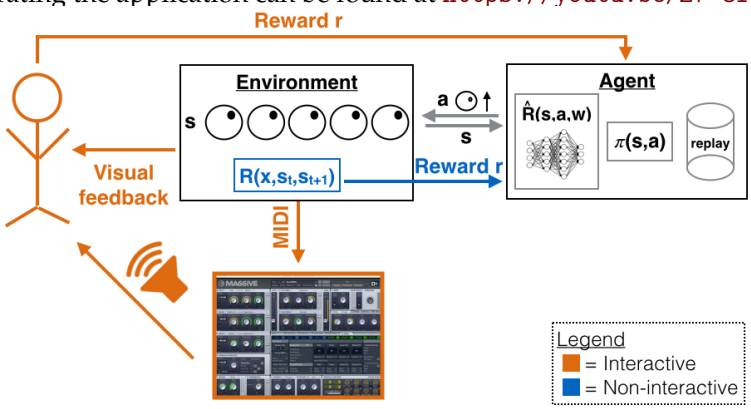


FIGURE 3.1: Interactive and non-interactive deep reinforcement learning model

Representation Every reinforcement learning problem is based on the notion of states, actions and rewards. States for our musical setting are represented using generic parameters for a digital musical instrument or VST. Applied to a frequency modulation synthesizer, parameters can for example control the modulation index, harmonicity ratio, pitch, filter frequency and filter resonance. Actions are directly related to every parameter and can increment or decrement a parameter value. Throughout the interaction, the action size might vary resulting in smooth or sudden state transitions. A sequence of state transitions in our problem setting is called a trajectory. Rewards are provided by a user or synthetic reward function. For the first implementations, I will define a synthetic target state x and a heuristic and synthetic reward function $R(x, s_t, s_{t+1}) = -sign((norm(x, s_{t+1}) - norm(x, s_t))) * 0.5$. The reward function will provide positive or negative rewards depending on whether the agent is approaching or moving away from the target state. The basic model has a state s with five parameters $s = (S1, S2, S3, S4, S5) \in [0, 1]^{|\mathcal{S}|}$ and thus 10 possible actions $A1, A2, ..., A10$ from $\mathcal{A}$ augmenting and decrementing each state dimension with a size of any action $a \in (0, 0.1]$. While the number of states, number of actions and action discretisation might seem small (respectively 5, 10 and 50), the curse of dimensionality makes that the state-action space already is $(1/a)^{|S|} * |\mathcal{A}| = 50^5 * 10 = 3.125$ billion large.

Interaction Feedback provided by a user will be used as rewards (in DSARSA, see equation 2.24) , as labels for an action-value function (in DTAMER, see equation 2.33) or as labels for the advantage function (in COACH, see equation 2.34). Users will listen to state transitions that can have a variable state transition time. User feedback will be stored and incorporated after every transition or after a fixed or variable state trajectory length. Other aspects for the interaction such as pausing, storing or rewinding state exploration could be added as features later on. A basic use-case has 30 minutes of exploration time, state transitions $\in [0.05, 1]$ seconds and a user feedback delay and frequency of respectively 0.2 and $\in [0.2, 1]$ seconds.

Exploration Exploration will be based on ϵ-greedy exploration with ϵ-decay and with the addition of an exploration bonus from section 3.3.2 onwards. The task will be continuing as opposed to episodic. While many algorithms allow to work with both episodic as continuing tasks, the vast majority of recent research focuses on episodic tasks with a slightly outdated literature base focusing on continuing tasks [46]. Furthermore, I will test the ability of the implementation to deal with varying action sizes and random starting states during training.

Reinforcement Learning I will test both value-based as actor-critic and both on- as off-policy algorithms with a discrete action-space. For the off-policy algorithm, a replay memory will be used to tackle the sparse reward challenge and decorrelate training data. An ideal model is on-policy as the user wants to explore and exploit the state space on-line (in real-time). It should be continuing as the user wants to explore the state space with the possibility for infinite trajectories. Continuing spaces would be interesting in a musical setting but is left aside as a requirement for now due to the high complexity and variance in state-of-the-art implementations. A stochastic policy would be good as it allows to set the probability of illegal actions to zero and offers more flexibility to determine an exploration strategy.

Supervised Learning Activation functions of the ANN are Relu for hidden layers and linear (value functions) or Softmax (stochastic policy) for the output layer as modelled for a deep reinforcement learning problem [49] [48]. Loss functions are MSE for a value function and cross-entropy for a stochastic policy as also defined in the literature [73]. ANN inputs are rescaled state values $\in [-0.5, 0.5]$. The ANN has 1,2 or 3 hidden layers with sizes $\in [5, 300]$. An ADAM optimizer [37] is used with learning rates $\in [0.0001, 0.005]$ and mini-batch size $\in (16, 32, 64)$.

Technical specifications All implementation are done using the Python language and with help of the PyCharm IDE[1]. The Tensorflow[2] Python framework was used to support the ANN implementation. All models and training ran on an i7-4578U CPU (double core, 3GHz processor) of a Mac mini computer. User interaction was implemented using Max MSP[3] and communicated to the Python processes over the OSC protocol. Patches in Max MSP or a VST from Native Instruments called Massive[4] were used as a digital instrument to generate sounds to the user. Communication between Max and the VST was done using the MIDI protocol.

3.2 Analysis of Deep Reinforcement Learning

This section presents the analysis of both value-based and an actor-critic algorithm applied to our problem setting. An analysis of several hyperparameters will be presented as well.

3.2.1 Base Models

Three DRL algorithms are implemented in this section and listed below.

Deep SARSA is a value-based, continuing, on-policy RL algorithm with ANN approximation. The update rule for the episodic case can be found in equation 2.18. For a continuing task, the algorithm can be found in appendix C. The update rule was presented in 2.24 and restated here:

$$w_{t+1} = w_t + \alpha \Big[R_{t+1} - \bar{R}_{t+1} + q(S_{t+1}, A_{t+1}, w_t) - q(S_t, A_t, w_t) \Big] \nabla q(S_t, A_t, w_t) \tag{3.1}$$

DDDQN is a value-based, continuing, off-policy RL algorithm with experience replay and ANN approximation. The update rule for the episodic can be found in equation 2.27. For a continuing task, this update rule becomes:

$$\begin{aligned} w_{t+1} = w_t + \alpha \Big[& R_{t+1} - \bar{R}_{t+1} + q(S_{t+1}, argmax_a q(S_{t+1}, a, w'_t), w_t) \\ & - q(S_t, A_t, w_t) \Big] \nabla q(S_t, A_t, w_t) \end{aligned} \tag{3.2}$$

A2C is an actor-critic, continuing RL algorithm with ANN approximation. The update rule for the algorithm can be found in equation 2.25 and is restated here:

$$\theta_{t+1} = \theta_t + \alpha \Big(R_{t+1} - \bar{R}_{t+1} + v(S_{t+1}, w_t) - v(S_t, w) \Big) \frac{\nabla_\theta \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)}$$

I ran the aforementioned algorithms using following parameters obtained with an informal search

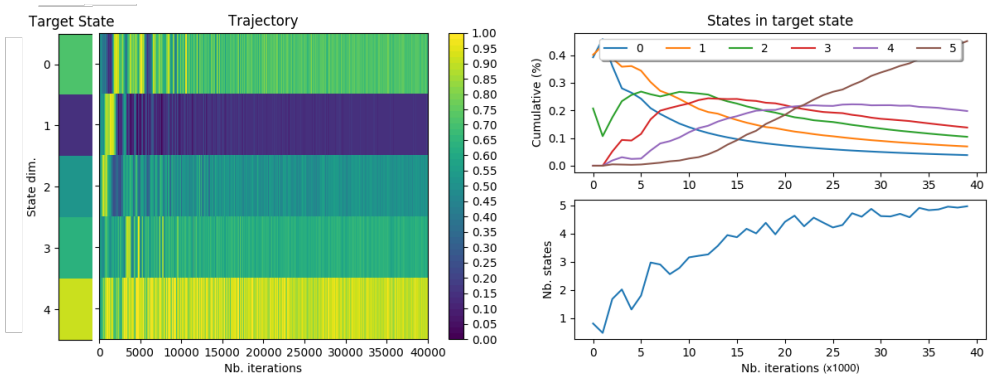
RL model	ANN model
# Iterations = 40,000	# HLs = 2
$\lvert\mathcal{S}\rvert = 5$	$\lvert HL \rvert = 25$
$x = [0.72, 0.15, 0.51, 0.63, 0.91]$	$\alpha = 0.0005$
$a \in [-0.02, 0.02]$	Glorot initialisation [26]
ϵ_d = 20,000	

TABLE 3.1: Basic model parameters

[1] https://www.jetbrains.com/pycharm/
[2] https://www.tensorflow.org/
[3] https://cycling74.com/products/max/
[4] https://www.native-instruments.com/en/products/komplete/synths/massive/

with # HLs and |HL| abbreviations for respectively the number of ANN hidden layers and the ANN hidden layers sizes.

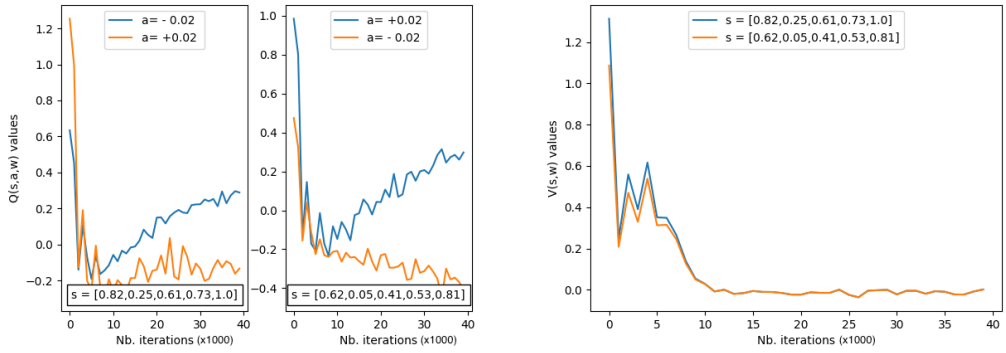
Figure 3.2 present a snapshot of training the different algorithms on the aforementioned problem of finding a target state. Plots for the different algorithms are presented interchangeably as DSARSA, DDDQN and A2C have no problem determining an efficient policy for the basic problem. The basic problem is simple due to the large number of rewards provided, simple reward function and little exploration. As such, the algorithms are heavily overfitting on the training data. These plots are used mainly to illustrate algorithmic differences and interesting characteristics for later implementations. A first selection of figures respectively show the target state, state trajectories and number of state components Si in the target state during training and illustrate a fast convergence towards the target state x.



(A) Target x and DSARSA (B) DDQN

FIGURE 3.2: State trajectories during training

Next, learned Q and V functions averaged over all augmenting and decrementing actions for a given state s are shown in respectively figure 3.3a and 3.3b. It shows how the DSARSA and DDQN algorithms correctly value 'optimal' and 'non-optimal' actions differently ('optimal' meaning action a towards the target state $x = [0.72, 0.15, 0.51, 0.63, 0.91]$). For the A2C algorithm, the critic's network output $V(s, w)$ is plotted and shows how state-value estimates approach zero after 10k iterations. This is due to the fact that the agent has reached the target state and oscillates around it leading to alternating positive and negative rewards. As the state-value function estimates the average reward per time-step, the estimate will move to zero. This then leads to increasingly smaller network updates which can be problematic for learning. Ensuring enough exploration through random action selections using the stochastic policy or continuing a trajectory from a randomly selected starting state could alleviate this issue.

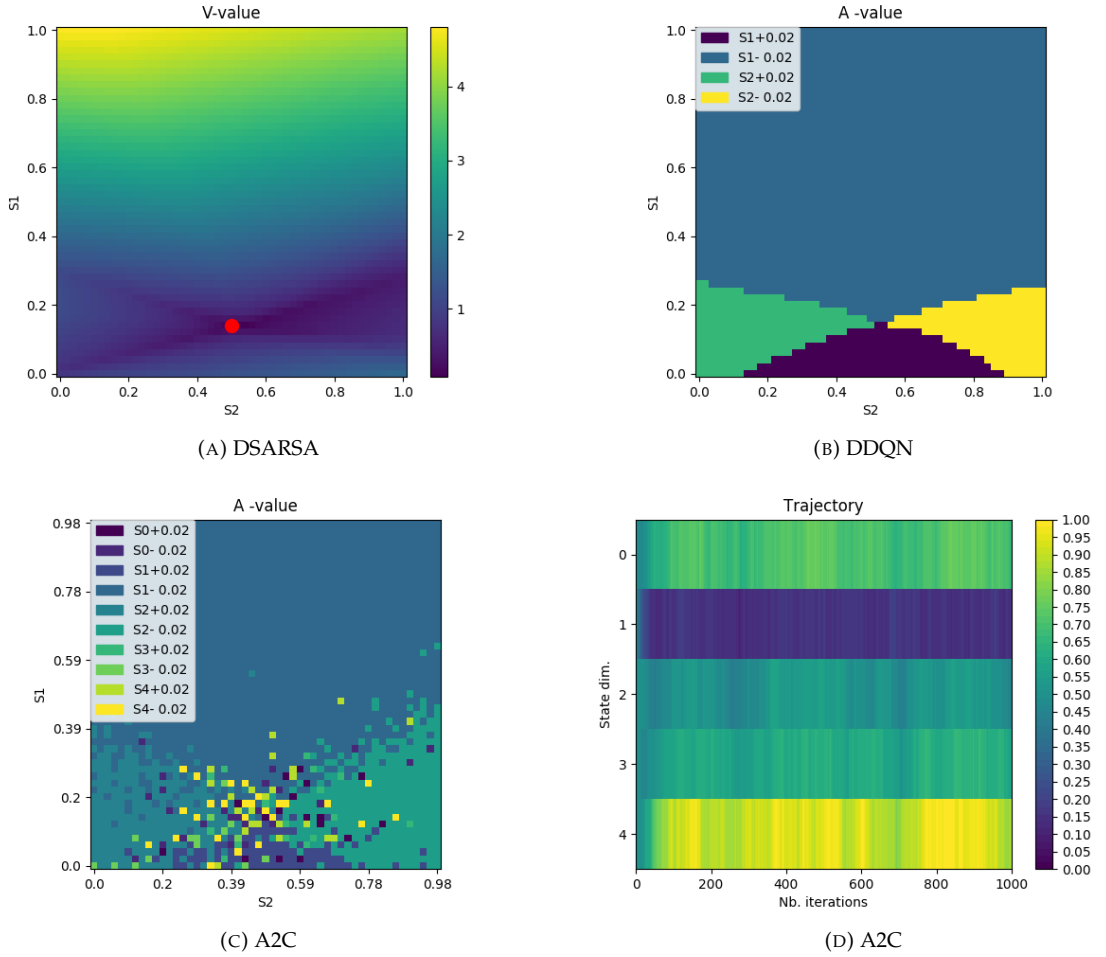


(A) DSARSA (left) and DDQN (right) (B) A2C

FIGURE 3.3: Evolution of state(-action) functions during training

Next, figure 3.4 shows a state-value plot (A), action-value plots displaying actions selected by a greedy and deterministic (B) or stochastic (C) policy and a sampled trajectory (D) to present the networks' learning after training. Axes S1 and S2 respectively refer to state-values of state-dimensions 1 and 2. The target state x with $S1 = 0.15$ and $S2 = 0.5$ is

indicated by the red dot in (A). The target state is predicted to have a minimum state-value as the network learns the estimated return of rewards and once in the target state, rewards will oscillate around zero. One can see for the A2C network in (C) how different actions are chosen around the target state due to the stochastic policy output from the Actor network (multiple actions with probabilities > 0) .



(A) DSARSA

(B) DDQN

(C) A2C

(D) A2C

FIGURE 3.4: DRL networks after training

An important comment can be made on the stability of the results. Due to the randomness in the action selection (the ϵ-greedy exploration) and a large sensitivity to the parameter setting, it remains difficult to obtain consistent results with a constrained training time. The next section provides more details on this matter when analysing the influence of hyperparameters.

Several insights can be gained from these first tests. First, each algorithm has its own strengths and weaknesses. DSARSA offers a straightforward implementation, more stable results due to its simplicity and offers more room for algorithmic improvements as we shall see. DDDQN offers the advantage of being an off-policy method (and thus has room for a separate exploration strategy next to the learned policy) and has the techniques of experience replay and duelling networks to improve sample efficiency as well as stabilise and speed up learning. Finally, A2C as an actor-critic algorithm offers the advantage of directly optimising towards the objective of an optimal policy. With a stochastic policy model, it could offer interesting exploration characteristics for in the interactive case. The critic network however offers little added value with the current reward function but will offer greater potential in the interactive case later on (see the COACH algorithm).

3.2.2 Network Training

To properly set the hyperparameters of the model, I performed a parameter sweep and analysed the algorithms' convergence speed and generalisation. Model parameters are distinguished between parameters of the ANN (number of hidden layers, hidden layer size, learning rate) and parameters of the RL model (training time, ϵ_d, action size). With a target state x in mind, convergence speed is measured using two simple, handcrafted measures such as an Average Target State for iteration t (ATS(t)) and a Time to Target (TT). The ATS can be seen as a moving average of the number of state dimensions or simply put, the number of VST buttons in the target state.

$$ATS(t) = \frac{1}{1000} \sum_{k=t-1000}^{t} \sum_{i}^{|\mathcal{S}|} \begin{cases} s_i(t).i, & \text{if } |x_i - s_i(t)| < 0.05. \\ 0, & \text{otherwise.} \end{cases}$$

$$TT = min(t) \qquad \text{for when } s(t) = x$$

Generalisation is evaluated using the ATS and TT measures for trained models with different starting states. Two runs were completed after training using the trained ANN with same starting state as during training and a starting state with maximum distance from the target state (respectively $s1$ and $s2$ in table 3.3).

With an ϵ-greedy exploration strategy, the added randomness results in variation among training runs. For this reason, I ran the model with same parameters four times and calculated mean and standard deviation on the ATS and TT measures between runs. Runs were evaluated as good if they achieved a high ATS, a low TT with low standard deviation between runs for both during as after training.

Hyperparameters and the associated ranges that were tested are shown in the following table:

RL model	ANN model
# Hidden Layers = [1,2,3]	Training time = [5k, 10k, 20k, 40k]
Hidden layer size = [5,15,25,50,75,100]	ϵ_d / training time = [0.5,0.75, 1, 1.5]
Learning rate = [1e-4, 5e-4, 1e-3]	Action size = [0.04, 0.02, 0.01, 0.005]

TABLE 3.2: Hyperparameter ranges

I ran the above parameters for the DSARSA, DDDQN and A2C model resulting in a total of 3x24x4=288 runs. Note that some runs overlap as I choose the base model from section 3.2.1's parameter setting as the basis for varying other parameters. Also, the A2C model does not have an explicit ϵ parameter and thus had 4x4=16 additional overlapping runs. A selection of training runs is shown in the table below with a parameter setting as for the base model but with ϵ_s equal to 1 and the number of hidden neurons equal to 75. It illustrates how the DSARSA model obtains a best convergence during and after training for both starting states. The results also illustrate the sometimes large variance between training runs. A full overview of the results with the DSARSA algorithm is shown in Appendix D.

	Training (start state $s1$)				After training (start state $s1$ and $s2$)							
	$ATS(s1)$		$TT(s1)$		$ATS(s1)$		$TT(s1)$		$ATS(s2)$		$TT(s2)$	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
DSARSA	4,83	0,13	4276	2400	4,38	0,53	220	283	4,35	0,13	173	25
DDDQN	4,77	0,08	5306	1177	4,14	0,67	286	210	3,68	0,97	435	281
A2C	4,49	0,09	3715	1949	3,34	0,77	181	52	2,95	0,62	428	358

TABLE 3.3: Evaluation of hyperparameters for DSARSA, DDQN and A2C models

Based on all hyperparameter test runs, the following parameter settings performed best for DSARSA, DDQN and Q2C:

	DSARSA	DDQN	A2C
| HL |	[25-75]	[50,100]	[75-100]
# HLs	[1,2]	[1-3]	[2-3]
Learningrate	0.0005	0.001	0.001
ϵ_d / training time	[0.5-1]	1	/
| a |	0.01	0.01	0.04

TABLE 3.4: 'Optimal' model parameters

To analyse training runs I used the earlier introduced ATS and TT measures as well as analysed the evolution of ANN parameters, gradients and network activations using histograms in Tensorboard[5]. Histograms of network parameters in tensorboard showed proper distributions of parameter and update values for an ANN (close to standard normal distribution and evolving over time meaning the network is learning something). I also experimented using various forms of regularisation in the ANN such as L2 regularisation, drop-out and early stopping [51]. However, added regularisation severely degraded results of training runs. Without regularisation, the model is strongly overfitting on the training data. However for a reinforcement learning problem overfitting does not pose a problem as new training data is generated on the fly and sufficient exploration of the parameter space can be guaranteed through a proper parameter setting. Other training techniques such as batch normalisation, error clipping and different ANN parameter initialisations were tested as well but did not improve or degraded performance.

Insights from these runs are:

- High ϵ is good for generalisation (i.e. convergence after training from different starting states) after training while degrades performance during training. Although an obvious insight directly related to the increase degree of randomness in the exploration during training, it is important to keep in mind when setting parameters for user interaction.
- Several runs did not converge towards the target state after training (indicated as X in Appendix D). Causes are insufficient training time, too many hidden layers , too much exploration (as compared to exploitation) or a too large combinatorial state-action space due to a very small action size
- Learning rates can be increased compared to the more standard 1e-4 value found in literature
- Variance between training runs can be high especially for low training time. This observation stresses the need for a good hyperparameter choice to stabilise results between training runs
- Generalisation for DDQN and A2C is poor (1 in 4 'after training' runs reached the target state). I expect this to be due to a fast convergence during training and thus not enough exploration to generalise to non-visited states.
- DSARSA performed best between the three models, DDDQN improved with early stopping to prevent degraded results (I expect due to retraining on its replay memory) and A2C was slow and required a higher number of hidden layers and number of neurons in each layer for better results.
- A high number of rewards is needed before convergence towards target state and generalisation to non-visited states.

With 40k iterations and a consistent reward signal on every iteration, a high number of rewards is the main weakness of models presented so far. This challenge and the user interaction with the agents themselves will be the focus of the next section.

[5] https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard

3.3 Analysis of Interactive Deep Reinforcement Learning

With a better understanding of the influence of model hyperparameters and the strengths and weaknesses of each RL model, this section will focus on the challenges to obtain implementations with user interaction.

3.3.1 Remaining Challenges

Four main weaknesses with the current models are:

1. Variation between training runs
2. Large training time
3. High number of rewards
4. The notion of a target state

The first weakness is due to the randomness associated to the ϵ-greedy exploration and parameter sensitivity. Also, as no theoretical proof exists for convergence towards a global optimum in the case of reinforcement learning with function approximation (such as deep reinforcement learning), different initialisations or the aforementioned randomness can lead to different local optima. This variation between runs will be partly addressed using consistent parameter settings between runs and extending the exploration strategy using the exploration bonus.

Next, the weaknesses of a large training time as well as the high number of rewards required before sufficient learning proved to be most challenging to deal with. First, it is known that reinforcement learning models are very sample inefficient. This mainly because of their bootstrapping updates i.e. using estimates of future rewards based on other estimates as ground-truths for training. Second, as the aim of this research is to develop a model with user interaction, we want the model to learn fast during interaction knowing there is a severe restraint on training time and the number of interactions or sparse feedback. Conform to the reinforcement learning literature, the design of the reward function and its role in the model will prove to be crucial for performance. I will address these challenges with the use of dedicated models for user interaction and an appropriate reward distribution. Rewards will be very simple and guiding the agent in a continuing task as opposed to evaluating the agent in an episodic task. Other techniques to address the sparse rewards challenge are reward shaping (carefully tuning the reward signal using for example higher rewards after a certain sequence of actions [52]), hallucinating rewards (adding rewards based on history or using different reward functions [3]) or the definition of auxiliary tasks [34]. While these might be promising alternative routes, they are left for future work.

Finally, while we certainly want the agent to learn a policy based on the user preferences, it will become more and more difficult to consider the notion of a 'target state'. A target state was introduced to test and evaluate algorithms using a synthetic reward signal. However, with user interaction the focus will shift towards exploration as an objective in itself as a user might have a moving goal or no goal at all. Notions of a target state or convergence speed will so become less of a priority. Fast learning from user preferences obviously remains important but more in function of a proper exploration strategy.

3.3.2 Interactive Models and Evaluation

Integrating user feedback in the models can be done in various ways as explained in section 2.5. Starting from the base models from section 3.2.1, I continued with the DSARSA model as well as adapted the DDQN and A2C models to obtain respectively the interactive DSARSA, DTAMER and COACH models from 2.5.

In addition, I tested several reward distributions over a variable length trajectory. More specific, a reward was positive if the vector between start and final state of a trajectory was in the direction of the target state. This reward was then distributed with linear or exponentially decreasing decrements over earlier states in a trajectory. I also tested a distribution depending on the variance of each state from the vector between start and final state without

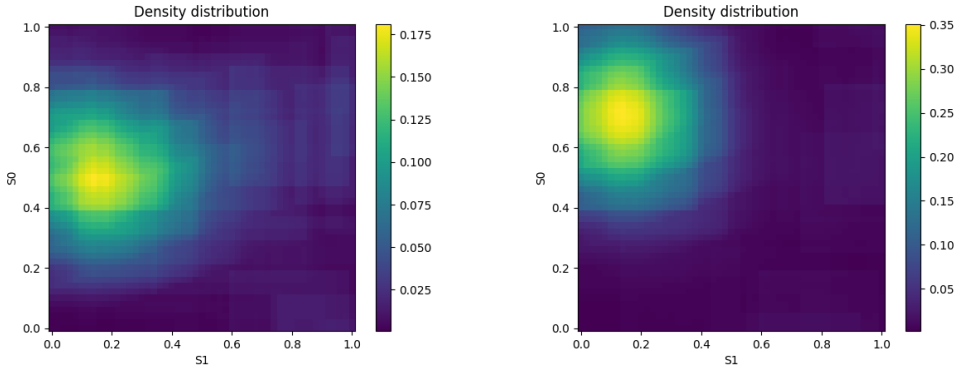
performance improvements. I also implemented the notion of eligibility traces (an exponentially decaying vector keeping track of the policy gradient to deal with delayed rewards [4]) for the COACH model to test the influence of a backward-looking view on action selections but this did not improve or degraded performance.

Summarising the models used to obtain the results presented below:

- interactive DSARSA: user feedback directly replaces the reward function, on-policy
- DTAMER: user feedback as comment on agent's behaviour, off-policy with replay memory
- COACH: policy-dependent user feedback as comment on agent's behaviour, on-policy and feedback as critic in an actor-critic model

To improve the exploration strategy from naive ϵ-greedy exploration, I used the technique of an exploration bonus based on pseudo-counts as formalised in section 2.4. Varying the weight of the exploration bonus allows to reward an agent relative to the novelty of a state. Pseudo-count estimations need a density-model for which I used tile-coding with hashing [79]. Tile-coding has been used extensively in the reinforcement learning literature for feature construction and discretisation of the state-action space. However, to my knowledge, it has not been used for density estimation or in relation with the pseudo-count technique as used here. Tile-coding as density estimation techniques was preferred over other techniques such as Gaussian Mixture Models or using ANNs for its low computational cost and ability to scale to higher dimensions.

The plots below show the density estimations running the DTAMER model after 1000 and 5000 iterations with parameters as in table 3.5 and $\beta = 0.4$ (see equation (2.29)). As the DTAMER agent quickly moves towards the target, state density during training is highly concentrated around the target value ($S0 = 0.72$ and $S1 = 0.15$) with the same synthetic reward signal and target state as before. The plots illustrate how tile coding can be used effectively as a density estimation technique and will be a useful tool to encourage and direct exploration to non-visited states in implementations later on. Precision of every density estimation is calculated dividing the tile size by the number of tilings. Tile-coding was implemented using a hash table to reduce the memory requirements with hashtable size $= \frac{1}{tilesize}^{|\mathcal{S}|}$.#tiles. For a tilesize$=$ 0.4 and the #tiles$=$ 64, the precision equals 0.00625 with a hashtable size$=$ 6250.



(A) Density estimation after 1000 iterations

(B) Density estimation after 5000 iterations

FIGURE 3.5: Density estimation using tile-coding

I ran the three models with several hyperparameter settings using the insights gained from section 3.2.2 and settings from 3.4, with rewards distributed exponentially over trajectories of length 2 to 8 and a number of iterations between 20 and 30,000. A selection of results in shown in table 3.6 for model hyperparameters as in table 3.5. The full training results for the DTAMER algorithm are shown in D.

|HL|	75
# HLs	2
Learning rate	0.001
Training time	20k
ϵ_{start}	0.5
Action size	0.02
Trajectory length	4

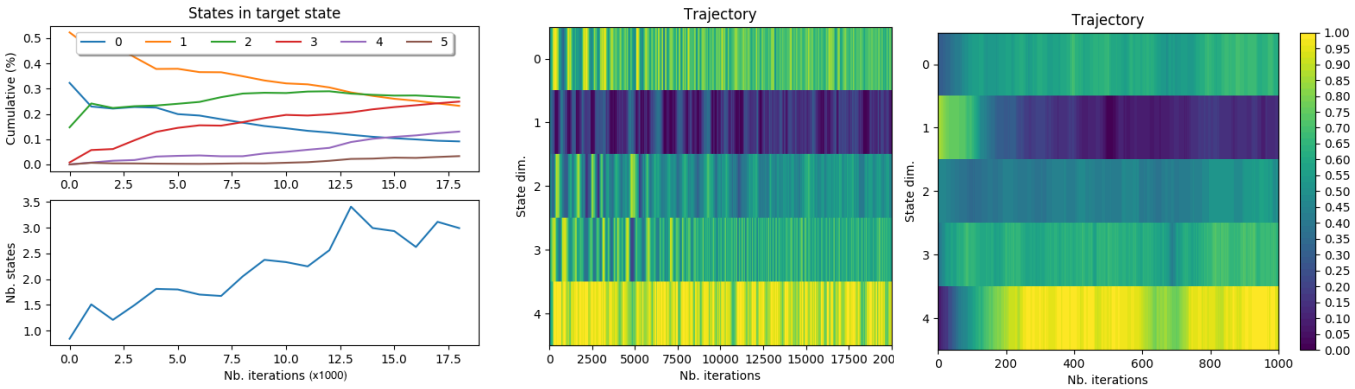
TABLE 3.5: Interactive DRL model parameters

	Training (start state $s1$)		After training (start state $s1$ and $s2$)			
	$ATS(s1)$	$TT(s1)$	$ATS(s1)$	$TT(s1)$	$ATS(s2)$	$TT(s2)$
interactive DSARSA	3,385	1655	1,073	X	2,64	X
DTAMER	4,984	1623	4,817	54	4,34	117
COACH	2,992	1303	2,280	812	1,96	791

TABLE 3.6: Evaluation of Interactive DRL

With fewer rewards and rewards distributed over a trajectory, training becomes significantly harder. As a trajectory includes several state-action pairs, some rewards will be wrongly assigned and not correspond with the objective of reaching a target state. For example, it is possible the agent was moving towards the target state for several steps before taking one action away from the target resulting in a negative reward over the full trajectory. Rewards now correspond to larger trajectories of the agent which could be a beneficial trait for the case with user interaction. I did not attempt to remodel the synthetic reward function to better represent user feedback over longer periods of time (i.e. one or more trajectories) as to directly move towards a case with real user interaction.

However, for the synthetic case as defined in the previous section, DTAMER greatly outperformed the interactive DSARSA and COACH models as shown in table 3.6 (higher ATS and lower TT). 20k iterations with 5k rewards proved to be too little for the interactive DSARSA and COACH agents to properly generalise acquired learning. Only 13 % of their training runs in an informal hyperparameter search generalised learning i.e. obtained a TT value for both starting states $s1$ and $s2$. With 30k iterations, this percentage increased to 40 % which is still very poor. For DTAMER, 85 % of training runs with 20k iterations properly generalised and found the target state from different starting positions. It must be noted that these percentages represent a subset of runs from a hyperparameter search and thus also include the model's parameter sensitivity. Also, while these results may seem very bleak, they only discriminate between generalising or converging and non-generalising models. The latter class of models may still approach the target state but may never get there. This is more often the case for the COACH model due to its stochastic policy and implicit exploration (absence of an exploration parameter as ϵ). As an example of a run approaching but never reaching the target state, the run with results in table 3.6 is shown in figure 3.6.



(A) ATS measure during training

(B) During training

(C) After training with starting state $s2$

FIGURE 3.6: COACH training run

The model analysis done so far remains quite superficial. Other properties such as the algorithms' sensitivity to differential feedback, inconsistent feedback, moving targets, different reward functions and larger state dimension sizes should be further investigated. The algorithms' behaviour and capability to deal with local and global optima, to incorporate or transfer learning between training runs as well as the uncertainty of their value estimates are other important and interesting analyses to be carried out. However, while these may be pertinent from an algorithmic and technical perspective, I argue that it is time to include the user more tightly and that a higher priority in this work now is to focus on the interaction.

3.4 Discussion and Future Work

If one thing should be clear from this chapter, it is that reinforcement learning is hard. The large combinatorial spaces, the different sources of randomness and variation in the algorithms, the slow learning and sample inefficiency, local optima and overfitting all lead to difficulties in training agents. In addition, our problem setting adds the challenging requirements of low training time and sparse rewards.

However, one can obtain satisfying results with the generic problem formalised in section one and with a low state space dimension. First, best performance is achieved using the simplest models. Both DSARSA and DTAMER obtain better results as compared to their peers in our problem setting. Second, it has been shown that sufficient exploration is critical for an algorithm's performance and generalisation. To this end, the exploration bonus obtained using tile coding proved to be very good in terms of simplicity and efficiency. Also, it provides an elegant alternative to a crude ϵ-greedy exploration strategy.

While the analysis done so far has shown convergence to a target state under mild conditions and after training, there remains a question if this convergence is required for a real-life application. When a user is only interested in on-line exploration or exploration during training of the agent, convergence after training becomes less important. This insight provides the opportunity to tune the agent to be more reactive during training (and thus converge faster to a target state) while sacrificing its learning or generalisation to non-visited states.

Another comment can be made on the assumption of a target state and the interest in sounds as such. For a real-life application target states might be moving or non-existing and user interest might be on the sonic evolution or trajectories instead of on sounds as such.

Also, while the models currently have the advantage of being generic in the sense of a neutral parametric state representation, the agent acts only on synthesis parameter values and has no real notion of sound qualities. One could however insert timbral information such as sound descriptors or spectra into the state description to alleviate this issue. Finally, while DTAMER proved to be best in terms of convergence and generalisation, future work could investigate the use of the COACH algorithm to obtain a stochastic instead of deterministic policy. A stochastic policy allows more flexibility in the definition of an exploration strategy and could give interesting (sonic) transitions for actions with equal probability or when the there is high uncertainty in the agent's predictions.

Also, to improve the agent's learning in a real user setting, one could perform a hyperparameter search using real user feedback to train and evaluate the agent. Other ways to improve or change models concern the use of continuous action spaces, inserting a notion of uncertainty in the network so the agent can evaluate its predictions, evaluating the use of n-step methods to improve learning [79] and assure a more structured exploration (see 2.4).

Chapter 4

Design of the User Interaction

We now have an interactive deep reinforcement learning model with flexibility through its generic formalisation and many parameters. From hereon, I will add improvements to tackle the sparse rewards challenge as well as develop new tools for the user to be used during user-agent interaction. These new tools will offer more user control and hopefully a more efficient exploration and enjoyable user experience. While the DTAMER model proved best for our setting, the controls developed in this section can be applied to the other algorithms as well (interactive SARSA, COACH). The outline of this chapter is as follows:

4.1 The Application: This section will present the interface used during the experiments in the next chapter as well as the user-agent interaction process.

4.2 Interaction Controls: Section 4.2 provides an overview of the controls I added to the existing algorithms. They are specific to our problem setting and should improve agent's learning as well as the user experience.

4.3 Additional Features: Here, I present several aspects of the application that I did not analyse in-depth but proved to be important for the experiment in the next chapter. I will present a feature called 'PCA overview', the sound engine and a new version of the application integrating several user remarks obtained during the experiment.

4.1 The Application

The figures in this section present an overview of the final model. It shows a typical application using a VST although the current model is sufficiently generic to apply it to other contexts. One can for example adapt the model to incorporate sound descriptors in the state representation, provide another input modality than raw VST parameters such as for example the acceleration and position of movement data or use the model in a collaborative context such as for example done by Derbinsky in [16].

The current model can be described using two concurrent processes namely a training and an interaction cycle. A typical training cycle is shown in figure 4.1 and can be described as:

1. The environment is currently in state s which it translates to a MIDI signal to the VST.
2. The VST translates this signal through its sound engine into sound towards the user.
3. A user with an actual (most probably non-stationary) reward function $R(s,a)$ in mind provides its feedback R on the sound to the agent.
4. The agent
   (a) uses feedback R to update its estimated reward function $\hat{R}^+(s,a,w)$
   (b) uses the state s to update its density model $\hat{p}(s,\phi)$
   (c) selects a new action a using state s, the output of its learned reward function $\hat{R}^+(s,a,w)$ and its policy $\pi(s)$
   (d) stores the couple (s,a,r^+) in its replay memory for later retraining
   (e) sends action a to the environment
5. Finally, the environment updates it's state s based on received action a and the cycle repeats

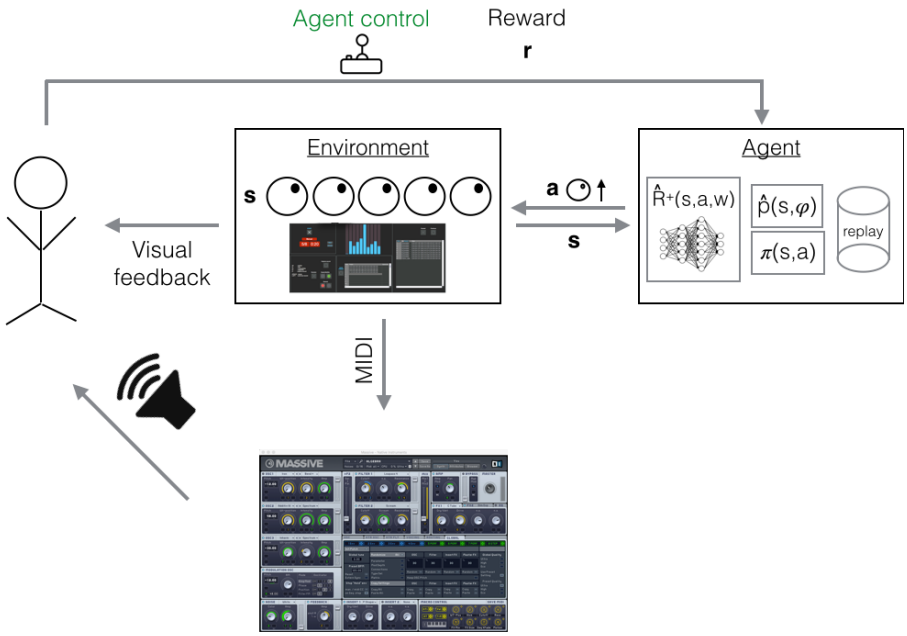


FIGURE 4.1: User-agent-environment architecture

A typical interaction cycle consists of agent control commands from user to agent and returned auditory and visual feedback. The training and interaction cycles can be discrete or continuous reflecting whether the agent will pause and wait for user feedback after taking an action or not.

The user interface is shown in figure 4.2. It shows a counter and timer in red, a preset list to save sounds and a list with exploration history which are all related to the experiment presented in the next chapter.

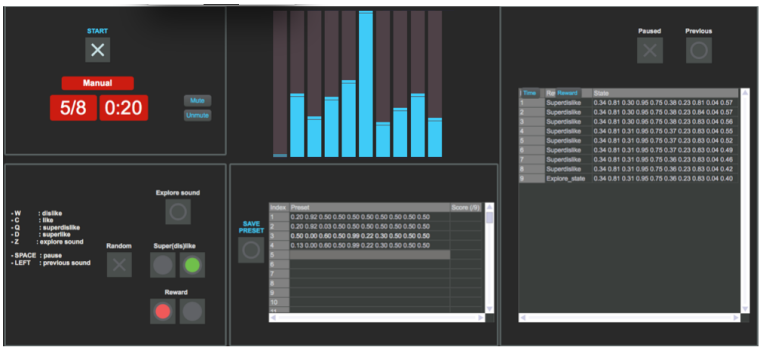


FIGURE 4.2: User interface

4.2 Interaction Controls

Controls are a new addition to the model and are listed in table 4.1. They provide more control to the user and move the application from a 'human-in-the-loop' to a more 'machine-in-the-loop' perspective. They could increase the user experience through more autonomy, flexibility as well as algorithmic qualities and learning through richer user feedback and better exploration strategies. Ideas for these controls resulted from user feedback during experiments with complex musical interfaces [65]. An architectural overview of the agent implementation with user controls is shown in figure 4.3. The figure also shows the internal signal flow within the agent which should be understandable from previous discussions. Agent controls are indicated in green. They are shown directly from user to agent as the environment only acts as an interface to the user and translates user keyboard presses into OSC messages. Agent controls now offered by the environment are:

1. Evaluative Controls	**2. Instructive Controls**	**3. Exploration Controls**
Reward r: Like or dislike state-action pairs	Precision: Increase or decrease state transition speed	Explore state: Sample a new (non-visited) state to start a new trajectory
Super(dis)like: Superlike or superdislike states	Speed: Increase or decrease action size	Explore action: Sample a new (least-taken) action
	(Un)Pause: (Un)Pause the exploration and get an exploration overview	

TABLE 4.1: User interaction controls

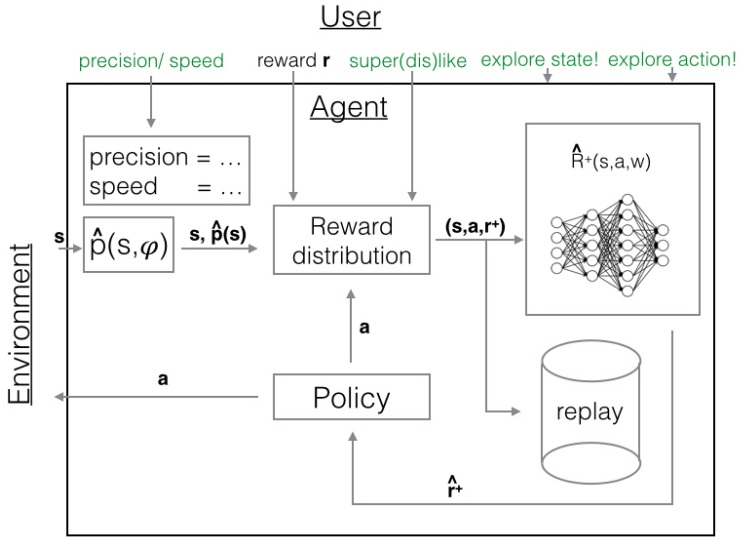
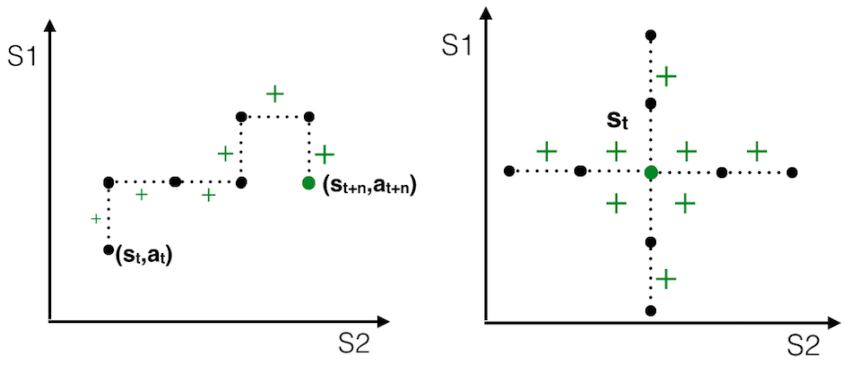


FIGURE 4.3: Agent architecture

The added 'agent controls' open up a new set of affordances in the application. The following paragraphs will discuss their implementation and their impact.

1. Evaluative controls As before, the user can give a reward on the sound transition he or she perceives. Density information is used to calculate an exploration bonus which is added to the reward. This augmented reward is then assigned to state-action pairs and distributed over a trajectory. Apart from a simple reward, the user can also give superlikes or superdislikes to specific states (and in our case, sounds) as such. These superlikes and superdislike translate into rewards distributed to state-action pairs respectively leading towards and away from the concerned state. Figure 4.4 shows a simple reward on the left for state s and action a at time $t + n$ distributed over a trajectory of n states. A superlike shown on the right will impact several state-action pairs leading to the affected state at time t.

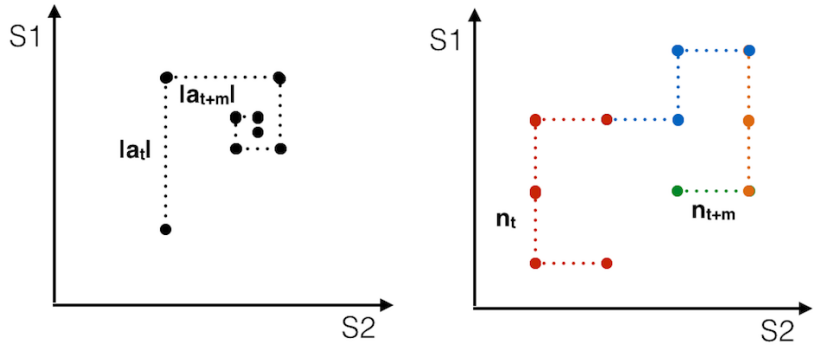


(A) Reward (B) Superlike

FIGURE 4.4: Evaluative controls in a 2-dimensional state space

These two mechanisms each afford the user different controls. Whereas the simple reward comments on the agent's behaviour (the action taken), the super(dis)like evaluates states as such. User feedback can thus be of a guiding or a more evaluative nature which allows for a more flexible user interaction and precise user control.

2. Instructive controls The user can independently modify the precision and the exploration speed of the agent as well as the pause the agent's exploration alltogether. Precision and speed are controlled changing the action size and the state transition time. Changing the transition time also influences the reward distribution length n to ensure the user always evaluates trajectories of 1 second. The effect of both controls are shown in figure 4.5 for an action a and reward distribution length n at time t and $t + m$.



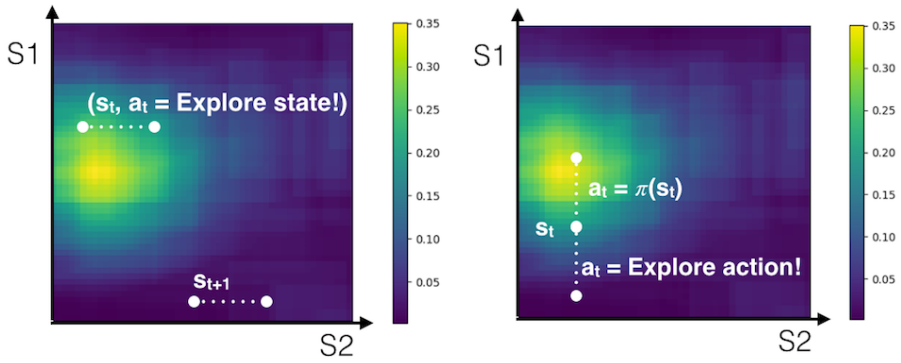
(A) Precision (B) Speed

FIGURE 4.5: Instructive controls

With control over the precision and the speed, the user can for example take finer steps and slow down the exploration. It allows the user to better explore regions of interest or move the agent more quickly out of regions it does not want to explore further.

The user can also pause and unpause the agent's exploration and manually select previously visited states from a list containing the exploration history. I visualised the history in different ways (list, PCA, visualising feedback and history in real-time) as well as allowed the user to take back manual control from the agent.

3. Exploration controls The user has two controls to directly influence the exploration. With the 'explore state' control, the user moves the agent to a new state chosen using random sampling of the estimated state density distribution and selecting the state with the lowest density probability. With 'explore action', the user instructs the agent to take an action to the nearest least-visited state again informed by the density distribution. Both controls are visualised in figure 4.6.



(A) Explore state (B) Explore action

FIGURE 4.6: Exploration controls

Directly influencing the next state or action in an interaction is an important control for the user. As such, he or she can take over or 'restart' the exploration trajectory of the agent using it to escape non-satisfying trajectories chosen by the agent.

Interaction strategies New affordances as explained with the offered controls to the user have important implications for the applications. In this new setting, the user is allowed new interaction or exploration strategies moving beyond simple observation and binary feedback cycles. Two scenarios at the extremes are listed below.

- Scenario 1: A 'passive' user might let the agent explore and only provide simple rewards during the agent's exploration. He or she could gradually decrease the precision and the speed and only rarely use the super(dis)like or explore controls.
- Scenario 2: An 'active' user might first crudely explore the state-space by resetting the agent frequently using the 'explore state' control and the super(dis)like evaluations. He or she could then resort to finer exploration increasing the precision, decreasing the speed and providing simple rewards.

Final implementation A full algorithmic overview with user controls is shown in Algorithm 2 in Appendix C.

4.3 Additional Features

This section describes additional aspects of the model that but proved to be useful for the final application. As such, the PCA interaction was added as a static overview afterwards. The choice of sound mapping proved to be a important factor for the user experience and associated evaluation. I also present a version of the application that was only developed at the end of my internship and

PCA interaction When the training is done i.e. when the user decides to exit the training phase, I also provided an after-training phase. During this 'PCA interaction', the application offers a visualisation with dimensionality reduction using PCA to the user using the state-trajectories explored during training and the user feedback. The user can then re-explore and re-listen to earlier visited states. This new interaction phase is visualised in figure 4.7. While this interaction phase remains quite simple in its current implementation, it offers many opportunities and new routes to improve the application. For example, it requires not much work to restart the training phase using user feedback obtained during the PCA interaction phase. As such, one can alternate between a more agent-exploration oriented and a more user-exploration oriented phase. Another improvement could be the selection of a subspace to restart exploration or to correct earlier provided feedback.

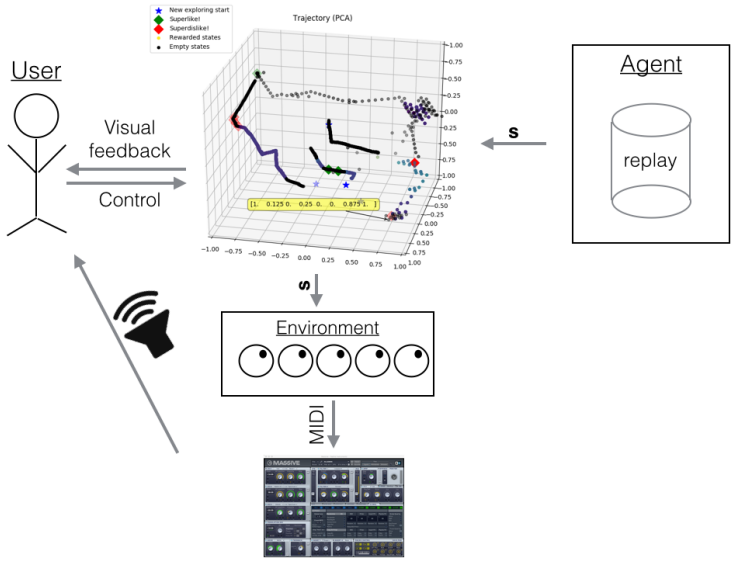


FIGURE 4.7: After training interaction: model

Sound engine For the sound engine and the sound synthesis parameters I used a granular synthesis and FM patch in Max and a VST from Native Instruments called 'Massive' (shown in E). I did not spend much time on choosing a parameter setting and state-sound mapping to focus more on the algorithm and the interaction as presented in earlier sections. The three sound synthesis choices did each have interesting characteristics that could be further explored. For example, state dimensions and parameter settings were distinguishable between agent trajectories for the granular patch while being much less so for the FM and VST synthesis choice due to the non-linear coupling between parameters. The degree of (sound) variation for one parameter, the (non-linear) coupling between parameters or the use of a self-defines sound sample (granular synthesis) all have important consequences for the application and the user interaction. The analysis and experimentation of these is left for future work.

Mixed manual-agent version As mentioned before, the discussion has focused solely on the agent's exploration while a combination with manual exploration is possible. A full implementation with concurrent manual and agent exploration as well as real-time history is shown in figure 4.8. This implementation was developed at the end of my internship and incorporated many of the user remarks presented in the next chapter. Preliminary user tests showed improved potential for (creative) use and this application will also be used in an installation at the upcoming ISMIR 2018 conference in Paris [1].

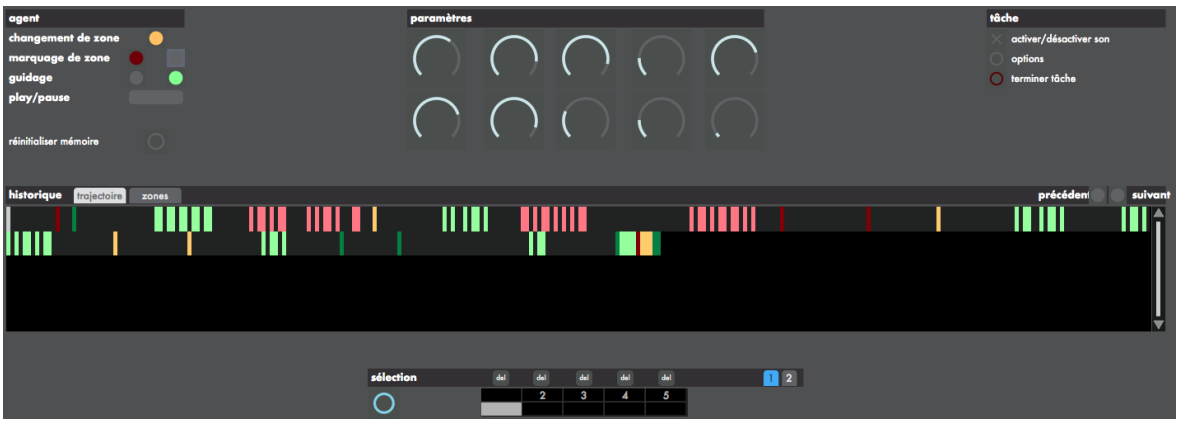


FIGURE 4.8: Mixed manual-agent version of the IDRL

[1] http://ismir2018.ircam.fr/

Chapter 5

Experiment and Evaluation

This chapter evaluates and discusses a final version of the 'interactive deep reinforcement learning agent for timbral exploration'. Its outline is as follows:

5.1 Experimental Set-up and Process

5.2 User Experience and Interaction: This section first presents an analysis of the user experience and the interaction according to feature use for individual users, compared to average use and use over time.

5.3 User Presets and Exploration: This section will focus on the variance between user exploration strategies and link it to user presets and feedback. It successively presents user presets saved during exploration with associated evaluation, the relation between the user preset and feedback distribution and the structure of exploration trajectories.

5.4 Agent's Learning: This section will evaluate the learning of the agent by analysing the user feedback consistency, exploration density and the the behaviour of trained agents in the timbral space.

5.5 Discussion and Future Work

5.1 Experimental Set-up and Process

The goal of the experiment is to analyse the interactive system as presented in earlier chapters in terms of user experience, the human-computer interaction and the trained agent.

Process: For an evaluation of the system, participants were asked to explore the earlier-introduced Massive VST. Participants were asked to explore 10 different parameters for which mapping from each 'macro' parameter to a Massive sound synthesis parameter is indicated in yellow in Appendix E. Exploration was done alternating between three types of interaction as shown in figure 5.1: Manually, through providing feedback to the interactive agent and using the PCA overview. This alternation between phases was chosen to remove bias towards user preference of an interaction type resulting from knowledge of the timbral space.

During exploration, the participants were asked to save an arbitrary number of varied sounds they appreciated. This task was chosen to direct their intentions towards a predefined goal. Variation and appreciation in this context were defined as perceptively different (e.g. bright, dull, harmonic, noisy, etc.) and appreciated according to participants' own subjective criteria (e.g. beautiful, calming, aggressive as positive trait, etc.). At the end of the interaction, participants were asked to score each sound between 0 and 9 to evaluate the resulting sounds from each mode of interaction.

Finally, participants received a questionnaire in which they were asked to evaluate their experience and the interaction itself followed by a brief and informal discussion of their experience. The total duration of the experience was 55 minutes. An overview of the experimental process is shown in figure 5.1. The questionnaire will be discussed in the next section.

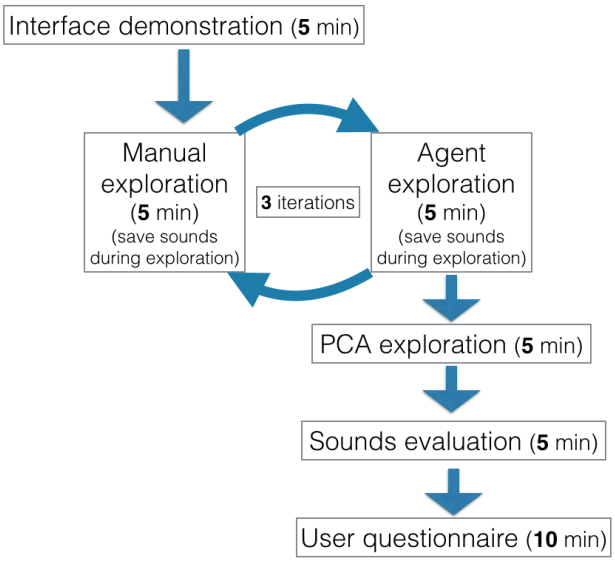


FIGURE 5.1: Experiment process flow

Participants and set-up: In total, 14 participants performed the experiment of which three were female. participant backgrounds comprised sound engineering students, (post-) doctoral researchers in computer music, sound engineers and IT researchers/developers. Ages varied between 22 and 42 years old. In the questionnaire, participants indicated widely varying experience with music production and audio plug-ins from no experience and no use at all to professional experience and daily use. All experiments were executed at my work desk, with my Mac mini computer, AKG K271 headphones and Mac mouse and keyboard.

Reinforcement learning model: During informal tests, it became clear some adjustments needed to be made in the final design of the application obtained in chapter 3. First, I increased the state-space dimension from eight to ten to increase the complexity of the timbral space to be explored. Second, I disabled the instructive controls precision and speed as the application seemed too complex for a participant during a first appropriation of the system. Thirdly, I increased learning rates, decrease the replay memory size and increased the rewards associated with super(dis)likes. These changes were made after first informal tests were participants complained about the reactivity of the agent. While these changes improved the reactivity (increased learning rate gives stronger network updates, decreased replay memory size assures retraining on more recent history and an increase super(dis)like reward also increases its impact on agent's learning), they proved to be detrimental for learning overall as will be shown later on. Participants did the experiment using the interface from figure 4.2. Manual control was disabled during training of the agent and only a simple list overview of the exploration history was shown. An algorithmic overview with parameter settings is shown in Appendix C.

5.2 User Experience and Interaction

After performing all the experiments, one major observation was the diversity between participants in their exploration strategies and their specific needs and frustrations. This first part will present the qualitative results based on the participant questionnaire, user controls frequency of use over time and participant's comments.

Questionnaire: The participant questionnaire can be found in Appendix E. Participants were asked to compare the manual and agent exploration using scores between contrasting adjectives using a Likert scale of 1-5. These adjectives were based on research by Laugwitz who did an in-depth study in how to construct an end-user questionnaire to measure user experience [40]. She identified several adjectives representing criteria in classes such as

perceived ergonomic quality, perceived hedonic quality [1] and perceived attractiveness of a product. The division of classes in criteria and associated adjectives is presented below:

Classes	Criteria	Adjectives
Attractiveness	->	Annoying - enjoyable
Ergonomic quality	Perspicuity	Confusing - clear
		Ambiguous - understandable
	Dependability	Obstructive - supportive
	Efficiency	Inefficient - efficient
Hedonic quality	Novelty	Conventional - inventive
		Dull - creative
	Stimulation	Demotivating - motivating
		Boring - exciting

TABLE 5.1: User experience measures and adjectives

I made one adjective replacement compared to [40] which had 'non-understandable' instead of 'ambiguous' as a counterpart to 'understandable'. My motivation is that I wanted to insert a negative adjective as a positive characteristic for creativity. However, only two out of 14 participants scored the agent very high on ambiguity, the remaining participants' scores were evenly distributed.

I performed a 2-factor ANOVA-analysis with replication using the scores of each participant for all adjectives (see table 5.1) and for both the agent and manual exploration mode. Results with associated scores are summarised as (evaluation results in Appendix G):

Hypothesis and conclusion	F-value	Fcritical	p-value
Hedonic quality: Agent >manual exploration	4.95	3.92	0.028
Ergonomic quality: Manual >agent exploration	22.13	3.92	6e-6
Hedonic adjective average scores are significantly different from each other	3.44	2.68	0.019
Hedonic adjectives and exploration mode are not independent from each other	5.81	2.68	0.001
Attractiveness for both exploration modes was not significantly different	2.23	4.17	0.15

TABLE 5.2: Results ANOVA-analysis user experience questionnaire

In addition to the experience, I asked participants to evaluate the agent controls on their usefulness, usability and satisfaction. The agent tools were in overall considered medium to very useful with an overall average score of 3.88/5 and standard deviation of 0.18. Sound trajectories between the two exploration modes were evaluated by participants as well but were not significantly different. The average results are summarised in the figure below.

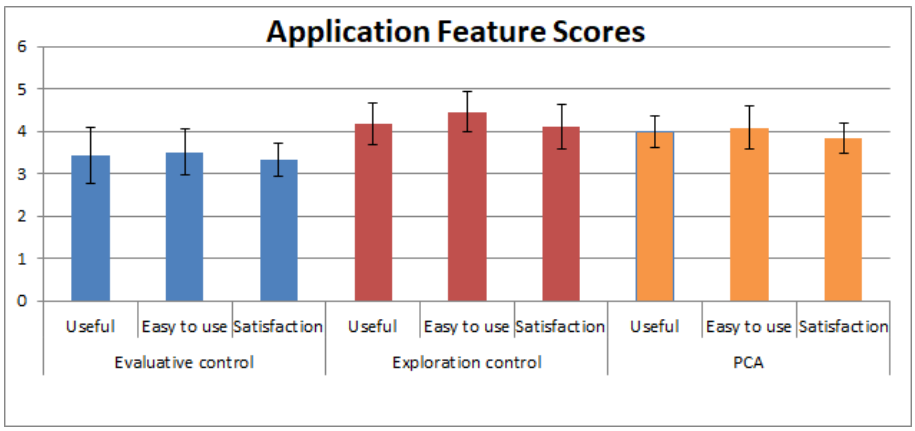


FIGURE 5.2: User evaluation of user controls

[1] Hedonic quality focuses on non-task oriented quality aspects, for example the originality of the design or the beauty of the interface.

User controls and interaction: When analysing the use of agent controls over time and between participants, the variety of participants' exploration strategies becomes clear. The below figures show the percentage and absolute frequency of controls' use. Controls shown are Explore State (ES), SuperLike (SL), SuperDisLike (SDL), Like (L) and DisLike (DL). They show the wide-ranging variety between participants in absolute and relative terms. For example, participant 12 provided a lot of guiding feedback, while participant 8 provided a lot of evaluating feedback as compared to its use of guiding feedback. Another example is the relative large use of the exploring control of participant 1 and 2 and the large variation between absolute counts of all controls. This large variety should be further analysed to adapt the current application to provide an appropriate interface and behaviour of the agent for each participant.

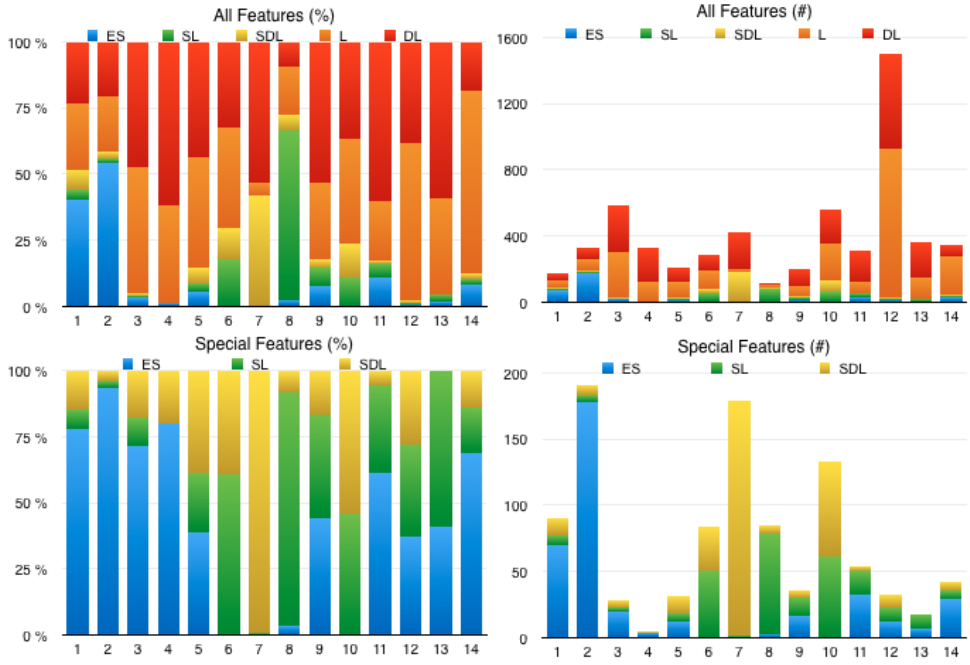


FIGURE 5.3: Frequency of user controls (x-axis: participants, y-axis: relative and absolute count)

Other insights are gained analysing the user interaction over time i.e. for each of the three training runs. The below figures show the use of controls during each phase in absolute and relative terms between participants. Several observations are:

- Participants used less controls over time
- More exploration control in phase 2, then again less in phase 3
- A large decrease and increase of respectively the 'superdislike' and 'explore state' control after the 1st phase (both 31%)
- The most 'superlike' controls are given in the final phase

While the given task will have influenced the results, the observation that participants gave less superdislikes and less feedback overall over time could indicate the agent was performing better or that participants let the agent explore more while providing less feedback.

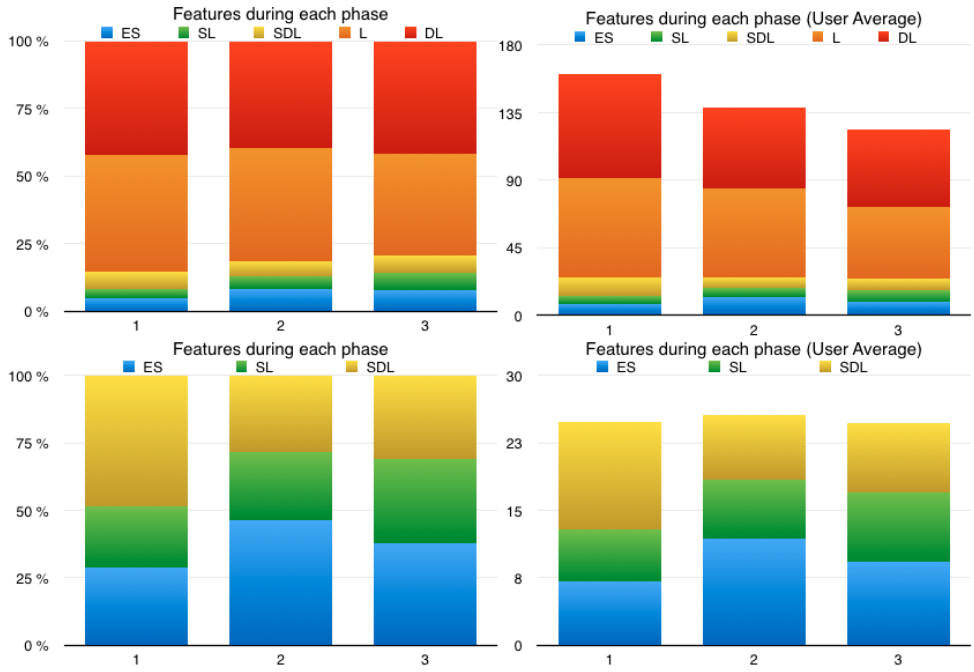


FIGURE 5.4: User controls over time (x-axis: interaction phase, y-axis: relative and absolute count)

Participant comments: Finally, remarks from different participants in the questionnaires and after the experiment also provided several insights.

A first comment can be made on the distinction between trajectories and sounds themselves. While the task was motivating participants to listen to and search for individual sounds, one of the big advantages of using reinforcement learning is the incorporation of transition dynamics. This added value was highlighted by several participants. Participant backgrounds also proved to be important in this respect. More technically oriented or 'analytical' participants such as sound engineers or avid plug-in participants strongly focused on understanding each parameter's role while giving less importance to the exploration dynamics brought about by the agent. Limited task time and confusion about guiding and evaluating feedback was also mentioned as a cause for the trajectory-sound distinction.

A second comment can be made regarding the interaction and exploration itself. Participants appreciated gradually learning more about the agent's 'black-box' and formulated specific notions of exploration. Others found the interaction less direct in the sense that they did not know which parameter the agent was moving. This may suggest that a less direct interaction here refers to the analytical approach, meaning less direct to the sound synthesis parameters. With an agent exploring, participants are motivated to take a more 'embodied' approach, and listen and evaluate sounds themselves without linking parameters with sound synthesis effects.

Another comment is related to the importance of the timbral space and the parameter mapping. The non-linear and one-to-many agent-to-VST parameter mapping proved to be very important for the task. One participant strongly disliked the timbral space and could not appreciate exploring it while others were more interested in the mapping itself than exploring it.

Finally, a large portion of remarks concerned suggestions for improvements. Almost all participants commented on the precision and the speed of the agent and stressed the impact on their questionnaire scores. They had difficulties with the task as the agent was constantly moving (while they did have the possibility to pause the agent) as well as the agent's fixed action size. Consequently, several participants argued to combine the manual with the agent exploration. The PCA was conceived as very useful and enjoyable overall while participants would have appreciated more structure, real-time visualisation, clear labels, the possibility to correct previously provided feedback and to select a PCA subspace to restart training. Selecting parameter subranges or excluding parameters from exploration were also mentioned as improvements.

5.3 User Presets and Exploration

This section will take a look closer to the saved sounds or presets and the exploration of the timbral space during experiments. It will present analyses on the saved sounds and associated user evaluation, the relation between saved sounds (the so-called presets) and user feedback and the various user exploration strategies.

User presets: During each phase of the experiment, namely the three manual and agent exploration phases and the PCA overview phase, participants were asked to store varied sounds they appreciated. At the end of the experiment, the order of saved sounds was randomized and participants were asked to give scores between 0 and 9. The figure below shows some trends on the obtained scores for 12 participants as scores from participant 1 and 4 were lost during saving. It shows how participants saved more sounds during manual exploration as well as valued them slightly higher than during agent exploration. Variance between participants however was high. Sounds saved during the PCA overview obtained the highest scores although were only significantly higher compared to sounds from the first manual interaction (p-values equal to 0.028, 0.072 and 0.143 respectively compared to manual interactions 1, 1+2 and 1+2+3).

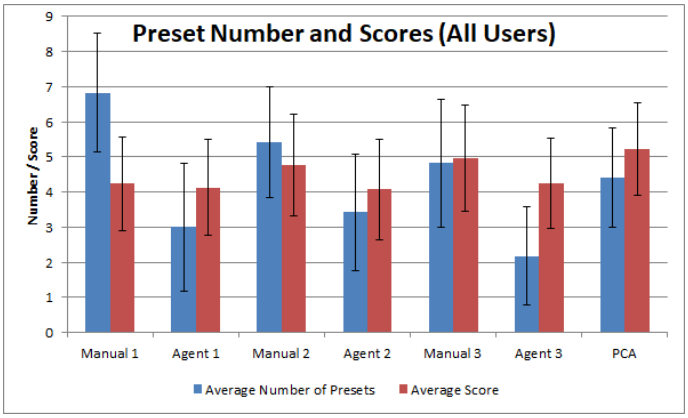


FIGURE 5.5: Saved sounds statistics

User presets and feedback: The following plots show the PCA overview to get an overview of participant's saved sounds and feedback distribution. The PCA projections were done using a uniform PCA giving equal weight to all 10 dimensions to have a coherent projection state spaces between participants. As the task was to obtain varied sounds, figure 5.6 shows participants that had a local and broad exploration respectively shown left and right. One must note, that broad exploration only concern parametric states as no sound perception is taken into account in the PCA projection.

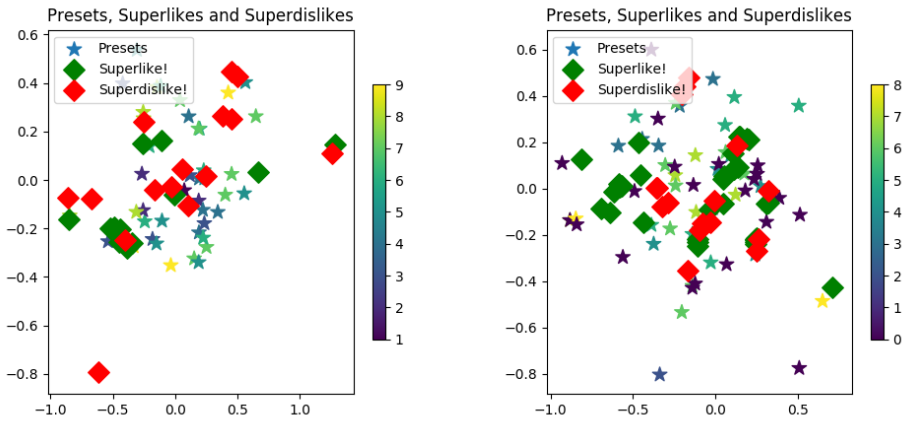


FIGURE 5.6: (Uniform) PCA overview of feedback and presets for participant 1 (left) and 14 (right)

The next figure obtained with a PCA projection with each participant's data, shows how one participant had a consistent strategy to save sounds he or she superliked. He or she also made a clear distinction between superlikes and dislikes which can be interpreted as a consistent feedback strategy beneficial for the agent's learning.

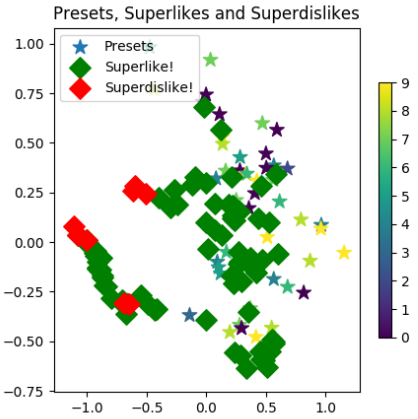


FIGURE 5.7: (Proper) PCA overview of feedback for participant 8

Figure 5.8 shows two PCA projections using all participants' data and shows two participants that had an overlapping region of corresponding feedback. While the distribution of superlikes and dislikes varied widely among participants, these participants seemed to demonstrate a comparable appreciation for sonic regions. They did approach the task differently though as demonstrated by different preset distributions.

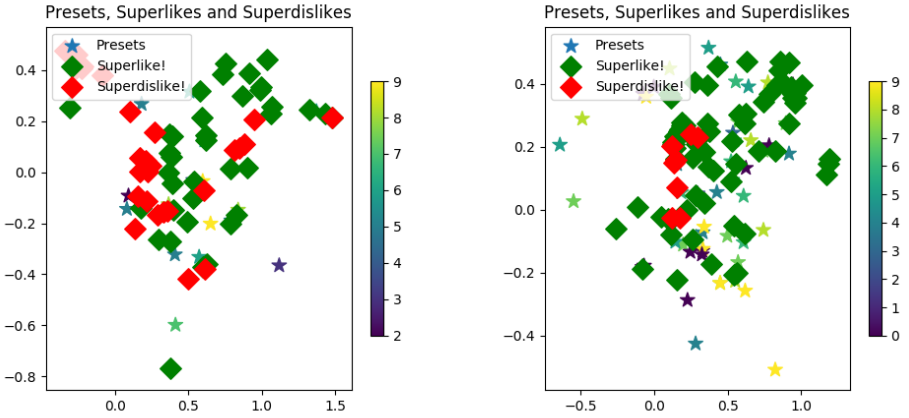
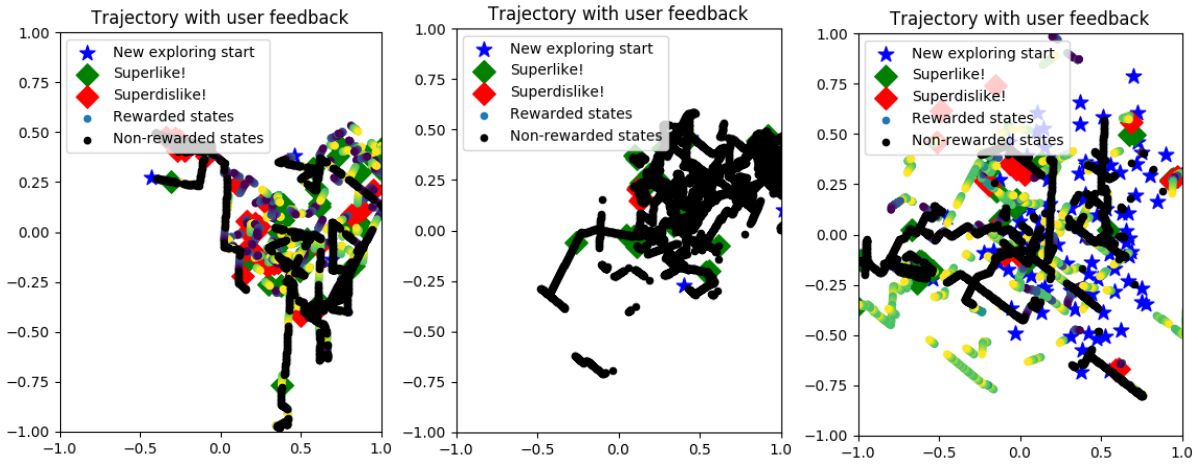


FIGURE 5.8: (Global) PCA overview of feedback for participant 6 (left) and 8 (right)

Exploration strategies: Another interesting illustration of the variation between participant exploration strategies is demonstrated through analysing agent trajectories and state space density weights. I used a uniform PCA matrix (all state dimensions have equal variance weight) and computed a PCA transformation matrix using all data from all participants. I then plotted agent trajectories for each participant over the full 15 minute and individual training phases. The plots below for the computed PCA matrix show how participant 14 used preferred the exploration control over the super(dis)like control as compared to participant 6. The difference in exploration strategies was also shown when plotting both agent and manual trajectories. These plots also suggest the usefulness of visualising agent trajectories with a potential added value for the user during training.

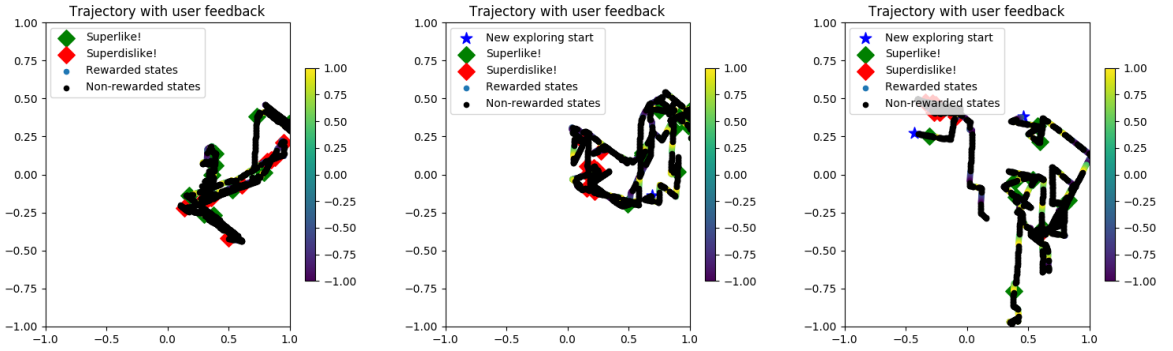


(A) Agent trajectory for participant 6

(B) Agent trajectory for participant 8

(C) Agent trajectory for participant 14

FIGURE 5.9: Agent trajectories for participants 6,8 and 14



(A) Agent trajectory (0-5 minutes)

(B) Agent trajectory (5-10 minutes)

(C) Agent trajectory (10-15 minutes)

FIGURE 5.10: Agent trajectories for participant 6

5.4 Agent's Learning

This section will evaluate the agent's learning by analysing the user feedback consistency, the exploration density distribution and the behaviour of the trained agents.

User feedback consistency: Providing consistent feedback can be challenging for participants, as sounds can be perceptively similar and because the agent is constantly moving. To facilitate proper training of the agent, feedback consistency is important. The following plots show an inconsistent and consistent participant respectively on the left and the right in their use of superlikes and superdislikes. The plots show histograms of the euclidean distance between the state vectors of each superlike with every superdislike. For the user on the left, as the distribution has a portion close to zero, state vectors of several superlike and superdislikes have a near zero distance between them. These plots demonstrate the variability in feedback strategies of participants and stress the importance of having an agent that is flexible over time in the sense that it can adapt to changing feedback strategies.

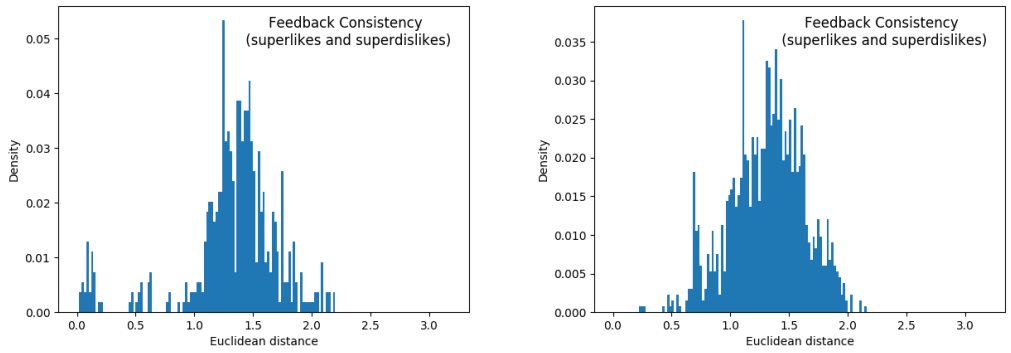


FIGURE 5.11: Histograms of Euclidean distance between superlike and superdislike state-vectors for user 14 (left) and user 10 (right)

Exploration density: I also calculated some statistics over the density distribution of each participant's agent trajectory obtained using aforementioned tile coding. I used a more coarse coding than during training to better illustrate differences between participants with 16 tiles of size 0.7 leading to a tile coding precision of 0.04. These also confirmed the variation between participants and how much of the timbral space they actually explored. The below table and plot shows some statistics and a sorted list of density weights for three participants. A flat curve means the density is more spread over the timbral space meaning the participant explored more. Differences in trajectory length arose because participants paused the agent for a variable amount of time. They show that participant 14 took more pauses but did explore more of the timbral space while participant 8 explored much less. Overall, participants did not pause the exploration for long and several participants did not pause the exploration at all. This may have been due to the fact that it already took some time to appropriate the other controls and because the pause control was not exposed clearly enough. Another reason could be that participants were focusing on the trajectories and did not want to stop the sonic evolution over time.

Participants	|Trajectory|	Min	Max	Mean	Stdev	Skewness	Kurtosis
6	8578	0.000061	545	8.37	31.32	6.69	65.51
8	8410	0.000061	1520	8.21	49.83	12.57	215.96
14	7685	0.000061	286	7.50	21.68	4.17	23.17

TABLE 5.3: Agent trajectories density distribution statistics

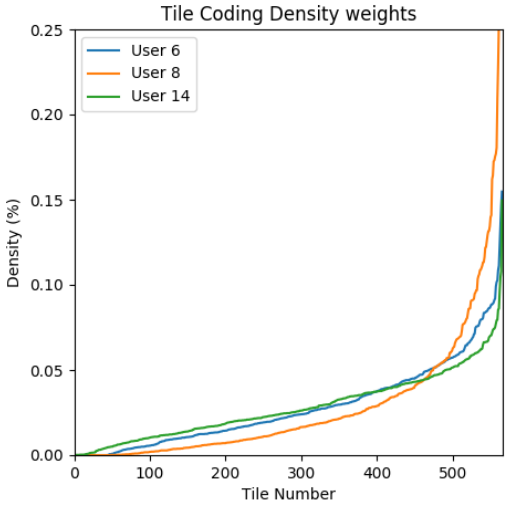


FIGURE 5.12: Density weights for different participants

Agent's learning: Finally, the agent's learning can be analysed by looking at a trained agent's behaviour towards super(dis)likes and (dis)likes. To this end, I collected the trained agents after each user interaction phase as well as created two new agents: one was retrained on each participants' feedback while the other only took random actions. I then had each agent perform 250 trajectories of 600 state transitions with random starting states and calculated the distance between the states with user feedback and each trajectory's start and ending state. A super(dis)like distance represents the relative in- or decrease between a super(dis)like and a trajectory's start and ending state. A (dis)like distance was calculated as the normal distance between the (dis)liked state and the trajectory multiplied by the cosinus angle between (dis)like and trajectory vectors (note that a like is always giving for an agent's action and thus has a direction).

When analysing the data, it became clear this evaluation is not an easy task. First, the large combinatorial state-action space and a limited amount of simulated trajectories causes a large variation between results. Second, the large variation between participants and uneven feedback distribution makes the agent behave differently between participants. Finally, with parameter settings that increased the reactivity of the agent (increased learning rate, decrease replay memory and increased rewards), the agent's learning becomes very unstable. With such strong parameter settings, the agent is severely overfitting on each new input data and the learning can 'jump around' or diverge. Also, it seems the agent now has the tendency to converge to state border values (0 and 1). This may suggest that the overfitting causes the agent to improperly generalise a reward for a state-action pair taking this action all the way up to state borders.

The following figures show the distances of the most recently provided feedback as well as the evolution of the distance over time. The left part of figure 5.13 shows how the agent did learn to move away from superdislikes (distribution centred on negative distance differences so end state further removed from SDL than start state), move corresponding to likes and dislikes (rewards in the figure) and had difficulties with moving towards superlikes. Figure B does show a distribution centred with positive distances but this was not the case for all participants. The main cause was that agents did learn to move towards superlikes but than moved past it. This behaviour is illustrated by figure 5.14 showing how distances between superlikes first decrease and then increase. It also illustrate the difficulty in acting optimally when lots of feedback is provided. For this reason, table 5.4 shows mean as well as the maximum and minimum distances for respectively superlikes (SL_min) and superdislikes (SDL_max). The 'X's in the table are shown when that type of feedback was not among participants 15 most recent feedback. The table shows how the mean distance change values for superlikes are only positive for two participants while they are positive for all participants when taking the maxima values over all trajectories. For now, I did not manage to get improved behaviour using retraining of the agents. The random agent only moved in small local oscillations with very small average distance values centered around zero.

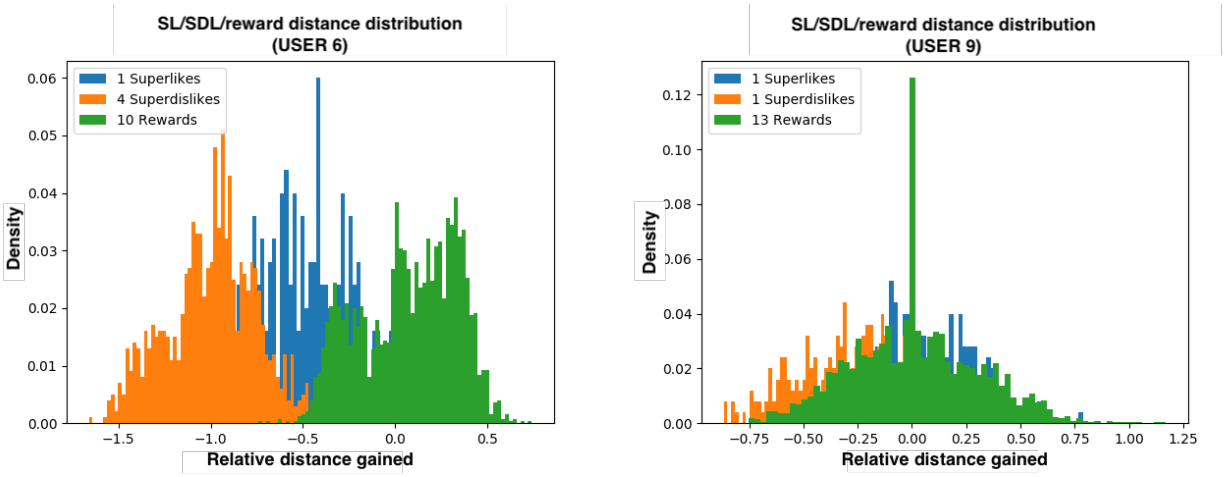


FIGURE 5.13: Participant feedback and trajectory distances

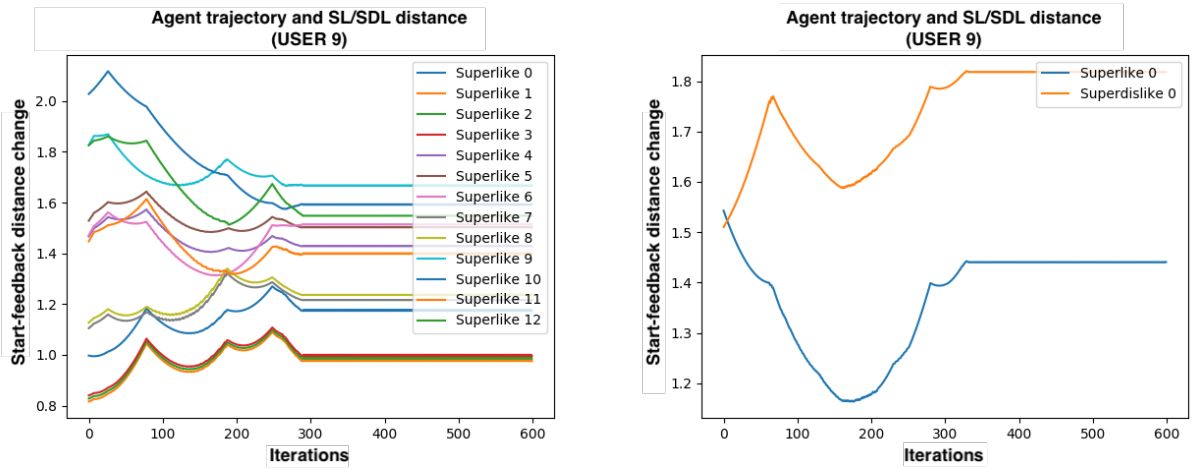


FIGURE 5.14: Participant feedback and trajectory distances

Participants	SL	SDL	R	SL_min	SDL_max	R_min
1	X	-0.332	0.002	X	-0.418	0.199
2	X	X	-0.031	X	X	0.162
3	0.287	X	-0.068	0.399	X	0.340
4	X	X	0.042	X	X	0.326
5	-0.493	-0.996	0.072	0.071	-1.002	0.080
6	X	-0.666	0.098	X	-0.756	0.076
7	X	-0.540	-0.115	X	-0.558	0.086
8	-0.136	X	-0.002	0.106	X	0.165
9	0.018	-0.311	0.021	0.338	-0.354	0.125
10	X	-0.950	0.090	X	-0.960	0.037
11	-0.282	X	0.093	0.087	X	0.277
12	-0.197	X	0.229	0.022	X	0.184
12	X	X	0.128	X	X	0.258
13	-0.065	-0.150	-0.032	0.184	-0.203	0.109

TABLE 5.4: User trained agent mean distance differences between trajectories' start-end state and feedback
(250 trajectories of length 600 with random starting states

5.5 Discussion and Future Work

The main contribution of this chapter is to demonstrate the variance of user exploration strategies. The evaluation also shows how exploration using an agent has qualities different from and complementary to manual exploration. While manual exploration proved to be more ergonomic, the agent's exploration had better hedonic qualities. Added user controls such as the explore and PCA control also proved to be highly useful, usable and satisfactory. With such different user strategies and attitudes as also shown during informal interviews, exploration with an agent has shown to accommodate these wide range of users to a certain extent and to stimulate many opportunities and motivations for improvement.

Unfortunately, results are not conclusive about the agent's learning. This is not too suprising as the state dimensionality of the problem was increased during the experiments, new controls were not taken into account during the model analysis and models were not adapted to the various user feedback strategies. As such, sounds saved during manual exploration remain more appreciated by users notwithstanding that the agent's exploration offered new outcomes such as sound trajectories. Re-exploring the timbral space with trained and retrained agent's as presented in the final part of this chapter did not show agents properly oscillating around or diverging from respectively superlikes and superdislikes. While potential causes have been presented such as the variability between users using the PCA

overviews, the user feedback inconsistency and the exploration density, I believe parameter tuning due to the agent's reactivity and agent's learning trade-off and the theoretical and experimental analysis discrepancy to be more important.

Using these insights, the primary directions for improvement concern the user interaction and the algorithm's performance itself.

For the user interaction, further development of the mixed manual-agent version from section 4.3 could be a good first step. Agent speed and precision control were already presented but were left out during experiments to simplify the interface. To improve the application, a way to integrate these new affordances while not complicating the interface needs to be found. Another control could be the possibility to select state ranges or exclude certain state dimensions for the agent to explore. Finally, while the PCA was proven to be beneficial for the exploration, one can better leverage this technique by displaying the overview in real-time or to use a different data projection to better structure the state space overview. While these improvements all concern specific features or a way to better integrate these features, other interaction modes could be interesting as well. For example, instead of using a mouse-keyboard-screen setting, one can also think of an interface using different controllers such as for example smartphones, using movement input or visualising sound qualities such as for example brightness instead of parameter values and states.

For the algorithm's performance, a next step would be to construct a database of user feedback to improve the current model for better performance in real-life situations. The latter could be done using a hyperparameter search, analysis of the impact of user controls on learning as well as changing the problem formalisation itself such as state representation, border effects (state value maxima and minima) and state space dimension size.

Finally, one caveat in this research remains the notion of sound trajectories versus sound themselves. While reinforcement learning in its core is based on transition dynamics, in my research and its evaluation, I only focused on static sounds or timbres. As such, one big and presumably exciting challenge would be to leverage and investigate the agent's trajectories during exploration. One could also include other musical features such as temporal envelopes and rhythms to move closer to a real musical exploration. One other exciting application could be to use a pre-organised timbral space obtained using for example a Variational Auto-Encoder for the agent to explore [18]. While navigating such a space can be challenging by hand as state dimensions are often difficult to interpret, an agent could be a good partner for such a setting.

Chapter 6

Conclusion

The research presented in this report investigated the use and added value of using an intelligent partner for the exploration of a musical space. While the use of and interaction with such an agent varied widely between users, the presented model has shown capable of accommodating these varied uses and of stimulating new ideas for improvement. It has also demonstrated the added value of using such an intelligent agent and complementary place next to a standard manual exploration. We believe having provided good starting points towards both the computer-centred basis as human-centred aspects of the problem.

Specifically, simple (interactive) deep reinforcement learning models such as DSARSA and DTAMER proved to be best suited for the problem setting formulated here. Models did prove to be highly sensible to parameter settings but proper settings were found using an extensive hyperparameter search with synthetic goals and reward functions. These models have shown to be able to handle the sparse reward challenge and limited training time under the condition of sufficient exploration. The latter was achieved using an exploration bonus based on earlier research and implemented with a novel use of the tile coding technique for density estimation. Nevertheless, evaluation of the user-trained agents' learning did not show properly generalising agents as opposed to the synthetic case. The main causes seemed to be due to the variation in exploration strategies between users, their feedback inconsistency and parameter settings beneficial for agent's reactivity but detrimental for agent's learning.

This research also proposed several new ways to interact with these interactive models to stimulate and structure exploration as well as offer a better user experience. Exploration was stimulated with controls informed by a density estimation and with direct manual intervention in the exploration. Manual intervention by the user was done through changing state values or interacting with a static PCA overview or exploration history list. These new controls improved the user experience through the appearance of new affordances and the possibility to take control of the exploration from the agent by the user. Using an agent for exploration and with the possibility to intervene also allowed new interaction strategies and a more embodied interaction as users could let the agent explore and focus on listening to the exploration result.

A user evaluation of the application also showed how exploration using an agent had better hedonic qualities as opposed to manual exploration which had higher ergonomic qualities. Controls were deemed useful, usable and attrative by users with a strong preference for the exploration control also based on tile coding. Other controls left out during the evaluation such as speed and precision were nevertheless missed during interviews with users. Other improvements suggested by users were a more structured PCA overview, a real-time exploration visualisation, the possibility to select subspaces for exploration and the possibility to record whole trajectories instead of only sounds as such. The latter also represents a direction for future work as while this research concentrated on sounds on its own, state transition dynamics are at the heart of reinforcement learning but were not included in the analysis so far.

Appendix A

Policy Improvement Theorem and Optimality

Theorem: *Let π and π' be any pair of deterministic policies such that, for all $s \in \mathcal{S}$,*

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{A.1}$$

Then the policy π' must be as good as, or better than, π. That is, it must obtain greater or equal expected return from all states $s \in \mathcal{S}$:

$$v_{\pi'}(s) \geq v_\pi(s). \tag{A.2}$$

Proof: Starting from A.1, we keep expanding the q_π side with

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \tag{A.3}$$

and reapplying A.1 until we get $v_{\pi'}(s)$:

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
&= \mathop{\mathbb{E}}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&\leq \mathop{\mathbb{E}}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
&= \mathop{\mathbb{E}}_{\pi'}[R_{t+1} + \gamma \mathop{\mathbb{E}}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})] | S_t = s] \\
&= \mathop{\mathbb{E}}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s] \\
&\leq \mathop{\mathbb{E}}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) | S_t = s] \\
&\vdots \\
&\leq \mathop{\mathbb{E}}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... | S_t = s] \\
&= v_{\pi'}(s).
\end{aligned} \tag{A.4}$$

□

Policy improvement: One can extend this idea of a change in the policy at a single state and particular action to all states and all possible actions. As such, one obtains the greedy policy π' that takes the action that looks best after one step of lookahead. By construction, this policy meets the condition of the policy improvement theorem A.1 and is thus as good as, or better than, the original policy. This process of making a new policy improve on an original policy, by making it greedy with respect to the value function of the original policy, is called policy improvement.

Optimal policy: Suppose the new greedy policy π' is as good as but not better then the old policy π. Then $v_\pi = v_{\pi'}$, and with

$$\pi'(s) \doteq argmax_a \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big] \tag{A.5}$$

if follows that for all $s \in \mathcal{S}$:

$$v_{\pi'}(s) = max_a \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big] \tag{A.6}$$

This last equation is the same as the Bellman optimality equation presented earlier in the text (equation 2.9) and therefore, $v_{\pi'}$ must be v_*, and both π and π' must be optimal policies. Policy improvement thus must give us a strictly better policy except when the original policy is already optimal.

Appendix B

Policy Gradient Theorem

Theorem: *Let J be the performance measure as the value of the start state (episodic tasks) or the average rate of return over time (continuing tasks):*

$$J(\theta) = v_{\pi_\theta}(s_0). \tag{B.1}$$

Then the following is true for the gradient of J

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_\theta \pi(a|s,\theta) \tag{B.2}$$

with $\mu(s)$ the on-policy state distribution under π. In the episodic case, the constant of proportionality is the average length of an episode, and in the continuing case it is 1.

Proof: The proof will only be derived for the episodic case. For the continuing case, the reader is referred to [73]. To keep the notation simple, we leave it implicit in all cases that π is a function of θ, and all gradients are implicit with respect to θ. First note that the gradient of the state-value function can be written in terms of the action-value function as

$$\begin{aligned}
\nabla v_\pi(s) &= \nabla \left[\sum_a \pi(a|s) q_\pi(s,a) \right], \text{ for all } s \in \mathcal{S} \\
&= \sum_a \left[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a) \right] \\
&= \sum_a \left[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla \sum_{s',r} p(s',r|s,a)(r + v_\pi(s')) \right] \\
&= \sum_a \left[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla v_\pi(s') \right] \\
&= \sum_a \Bigg[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \\
&\qquad \sum_{a'} \left[\nabla \pi(a'|s') q_(s',a') + \pi(a'|s') \sum_{s''} p(s''|s',a') \nabla v_\pi(s'') \right] \\
&= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} Pr(s \to x, k, \pi) \sum_a \nabla \pi(a|x) q_\pi(x,a),
\end{aligned} \tag{B.3}$$

after repeated unrolling, where $Pr(s \to x, k, \pi)$ is the probability of transitioning from state s to state x in k steps under policy π. With $\eta(s)$ the number of time steps spent, on average,

in state s in a single episode, and $\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}$, it is then immediate that

$$\begin{aligned}
\nabla J(\theta) &= \nabla v_\pi(s_0) \\
&= \sum_s \left(\sum_{k=0}^{\infty} Pr(s_o \to s, k, \pi \right) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_s \eta(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a).
\end{aligned} \tag{B.4}$$

□

This expression can then be converted to a sample-based expectation through

$$\begin{aligned}
\nabla J(\theta) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla_\theta \pi(a|s,\theta) \\
&= \mathop{\mathbb{E}}_{\pi} \left[\sum_a \pi(a|S_t,\theta) q_\pi(S_t,a) \frac{\nabla_\theta \pi(a|S_t,\theta)}{\pi(a|S_t,\theta)} \right] \\
&= \mathop{\mathbb{E}}_{\pi} \left[q_\pi(S_t,A_t) \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \right] \\
&= \mathop{\mathbb{E}}_{\pi} \left[G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \right].
\end{aligned} \tag{B.5}$$

Appendix C

Algorithms and Parameter Settings

Algorithm 1: Continuing semi-gradient DSARSA for estimating $\hat{q} \approx q_\pi$ (Adapted from [73] to include ANN approximation and continuing task)

Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \Re^d \rightarrow \Re$, a policy π;
Initialise: value-function weights $w \in \Re^d$ arbitrarily (e.g., $w = 0$);
Initialise: average-reward estimate $\hat{R} \in \Re$ arbitrarily (e.g., $\hat{R} = 0$) ;
Initialise: S_0 and A_0 ;
Algorithm parameters: step size α, an average-reward constant C, a small $\epsilon > 0$;
while $t < t_{max}$ **do**
  Take action A_t ;
  Observe and store the next reward as R_{t+1} and the next state as S_{t+1} ;
  Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$, or ϵ-greedy wrt $\hat{q}(S_{t+1}, \cdot, w)$;
  $\delta \leftarrow (R_{t+1} - \hat{R}) + \hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w)$;
  $\hat{R} \leftarrow \hat{R} + C\delta$;
  $w \leftarrow w + \alpha\delta\nabla\hat{q}(S_t, A_t, w)$;
  $S_t \leftarrow S_{t+1}$;
  $A_t \leftarrow A_{t+1}$;

Algorithm 2 presents my version of the DTAMER algorithm adapted to include the use of exploration bonus and to allow several user controls. All user feedback is assigned to states with a delay of 0.2 seconds. The reward distribution function $Env_aist()$ is a simple exponentially decaying function. Specific parameter settings as used during the experiment are listed below. The replaymemory size was chosen not too large to assure learning from recent experience while the C constant is taken from literature. The Rlength parameter refers to the reward distribution length. As such, rewards are distributed over trajectories of 1 second. The algorithm also shows three separate training cycles to assure direct training on feedback, training on experience after a pre-defined time ($2 * batchsize$) and training solely on the exploration bonus. The latter assures the agent focuses on exploring when no feedback is provided. User controls are initialised as speed $= 0.1s$, precision$= 0.01$, Superlike$= 1$, Explore_state$= 0$, Explore_action$= 0$

ANN	DRL	Exploration	Density $\hat{p}()$
$\|HL\| = 100$	$\mathcal{S} = 10$	$\epsilon_d = 2000$	# tiles $= 64$
$\#HLs = 2$	$s \in [0, 1]$	$\epsilon_s = 0.1$	tilesize $= 0.4$
batchsize $= 32$	$\|a\| = 0.01$	$\epsilon_e = 0.0$	$C = 0.01$
learningrate $\alpha = 0.002$	$\|R\| = 1$		
$\|replaymemory\mathcal{D} = 700$	Rlength $= 10$		

TABLE C.1: DTAMER experiment parameter settings

Algorithm 2: Continuing DTAMER with exporation bonus and user controls for estimating $\hat{R}() \approx R()$

Input: differentiable function $\hat{R}(s, a, w)$, policy $\pi()$ as $\epsilon-$greedy with exponential ϵ-decay, reward distribution function $Env\_dist()$;

Initialise: weights $w = 0 \in \Re^d$, average-reward $\hat{R} = 0$, $S_0 = 0.5 \in \Re^{|\mathcal{S}|}$and $A_0 = \pi(S_0)$, $x = 0 \in \mathcal{N}^{Rlength}$;

while *running* **do**

- Take action A_t and observe next state S_{t+1} ;
- Select new action $A_{t+1} \sim \pi(\cdot|S_{t+1})$;
- Store $(S_{t+1}, A_{t+1}, 0)$ in reward length vector x (R_{t+1} stored as 0);
- Update density model $\hat{p}()$;
- Observe reward as R_{t+1} ;
- $S_t \leftarrow S_{t+1}$;
- $A_t \leftarrow A_{t+1}$;
- **if** $R \neq 0$ **and** $t > Rlength$ **then** `// Train feedback+exploration bonus`
  - $x = Env\_dist(R)$;
  - Store x in $\mathcal{D}$;
  - Compute $\hat{R}_{t+1}$ using SGD (eq. 2.33) and x ;
- **else if** $|\mathcal{D}| > 2 * batchsize$ **then** `// Train on experience`
  - $\mathcal{D}_{t+1}$ = random sample from $\mathcal{D}$;
  - Compute $\hat{R}_{t+1}$ using SGD (eq. 2.33) and $\mathcal{D}_{t+1}$;
- **else if** $t > Rlength$ **then** `// Train on exploration bonus`
  - Compute $\hat{R}_{t+1}$ using SGD (eq. 2.33) and R^+ ;
- **while** *Paused* **do**
  - agent.get_currentstate() ;
- **if** *Explore_state* **then**
  - **for** *i in range(20)* **do**
    - Randomly sample state s_i ;
    - Evaluate predictiongain$(s_i) = log(\hat{p}_{t+1}(s_i)) - log(\hat{p}_t(s_i))$;
  - $S_t = argmax(predictiongain(s_i))$;
- **if** *Explore_action* **then**
  - **for** *i in range($|S| * 2$)* **do**
    - Take action A_i and observe next state s_i ;
    - Evaluate predictiongain$(s_i) = log(\hat{p}_{t+1}(s_i)) - log(\hat{p}_t(s_i))$;
  - $S_t = argmax(predictiongain(s_i))$;
- **if** *Super(dis)like* **then**
  - $S_{00} = x[0]$ and $A_{00} = Super(dis)like$;
  - **for** *i in range(Rlength)* **do**
    - **for** *j in range($|S|$)* **do**
      - Take action A_{ij} and observe state S_{ij} ;
      - Store S_{ij} in $\mathcal{D}_{t+1}$;
  - Compute $\hat{R}_{t+1}$ using SGD (eq. 2.33) and $\mathcal{D}_{t+1}$;
  - $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{t+1}$;
- **if** *New_precision* **then**
  - $|a| \leftarrow |a| / new\_precision$;
- **if** *New_speed* **then**
  - $speed \leftarrow new\_speed$;

Appendix D

Hyperparameter Training Results

Below the training results using the DSARSA algorithm using four runs for each parameter setting. The X indicate the number of runs not reaching the target state.

		During Training				After training							
		ATS(S1)		TT(S1)		ATS(S1)		TT(S1)		ATS(S2)		TT(S2)	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
|HL|	5	2,404	0,257	10861	6250	0,594	0,600		XXXX	0,395	0,581		XXXX
	15	4,948	0,076	4000	836	4,616	0,243	219	246	3,474	0,257	411	73
	25	4,980	0,021	2800	1086	4,425	0,307	317	196	4,362	0,343	175	76
	50	4,852	0,187	2623	529	3,952	1,209	374	387	4,207	0,317	171	74
	75	4,845	0,108	4091	1569	4,738	0,174	60	10	4,231	0,404	151	27
	100	4,790	0,283	3704	1840	4,688	0,225	123	129	4,226	0,265	154	57
# Hls	1	4,979	0,032	3566	1249	4,813	0,025	65	14	4,619	0,029	115	3
	2	4,980	0,021	2800	1086	4,425	0,307	317	196	4,362	0,343	175	76
	3	4,9145	0,067	4910	1850	4,383	0,655	166	212	4,073	0,536		X
Learning rate	0.0001	4,808	0,077	6248	1880	4,513	0,349	117	125	3,920	0,799	128	X
	0.0005	4,980	0,021	2800	1086	4,425	0,307	317	196	4,362	0,343	175	76
	0.001	4,852	0,152	4929	2958	4,521	0,632	188	270	4,033	1,046	324	381
Training Time	5k	2,314	0,550	1096	X	2,333	1,165		XXX	1,140	1,069		XXX
	10k	3,481	0,300	3132	2473	3,512	1,369		X	3,193	0,789	198	116
	20k	4,697	0,105	5464	972	4,059	0,782		X	3,589	0,837		X
	40k	4,980	0,021	2800	1086	4,425	0,307	317	196	4,362	0,343	175	76
ϵ_d / Training time	0.5	4,998	0,011	3548	922	4,457	0,381	215	145	3,857	0,708	296	227
	0.75	4,980	0,021	2800	1086	4,425	0,307	317	196	4,362	0,343	175	76
	1	4,832	0,128	4276	2400	4,375	0,532	220	283	4,353	0,132	173	25
	1.5	4,733	0,124	6086	2312	4,025	0,702		X	3,391	0,655		X
Action Size	25	4,890	0,078	5225	386	3,925	0,422	471	290	3,197	0,942	679	337
	50	4,980	0,021	2800	1086	4,425	0,307	317	196	4,362	0,343	175	76
	100	4,973	0,038	5150	1744	4,613	0,047	116	6	4,085	0,134	247	15
	200	4,188	0,570	3983	497	1,370	0,776		XXX	0,821	0,778		XXX

TABLE D.1: DSARSA hyperparameter search training results

Below the training results using the DTAMER algorithm using one run for each parameter setting. The X indicate a run that did not reach a target state.

		Training		After training			
		ATS(S1)	TT(S1)	ATS(S1)	TT(S1)	ATS(S2)	TT(S2)
|HL|	10	3,833	1425	0,300	X	0,391	X
	25	4,954	2019	4,825	50	4,579	111
	50	4,998	1888	4,823	47	4,584	107
	75	4,984	1623	4,817	54	4,34	117
	100	4,999	2080	4,823	55	4,624	112
# HLs	1	4,869	2301	4,833	48	3,855	109
	2	4,873	2952	4,745	145	4,298	296
	3	4,957	3761	4,808	51	4,5	129
ϵ_d / Training Time	0.25	4,994	1330	4,559	110	4,473	109
	0.75	4,938	1372	4,783	67	4,143	181
	1.25	4,867	2743	4,828	51	4,568	116
Training Time	10K	4,514	2048	1,257	X	0,982	X
	20K	4,942	966	3,87	665	3,139	X
	20K	4,835	2349	4,266	157	4,234	443
	25K	4,923	2863	2,456	X	1,82	X
	30K	4,99	1543	4,767	58	4,096	188
Learning Rate	0.0001	2,892	6580	1,733	X	1,884	X
	0.0005	4,252	1685	1,00	X	0,608	X
	0.001	4,967	3392	4,115	297	4,212	343
	0.0015	4,996	1897	4,598	186	3,5	437
Reward length	2	4,881	1053	4,846	49	4,555	117
	8	4,437	6233	3,24	X	2,509	X

TABLE D.2: DTAMER hyperparameter search training results

Appendix E

VST

FIGURE E.1: Massive VST

Appendix F

User Questionnaire

'Interactive Musical Exploration' Questionnaire

1. **Your name**: _____
2. **How old are you?** I am _____ years old.
3. **Your profession**: _____
4. **Your experience with music production:** ○ none ○ amateur ○ professional ○ other _____
5. **How often do you use audio plug-ins?** ○ never ○ monthly ○ weekly ○ daily

About the experience

Please evaluate **your experience** for the two different interactions below by circling a number for each criteria.

6. **Agent exploration**

annoying	1—2—3—4—5	enjoyable
confusing	1—2—3—4—5	clear
conventional	1—2—3—4—5	inventive
obstructive	1—2—3—4—5	supportive
ambiguous	1—2—3—4—5	understandable
demotivating	1—2—3—4—5	motivating
dull	1—2—3—4—5	creative
inefficient	1—2—3—4—5	efficient
boring	1—2—3—4—5	exciting

7. **Manual exploration**

annoying	1—2—3—4—5	enjoyable
confusing	1—2—3—4—5	clear
conventional	1—2—3—4—5	inventive
obstructive	1—2—3—4—5	supportive
ambiguous	1—2—3—4—5	understandable
demotivating	1—2—3—4—5	motivating
dull	1—2—3—4—5	creative
inefficient	1—2—3—4—5	efficient
boring	1—2—3—4—5	exciting

About the interaction

Please evaluate the following:

8. **Evaluative control: super(dis)like**

useful:	bad	1—2—3—4—5	good
easy to use:	bad	1—2—3—4—5	good
satisfaction:	bad	1—2—3—4—5	good

9. **Exploration control: explore sound**

useful:	bad	1—2—3—4—5	good
easy to use:	bad	1—2—3—4—5	good
satisfaction:	bad	1—2—3—4—5	good

10. **PCA overview**

useful:	bad	1—2—3—4—5	good
easy to use:	bad	1—2—3—4—5	good
satisfaction:	bad	1—2—3—4—5	good

11. **Sound trajectories/dynamics during following explorations:**

manually:	bad	1—2—3—4—5	good
with agent:	bad	1—2—3—4—5	good

Closing Remarks

12. **If you have any comments, critiques or questions, you can write them here:**

_____

_____

Appendix G

User Experience Questionnaire Results

	User	Attract-iveness	Ergonomic quality				Hedonic quality			
			perspicuity		dependa-bility	efficiency	novelty		stimulation	
		enjoy-able	clear	understand-able	suppor-tive	efficient	inven-tive	crea-tive	motiva-tion	exci-ting
Agent	1	4	3	4	4	4	5	5	5	5
	2	4	2	4	3	3	3	4	4	4
	3	3	2	4	2	4	4	4	3	4
	4	3	4	4	3	3	5	4	5	4
	5	4	3	4	3	3	5	5	5	5
	6	3	4	5	3	3	4	3	4	3
	7	2	2	3	2	2	3	2	2	3
	8	4	3	1	5	4	4	5	3	4
	9	3	2	1	3	3	5	4	3	3
	10	4	4	4	3	4	3	4	4	3
	11	5	4	4	4	4	5	5	5	5
	12	3	2	5	3	3	1	1	3	3
	13	2	2	5	2	1	3	2	3	2
	14	2	5	5	4	2	4	4	3	3
Manual	1	5	5	4	4	5	3	5	4	4
	2	4	2	4	4	4	3	3	5	4
	3	4	4	4	4	3	2	4	3	5
	4	4	5	4	4	4	1	3	3	4
	5	2	5	5	3	4	1	3	3	3
	6	4	5	4	3	3	2	4	4	4
	7	4	4	4	3	3	3	3	3	3
	8	5	3	2	4	4	3	4	4	5
	9	3	5	5	5	2	1	4	2	5
	10	4	5	4	4	4	4	4	5	4
	11	5	5	4	5	4	2	5	5	5
	12	4	5	5	3	5	1	4	3	3
	13	4	5	5	3	5	1	3	3	4
	14	5	5	5	5	5	5	5	5	5

TABLE G.1: User questionnaire results for agent and manual exploration of 14 users (1. = Agent, 2. = Manual)

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. 'Apprenticeship learning via inverse reinforcement learning'. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 1.

[2] Saleema Amershi et al. 'Power to the people: The role of humans in interactive machine learning'. In: *AI Magazine* 35.4 (2014), pp. 105–120.

[3] Marcin Andrychowicz et al. 'Hindsight Experience Replay'. In: *CoRR* abs/1707.01495 (2017). arXiv: 1707.01495. URL: http://arxiv.org/abs/1707.01495.

[4] Andrew G Barto, Richard S Sutton and Charles W Anderson. 'Neuronlike adaptive elements that can solve difficult learning control problems'. In: *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 834–846.

[5] Michel Beaudouin-Lafon. 'Designing interaction, not interfaces'. In: *Proceedings of the working conference on Advanced visual interfaces*. ACM. 2004, pp. 15–22.

[6] Marc Bellemare et al. 'Unifying count-based exploration and intrinsic motivation'. In: *Advances in Neural Information Processing Systems*. 2016, pp. 1471–1479.

[7] Baptiste Caramiaux. 'Studies on the Gesture – Sound Relationship for Musical Performance'. Thèse de doctorat. Paris: Université Université Pierre et Marie Curie (Paris 6), 2012. URL: http://articles.ircam.fr/textes/Caramiaux11f/index.pdf.

[8] Baptiste Caramiaux and Atau Tanaka. 'Machine Learning of Musical Gestures.' In: *NIME*. 2013, pp. 513–518.

[9] Mark Cartwright, Bryan Pardo and Josh Reiss. 'Mixploration: Rethinking the audio mixer interface'. In: *Proceedings of the 19th international conference on Intelligent User Interfaces*. ACM. 2014, pp. 365–370.

[10] Paul F Christiano et al. 'Deep reinforcement learning from human preferences'. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4302–4310.

[11] Andy Cockburn et al. 'Supporting novice to expert transitions in user interfaces'. In: *ACM Computing Surveys (CSUR)* 47.2 (2015), p. 31.

[12] Nick Collins. 'Reinforcement Learning for Live Musical Agents.' In: *ICMC*. 2008.

[13] Mihaly Csikszentmihalyi. 'The creative personality'. In: *Psychology today* 29.4 (1996), pp. 36–40.

[14] Mihaly Csikszentmihalyi and Isabella Selega Csikszentmihalyi. *Optimal experience: Psychological studies of flow in consciousness*. Cambridge university press, 1992.

[15] Thomas Degris, Martha White and Richard S Sutton. 'Off-policy actor-critic'. In: *arXiv preprint arXiv:1205.4839* (2012).

[16] Nate Derbinsky and Georg Essl. 'Exploring reinforcement learning for mobile percussive collaboration'. In: *Ann Arbor* 1001 (2012), pp. 48109–2121.

[17] Paul Dourish. *Where the action is: the foundations of embodied interaction*. MIT press, 2004.

[18] Philippe Esling, Adrien Bitton et al. 'Generative timbre spaces with variational audio synthesis'. In: *arXiv preprint arXiv:1805.08501* (2018).

[19] Rebecca Fiebrink and Baptiste Caramiaux. 'The machine learning algorithm as creative musical tool'. In: *Handbook of Algorithmic Music* (2016).

[20] Rebecca Fiebrink, Perry R Cook and Dan Trueman. 'Human model evaluation in interactive supervised learning'. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2011, pp. 147–156.

[21] Rebecca Fiebrink et al. 'Toward Understanding Human-Computer Interaction In Composing The Instrument.' In: *ICMC*. 2010.

[22] Meire Fortunato et al. 'Noisy networks for exploration'. In: *arXiv preprint arXiv:1706.10295* (2017).

[23] Jules Françoise. 'Gesture-Sound Mapping by Demonstration in Interactive Music Systems'. In: *Proceedings of the 21st ACM international conference on Multimedia (MM'13)*. Barcelona, Spain, 2013. URL: http://architexte.ircam.fr/textes/Francoise13c/index.pdf.

[24] Jules Françoise et al. 'Probabilistic models for designing motion and sound relationships'. In: *Proceedings of the 2014 international conference on new interfaces for musical expression*. 2014, pp. 287–292.

[25] Judy A Franklin and Victoria U Manfredi. 'Nonlinear credit assignment for musical sequences'. In: *Second international workshop on Intelligent systems design and application*. 2002, pp. 245–250.

[26] Xavier Glorot and Yoshua Bengio. 'Understanding the difficulty of training deep feedforward neural networks'. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[27] Shane Griffith et al. 'Policy shaping: Integrating human feedback with reinforcement learning'. In: *Advances in neural information processing systems*. 2013, pp. 2625–2633.

[28] Nicolas Heess et al. 'Learning continuous control policies by stochastic value gradients'. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2944–2952.

[29] Nicolas Heess et al. 'Memory-based control with recurrent neural networks'. In: *arXiv preprint arXiv:1512.04455* (2015).

[30] Mark K Ho et al. 'Teaching with rewards and punishments: Reinforcement or communication?' In: *CogSci*. 2015.

[31] Andy Hunt and Marcelo M Wanderley. 'Mapping performer parameters to synthesis engines'. In: *Organised sound* 7.2 (2002), pp. 97–108.

[32] Charles Isbell et al. 'A social reinforcement learning agent'. In: *Proceedings of the fifth international conference on Autonomous agents*. ACM. 2001, pp. 377–384.

[33] Giulio Jacucci. 'Interaction as performance'. In: (2004).

[34] Max Jaderberg et al. 'Reinforcement Learning with Unsupervised Auxiliary Tasks'. In: *CoRR* abs/1611.05397 (2016). arXiv: 1611.05397. URL: http://arxiv.org/abs/1611.05397.

[35] Natasha Jaques et al. 'Tuning recurrent neural networks with reinforcement learning'. In: (2017).

[36] Sergi Jorda. 'Digital Lutherie Crafting musical computers for new musics' performance and improvisation'. In: *Department of Information and Communication Technologies* (2005).

[37] Diederik P Kingma and Jimmy Ba. 'Adam: A method for stochastic optimization'. In: *arXiv preprint arXiv:1412.6980* (2014).

[38] W Bradley Knox and Peter Stone. 'Interactively shaping agents via human reinforcement: The TAMER framework'. In: *Proceedings of the fifth international conference on Knowledge capture*. ACM. 2009, pp. 9–16.

[39] William Bradley Knox. 'Learning from human-generated reward'. In: (2012).

[40] Bettina Laugwitz, Theo Held and Martin Schrepp. 'Construction and evaluation of a user experience questionnaire'. In: *Symposium of the Austrian HCI and Usability Engineering Group*. Springer. 2008, pp. 63–76.

[41] Sylvain Le Groux and PFMJ Verschure. 'Towards adaptive music generation by reinforcement learning of musical tension'. In: *Proceedings of the 6th Sound and Music Conference, Barcelona, Spain*. Vol. 134. 2010.

[42] Timothy P Lillicrap et al. 'Continuous control with deep reinforcement learning'. In: *arXiv preprint arXiv:1509.02971* (2015).

[43] Robert Loftin et al. 'Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning'. In: *Autonomous agents and multi-agent systems* 30.1 (2016), pp. 30–59.

[44] Jonas Löwgren. 'Articulating the use qualities of digital designs'. In: *Aesthetic computing* (2006), pp. 383–403.

[45] James MacGlashan et al. 'Interactive learning from policy-dependent human feedback'. In: *arXiv preprint arXiv:1701.06049* (2017).

[46] Sridhar Mahadevan. 'Average reward reinforcement learning: Foundations, algorithms, and empirical results'. In: *Machine learning* 22.1-3 (1996), pp. 159–195.

[47] Gary Marchionini. 'Exploratory search: from finding to understanding'. In: *Communications of the ACM* 49.4 (2006), pp. 41–46.

[48] Volodymyr Mnih et al. 'Asynchronous methods for deep reinforcement learning'. In: *International Conference on Machine Learning*. 2016, pp. 1928–1937.

[49] Volodymyr Mnih et al. 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540 (2015), p. 529.

[50] Volodymyr Mnih et al. 'Playing atari with deep reinforcement learning'. In: *arXiv preprint arXiv:1312.5602* (2013).

[51] Andrew Y Ng. 'Feature selection, L 1 vs. L 2 regularization, and rotational invariance'. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 78.

[52] Andrew Y. Ng, Daishi Harada and Stuart J. Russell. 'Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping'. In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278–287. ISBN: 1-55860-612-2. URL: http://dl.acm.org/citation.cfm?id=645528.657613.

[53] Andrew Y Ng, Stuart J Russell et al. 'Algorithms for inverse reinforcement learning.' In: *Icml*. 2000, pp. 663–670.

[54] Ian Osband et al. 'Deep exploration via bootstrapped DQN'. In: *Advances in neural information processing systems*. 2016, pp. 4026–4034.

[55] François Pachet. 'The future of content is in ourselves'. In: *Computers in Entertainment (CIE)* 6.3 (2008), p. 31.

[56] Garth Paine. 'Interactivity, where to from here?' In: *Organised Sound* 7.3 (2002), pp. 295–304.

[57] Patrick M Pilarski et al. 'Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning'. In: *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*. IEEE. 2011, pp. 1–7.

[58] Siddharth Reddy, Sergey Levine and Anca Dragan. 'Shared Autonomy via Deep Reinforcement Learning'. In: *arXiv preprint arXiv:1802.01744* (2018).

[59] Mitchel Resnick et al. 'Design principles for tools to support creative thinking'. In: (2005).

[60] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering, 1994.

[61] Tuukka Ruotsalo et al. 'Interactive intent modeling: Information discovery beyond search'. In: *Communications of the ACM* 58.1 (2015), pp. 86–92.

[62] Jan C Schacher, Chikashi Miyama and Daniel Bisig. 'Gestural electronic music using machine learning as generative device.' In: *NIME*. 2015, pp. 347–350.

[63] John Schulman et al. 'High-dimensional continuous control using generalized advantage estimation'. In: *arXiv preprint arXiv:1506.02438* (2015).

[64] John Schulman et al. 'Trust region policy optimization'. In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[65] Hugo Scurto and Frédéric Bevilacqua. 'Appropriating Music Computing Practices Through Human-AI Collaboration'. In: *Journées d'Informatique Musicale (JIM 2018)*. 2018.

[66] Hugo Scurto, Rebecca Fiebrink et al. *Grab-and-play mapping: Creative machine learning approaches for musical inclusion and exploration*. Goldsmiths University of London, 2016.

[67] Ben Shneiderman. 'Creativity support tools: Accelerating discovery and innovation'. In: *Communications of the ACM* 50.12 (2007), pp. 20–32.

[68] David Silver et al. 'Deterministic policy gradient algorithms'. In: *ICML*. 2014.

[69] David Silver et al. 'Mastering the game of Go with deep neural networks and tree search'. In: *nature* 529.7587 (2016), pp. 484–489.

[70] Denis Smalley. 'Spectromorphology: explaining sound-shapes'. In: *Organised sound* 2.2 (1997), pp. 107–126.

[71] Malcolm Strens. 'A Bayesian framework for reinforcement learning'. In: *ICML*. 2000, pp. 943–950.

[72] Richard S Sutton. 'Learning to predict by the methods of temporal differences'. In: *Machine learning* 3.1 (1988), pp. 9–44.

[73] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.

[74] Hado Van Hasselt, Arthur Guez and David Silver. 'Deep Reinforcement Learning with Double Q-Learning.' In: *AAAI*. Vol. 16. 2016, pp. 2094–2100.

[75] Ziyu Wang et al. 'Dueling network architectures for deep reinforcement learning'. In: *arXiv preprint arXiv:1511.06581* (2015).

[76] Ziyu Wang et al. 'Sample efficient actor-critic with experience replay'. In: *arXiv preprint arXiv:1611.01224* (2016).

[77] Garrett Warnell et al. 'Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces'. In: *arXiv preprint arXiv:1709.10163* (2017).

[78] Christopher JCH Watkins and Peter Dayan. 'Q-learning'. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[79] Christopher John Cornish Hellaby Watkins. 'Learning from delayed rewards'. PhD thesis. King's College, Cambridge, 1989.

[80] Ronald J Williams. 'Simple statistical gradient-following algorithms for connectionist reinforcement learning'. In: *Reinforcement Learning*. Springer, 1992, pp. 5–32.

[81] Victor Zappi and Andrew McPherson. 'Dimensionality and Appropriation in Digital Musical Instrument Design.' In: *NIME*. 2014, pp. 455–460.

[82] Dongbin Zhao et al. 'Deep reinforcement learning with experience replay based on sarsa'. In: *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE. 2016, pp. 1–6.