Department of Music,
Norwegian University of
Science and Technology

Department of Musicology,
University of Oslo

# Reinforcement learning for use in cross-adaptive audio processing

Ulrik Antoniussen Halmøy

May, 2021

Supervisor: Øyvind Brandtsegg

Master's programme in Music, Communication and Technology

**Abstract**

This thesis is a study of reinforcement learning as a possible method for finding mappings in cross-adaptive audio effects. The context of a cross-adaptive performance session is framed as a reinforcement learning environment, in which the idea is to modify one audio signal by the features of another. The results show that reinforcement learning is a feasible way to generate such mappings. Still, there are affordances of using reinforcement learning for this purpose that are yet to be explored. The thesis also proposes a system architecture design that allows for real-time performance with the machine learning agent but is yet to be adequately implemented and tested. Future work could look into how to integrate the system in musical practice and to extend the cross-adaptive reinforcement learning environment to incorporate human feedback in the learning process.

**Sammendrag**

Denne oppgaven tar for seg hvordan forsterkende læring kan fungere for å generere parameterkombinasjoner i kryssadaptive lydeffekter. Et scenario som innebefatter to musikere som spiller sammen, hvor kryssadaptive prosesseringsteknikker har blitt tatt i bruk, har blitt modellert som et forsterkende læringsmiljø, hvor ideen er at lyden til den ene musikeren påvirkes av de akustiske egenskapene til den andre musikeren. Resultatene viser at forsterkende læring fungerer for å finne slike parameterkombinasjoner. Samtidig åpner forsterkende læring opp et handlingsrom innen kryssadaptiv prosessering som det fortsatt er mulig å utforske ytterligere. Denne oppgaven foreslår også en systemarktitektur som muliggjør sanntidsprosessering av kryssadaptive lydeffekter med forsterkende læring. Arktitekturen er delvis implementert, så det gjenstår fortsatt å implementere denne fullstendig, og deretter å teste den med musikere. Videre arbeid kunne undersøkt hvordan systemet som er implementert i denne oppgaven kan forenes med musikeres vanlige måte å lage musikk på, og å utvide det eksisterende systemet til å bruke menneskelig tilbakemeldinger i læringsprosessen.

**Acknowledgements**

I wish to thank Øyvind Brandtsegg for being my supervisor, and for the friendly and helpful collaboration. Your ability to perceive the nuances in sound is inspiring. Thanks to my family, friends, and to Maria, in particular, for being supportive throughout my studies and life in general. Thanks to the MCT staff and my fellow MCT students for providing me with many inspiring perspectives throughout the master's program.

Finally, I want to thank my older brother Viktor. I'm incredibly grateful that you shared your passion for music with me.

# Table of contents

# Chapter 1

# Introduction

## 1.1 Context

This thesis explores reinforcement learning in cross-adaptive audio processing. It builds upon the research project titled "Cross-adaptive processing as musical intervention", led by the Music Technology Group at the Department of Music at NTNU from 2016 to 2018. Øyvind Brandtsegg, the supervisor of this thesis, was the leader of the project.

The research project studied musical performance sessions where cross-adaptive techniques for audio processing were utilized and evaluated [Brandtsegg et al., 2018][Emmerson et al., 2018]. Cross-adaptive audio processing refers to the process of modifying the timbral characteristics of one audio signal with a digital audio effect whose parameters are modulated by the features of one or more other audio signals. Research in the project looked into the technical inventions for cross-adaptive processing and their implications in performance, aesthetics, and philosophy. Several studies in the cross-adaptive research project focused on the performative aspect of cross-adaptive processing, investigating the influence of this music technological intervention on more traditional musical practices.

## 1.2 Motivation

Some of the findings in the cross-adaptive research project show that the choice of audio effects and feature extractors, and the way that the mappings between them are configured, have a large impact on the musical performance [Brandtsegg et al., 2018]. Setting up such a signal flow for a musical performance can also be quite elaborate, as

demonstrated in Sarkar et al. [Sarkar et al., 2017]. It can be complex to find interesting mappings between features and effect parameters manually. For many-to-many mappings between features and effect parameters, the number of possible combinations gives an enormous space to explore. The desire for a tool to help find mappings in that space is what motivated Jordal's master thesis [Jordal, 2017]. The problem was briefly addressed by Brandtsegg in his development of a toolkit for cross-adaptive processing [Brandtsegg, 2015], who then suggested that machine learning could be a useful framework for finding these mappings automatically. The goal of Jordal's thesis was to automate the process of mapping features from a target audio signal to effect parameters to affect a source audio signal [Jordal, 2017]. The metric for determining the quality of a mapping was based on similarity between the two audio signals, so that the best possible mappings were those that affected the source audio signal in such a way that it sounded as close as possible to the target audio signal.

This thesis shares much of the same motivation as Jordal's thesis, namely how high-level mappings can be constructed from audio features in a target sound to affect the timbre of a secondary sound source through parameters in an audio effect. Even though Jordal was successful in his project, there are other approaches to the problem that might be worth exploring.

The primary goal of this thesis has been to explore reinforcement learning for use in cross-adaptive audio effects. Reinforcement learning is a form of machine learning, often called the third branch of machine learning next to supervised and unsupervised learning. In the context of this thesis, a *cross-adaptive audio effect* is referred to as an intelligent adaptive audio effect, as defined by Reiss and Brandtsegg [Reiss and Brandtsegg, 2018] and illustrated in figure 1.1. An intelligent audio effect comprises feature extraction, feature processing, and converting the processed features into control signals. The *intelligent* part of the cross-adaptive audio effect is where reinforcement learning comes in. A reinforcement learning agent learns from the audio features and converts them into control signals according to the goals set in the learning process.

The reinforcement learning process is analogous to that of a collaborative music performance context, in which one musician has to observe, analyze and respond to what the other musicians are playing. Similarly, in the reinforcement learning process, a virtual agent learns from interacting with its surroundings. The analogy between collaborative music performance and reinforcement learning is not only conceptually attractive, but it also has implications for the technical implementation of the intelligent adaptive audio effect. Reinforcement learning does not require a stationary dataset to train a machine learning model. Instead, reinforcement learning agents are trained on a reward signal

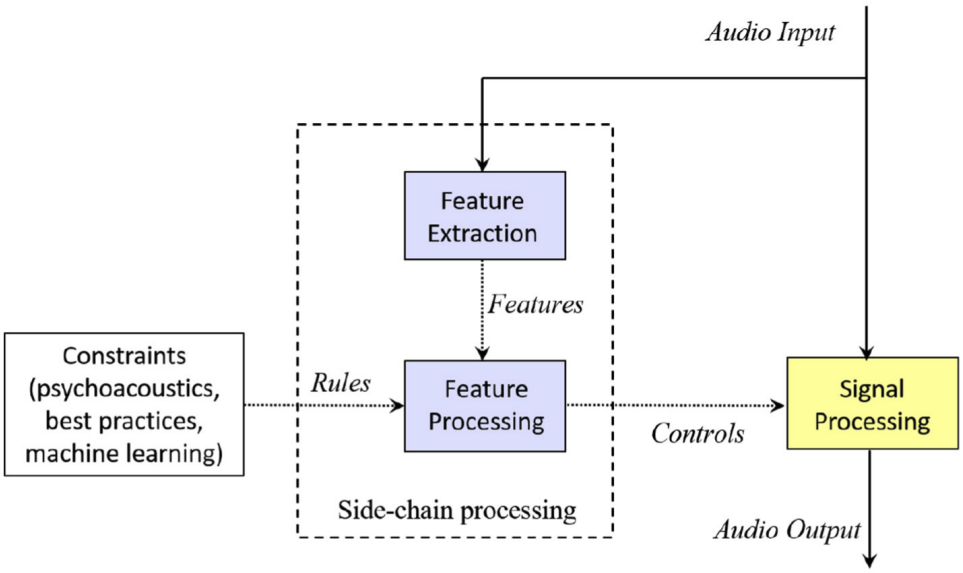



Figure 1.1: Intelligent audio effect. Figure credit: Reiss and Brandtsegg [Reiss and Brandtsegg, 2018]

that they receive for every step they take in the environment. This makes it possible to train reinforcement learning models while they are being used, and affords multiple ways of interacting with the model.

## 1.3 Research topic and contributions

The research topic of this thesis has been to study how a simplified version of a cross-adaptive performance scenario can be modeled as a reinforcement learning environment with the goal of creating mappings in cross-adaptive audio effects.

To narrow the scope of this thesis, the main emphasis has been to bridge the domains of reinforcement learning and cross-adaptive processing—less on the characteristics of various cross-adaptive processing scenarios. Consequently, well-known audio effects and feature extractors are reused from previous projects in the cross-adaptive research project. Even though there is no limit to how many signals can affect each other in the cross-adaptive processing, it is assumed here that there is one target sound that is shaping the output of another sound, hereby referred to as target and source.

The contributions of this thesis are mainly the design and implementation of a prototype of one possible way of modeling a cross-adaptive performance scenario as a reinforcement learning environment. Various results from testing the implementation have been evaluated and discussed to provide potential directions for further integrating the system into

musical practices. Additionally, a framework for running real-time inference, i.e., to generate mappings in real-time, on a pre-trained cross-adaptive machine learning model has been proposed and implemented. The system appears to work adequately for the tasks outlined in this thesis, although rigorous testing and bug fixing remains.

The total delivery of this thesis consists of text, code, and audio samples. To make the deliveries available for anyone reading this, both the code and the audio samples exist on public web pages. The code for the implementation of the cross-adaptive environment is open-source and is hosted on GitHub[1]. The repository includes a guide on how to set up the project locally. The audio samples have been published in a separate webpage[2]. The webpage should be consulted while reading the results chapter, as the audio samples will help understand the results. The reader will get a reminder in the relevant sections. Both links are also listed in the appendix A of this report.

Additionally, a blog post has been written as a supplement to this thesis[3], as is the custom in the master's program in Music, Communication, and Technology. The blog post is a brief and less formal version of this thesis, written to target a broader audience. The blog post does not contain any information that is not already presented in this thesis and its appendices.

[1]Source code repository: `https://github.com/ulrikah/rave`
[2]Audio samples: `https://ulrikah.github.io/thesis`
[3]Blog post: `https://mct-master.github.io/masters-thesis/2021/05/15/ulrikah-rave.html`

# Chapter 2

# Background

## 2.1 Cross-adaptive processing

Cross-adaptive processing originates from 2001, when Verfaille and Arfib first formalized the term *adaptive audio effect* [Verfaille and Arfib, 2001]. They define an adaptive audio effect as a combination of a sound transformation with a time-varying adaptive control. The adaptive control comes from the extraction of low-level and perceptual features from the audio signal. It is mapped to the parameters of the audio effect by a specified mapping function.

An example of a famous adaptive audio effect is the autotune, in which the estimated fundamental frequency of a signal is used to shift the pitch of the input signal into quantized steps. Since this adaptive effect uses features from the input signal to modify the output signal, it is defined as auto-adaptive by Verfaille et al. [Verfaille et al., 2006]. Cross-adaptive audio effects on the other hand, uses external input signals for the adaptive control of a source signal.

The most noteable applications of cross-adaptive processing are live performance and automatic mixing [Reiss and Brandtsegg, 2018]. Perez-Gonzalez and Reiss [Perez-Gonzalez and Reiss, 2009] and Chourdakis and Reiss [Chourdakis and Reiss, 2016][Chourdakis and Reiss, 2017] have showed that cross-adaptive methods can be used for automatical equalization of multi-channel audio and for applying aritifical reverberation based on desired characteristics, respectively. Both works have relevant applications for automatic mixing. Campbell et al. developed ADEPT (Adaptive Digital Effects Processing Tool), a framework for using adaptive digital audio effects in traditional digital audio workstations (DAW) [Campbell et al., 2016]. Several of the current implementations of tools for

cross-adaptive processing exist only as prototypes and are not yet integrated into existing music production and performance workflows, including the implementation done in this thesis. ADEPT is a push towards integrating adaptive audio effects in environments that are widely used in music production and performance.

### 2.1.1 Digital audio effects

Digital audio effects are the backbone of cross-adaptive processing. A digital audio effect refers to applying one or more digital signal processing techniques to modify or transform an audio signal [Verfaille et al., 2006]. Digital audio effects are applied to fixed-length blocks of audio samples whose lengths are typically powers of two, e.g., 128, 512, 1024. These blocks are also called audio frames.

An example of a very basic audio effect is a gain control. Gain affect the intensity of an input signal, and is perceived as loudness. At the digital signal processing level, a gain effect can be implemented by multiplying the sample values in the audio frame by the desired control value. A control value greater than one will increase the amplitude of the input signal, resulting in a louder signal, and a control value of less than one will do the opposite.

The audio effect that was used in this thesis is a combination of a resonant low-pass filter, distortion and a gain control. The resonant low-pass filter and the distortion will be briefly explained.

#### Resonant low-pass filter

A low-pass filter is a digital audio filter that lets frequencies below a cutoff frequency pass through and attenuates frequencies above the cutoff frequency [Zölzer et al., 2002]: the lower the cutoff frequency, the less high-frequency content. Resonance creates a peak in the frequency response, in a narrow band around the cutoff frequency.

In this audio effect, the controllable parameters are cutoff frequency and the amount of resonance. Figure 2.1 shows the spectrogram of noise being sent through the resonant low-pass filter used in this project. The cutoff frequency was set to 500 Hz together with a high resonance value. Note that the y-axis is logarithmically scaled.

There are many ways to implement a low-pass filter. The implementation used in Brandt-

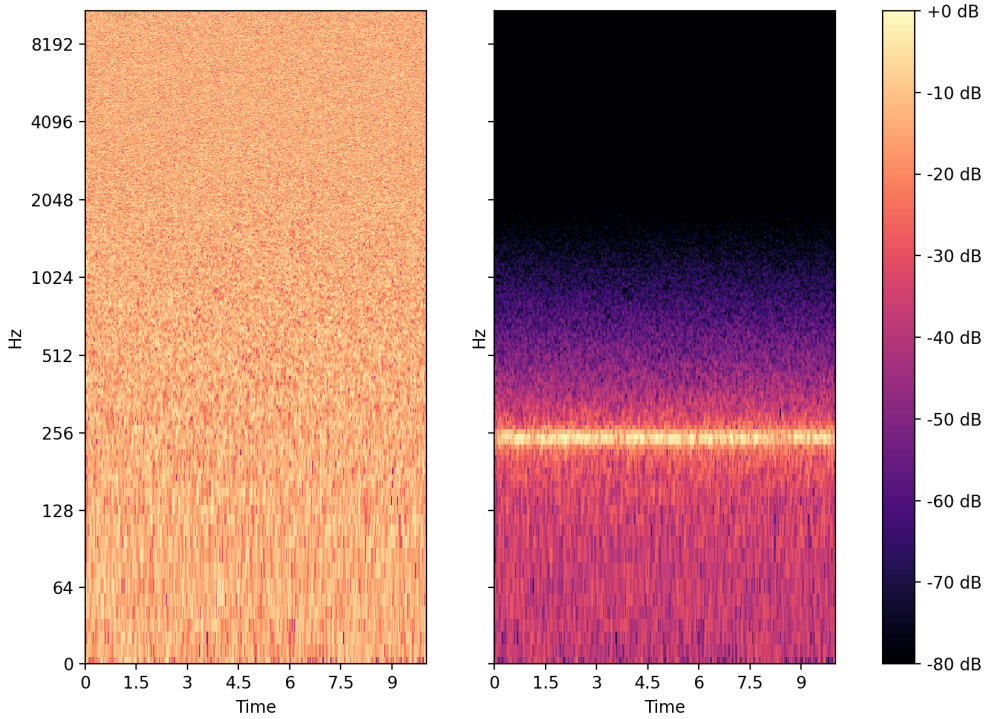



Figure 2.1: Spectrogram of noise processed by a resonant low-pass filter

segg [Brandtsegg, 2015] uses an opcode from Csound[1] which provides an additional control over the amount of distortion in the output signal.

### Distortion

Distortion adds new partials to an audio signal. In this context, the distortion is based on the hyperbolic tangent function *tanh*, which is known to create harmonic distortion.

Distortion is integrated into the implementation of the low-pass filter in the previous section. Figure 2.2 shows the effect of applying a high distortion value to the filtered noise signal. The spectrogram shows that distortion introduces additional harmonics above the peak of the cutoff frequency. The other parameter values for the filter were set the same as in figure 2.1.

The combination of lowpass filter, resonance and distortion thus facilitates a flexible sound shaping tool, as one can attenuate harmonics with the filter, amplify a selected range with the resonance, and create new harmonics with the distortion.

[1] `http://www.csounds.com/manual/html/lpf18.html`

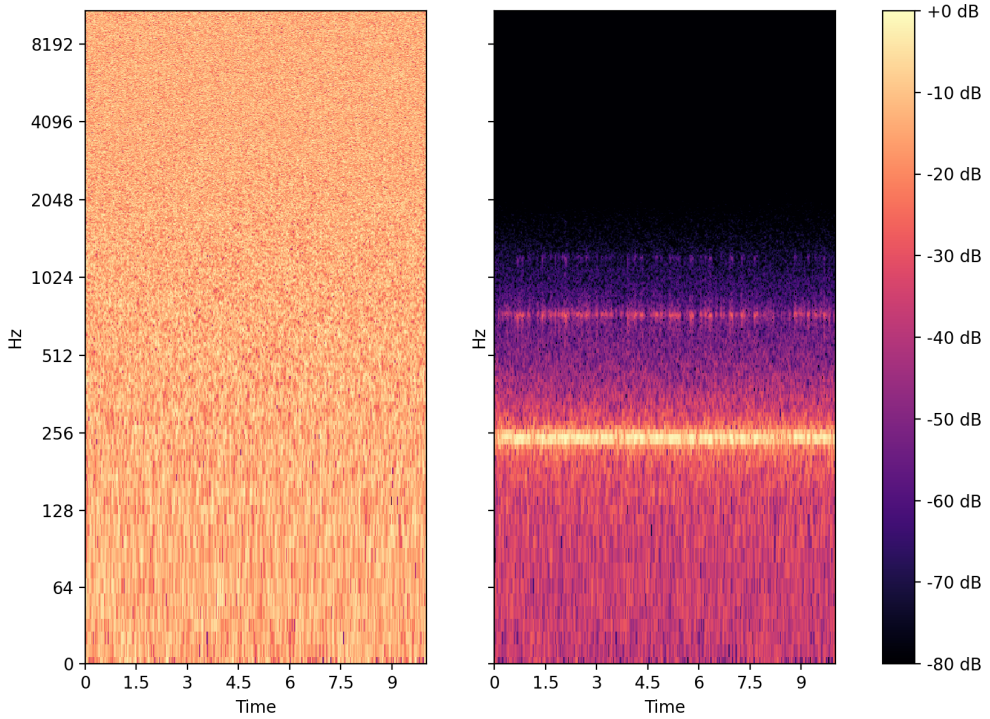



Figure 2.2: Spectrogram of noise processed by a resonant low-pass filter with distortion

As a side note, the hyperbolic tangent function is also used as activation function in the neural network that will be explained later. Even though the reasons to use *tanh* in digital signal processing and in neural networks are very different, they do the exact same job in both domains, namely to "squash" some input into the range of $-1$ to 1.

### 2.1.2 Automatic mappings in cross-adaptive effects

Automatic mappings in cross-adaptive audio effects are interesting because they allow for exploration of high-dimensional relations between audio features and effect parameters that otherwise would be hard to find by manually constructing mapping functions.

One of the contributions to the cross-adaptive research project was a master thesis from Eldhuset [Eldhuset, 2016]. Eldhuset showed that it was feasible to use genetic algorithms to find mappings in cross-adaptive audio effects based on fitness functions that compared audio features from two sound sources. The year after, the master thesis from Jordal iterated upon Eldhuset's work with many improvements that lead to a more flexible and detailed solution to the problem. Notably, Jordal concluded that neuroevolution, a technique that uses evolutionary algorithms to train artificial neural networks (ANN), was

feasible for finding automatic mappings in cross-adaptive audio effects [Jordal, 2017]. Neuroevolution was motivated by the fact that no dataset was available for a typical supervised machine learning approach, which prevented using backpropagation to train the ANN. Backpropagation, a popular algorithm for training neural networks [LeCun et al., 2012], uses the gradient of some performance metric (typically defined by the loss between the source and target) to update its weights. However, in reinforcement learning, one can directly use the reward signal as the performance metric in backpropagation. Hence, the neural network used in this thesis is trained by using backpropagation. Backpropagation does not necessarily perform better or worse than neuroevolution, but it is a different approach to learning ANNs.

Evolutionary algorithms have similarities with reinforcement learning. Both learn from the same kind of feedback signal (called reward function in reinforcement learning and fitness function in evolutionary computing), which essentially is a function that describes how good a proposed solution is. Salimans et al. claim that evolutionary strategies, one type of evolutionary algorithms, can have advantages over policy gradient algorithms (the type of reinforcement learning algorithms used in this thesis, see 2.2.2) upon certain conditions [Salimans et al., 2017]. These conditions can be when there are many steps per episode, the actions have long-lasting effects, and it is difficult to estimate a value for the current state [Salimans et al., 2017]. The reinforcement learning terminology will be explained shortly, but it suffices to say that there are trade-offs in both evolutionary algorithms and reinforcement learning that can yield better or worse results, depending on the problem at hand.

### 2.1.3 Music representation and feature extraction

The general dichotomy of representing music digitally is whether the representation is in a *symbolic* form, e.g., score sheets, MIDI notes, and piano rolls, or as *audio signals*. Physical audio (sound) is a continuous stream of oscillations in air pressure. As digital computers can not represent a continuous signal, representing sound as a digital audio signal involves sampling it at a high rate. This rate is called the *sampling rate* and is typically set to 44100 Hz for popular music applications to capture most of the relevant frequency domain. Consequently, a digital audio signal is a series of discrete floating-point values.

In a machine learning context, a major difference between symbolic representations and audio signals is the amount of volume the data comprises. One second of a pianist playing a note on the piano might be represented symbolically as a tuple consisting of four values:

*(onset, offset, pitch, velocity).* An audio signal representation of that same second of audio can be represented as a series of 44100 floating-point values. The advantage of an audio signal representation is that it can capture many more nuances in the music, such as the *timbre* of the instrument. The abovementioned symbolic representation gives no information of which instrument the note was played on, or of the size of the room it was played in. Even though it is not trivial to extract this information computationally from the audio signal, a trained human ear can hear it when playing the audio signal back over a loudspeaker. Recent advances in the domain of music information retrieval and machine learning have enabled complex processing tasks with audio signal representations, such as extracting the reverb from audio signals or doing complete timbral transformations from one instrument onto another [Engel et al., 2020].

In a musical context, feature extraction is the process of extracting musical content descriptors from audio signals [Schedl et al., 2014]. As an example, the perceptual descriptor loudness can, to a large extent, be described by the root mean square (RMS). RMS is extracted from an audio signal $x$, which is represented digitally by samples $x_1, x_2, ..., x_n$ by the following formula:

$$RMS(x) = \sqrt{\frac{1}{n}(x_1^2 + x_2^2 + ... + x_n^2)}$$

Typically, features are extracted by partitioning an audio signal into fixed-sized audio frames and then by using a feature extraction method for every frame. If the size of a frame is relatively small, it is possible to extract features from an audio signal in what is commonly referred to as real-time. This is done by storing a buffer of the most recent samples and then performing a calculation once the buffer has filled up. A typical buffer size for audio processing is 128 samples. This corresponds to approximately three milliseconds of an audio signal sampled at 44100 Hz, which is the time it takes for sound to travel about one meter (considering the speed of sound to be roughly 340 m/s). Although some feature extractors can be computationally expensive, the total latency of applying an effect to an audio buffer is below what is perceivable in musical performances. Even though low latency is desirable in audio processing systems, it usually comes at the cost of a higher processor load since the load increases as the audio buffer size get smaller.

RMS is one of the simplest features and is extracted in the time domain. Some more complex feature extraction methods are available in the frequency domain. Fourier transforms can be used to convert a signal from the time domain to the frequency domain [Schedl et al., 2014]. Several of the feature extractors that are used in this thesis operate in the frequency domain, such as spectral centroid, spectral spread, spectral flatness, and

spectral flux.

## 2.2 Reinforcement learning

Reinforcement learning is a computational approach to learning from interaction. Although reinforcement learning nowadays is often seen as a third paradigm in machine learning next to supervised and unsupervised machine learning, traditional approaches to reinforcement learning did not involve machine learning as the term is understood today. The independent studies of the psychology of animal learning and optimal control in robotics and cybernetics intersected in the late 1980s and formed the field of reinforcement learning [Sutton and Barto, 2018].

The terminology in reinforcement learning consists of multiple concepts that are important to define. The *agent* is the subject that performs *actions*. The scene or context in which the agent acts is called the *environment*. The goal in reinforcement learning is to learn to maximize the *reward* that the environment gives to the agent based on its actions [Sutton and Barto, 2018]. *Return* is used to describe a sequence of rewards over time, and the reinforcement learning objective is often formulated as maximizing the *expected return*. The *policy* is the agent's brain and determines how the agent should map observations to actions. Furthermore, a reinforcement learning algorithm is what is used to update the policy through iterative training. A policy can either be deterministic or stochastic, but only the latter is used in this project. A stochastic policy samples actions from a normal distribution over probable actions. Stochastic policies are very practical as they allow the agent to choose the degree of exploration by sampling actions that are further away from the mean of the distribution. The reward is a metric that signify how (well) the agent performs in the environment. The reward is distributed from a reward function, which is a function of the state of the environment. The reward function is defined by the creator of the environment. The agent receives a reward for every action it takes in the environment, and may be negative, positive, or neutral. The agent acts based on *observations* of the environment. Formally, partial representations of the environment are called observations, and complete representations are called states. In practice, the terms observation and state are used interchangeably. One iteration in the reinforcement learning cycle (see figure 2.3) is typically referred to as a step, and consist of a $(state, action)$ pair. The sets of possible observations and actions are often referred to as observation space and action space, respectively. These spaces are vector spaces, as both observations and actions ultimately need to be represented by vectors. The observation and action spaces can either be discrete or continuous, depending on the environment. An

example of a discrete action space could be the positional controls of a virtual character, e.g., up, down, left, or right, whereas a continuous action space could be the rotational angle of one or more joints in a physical machine. Some reinforcement learning algorithms only work in environments that object to specific conditions, such as Deep Q-Network, which require the action space to be discrete [Mnih et al., 2013].

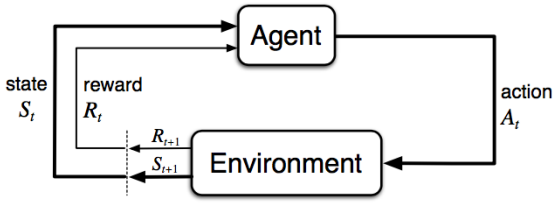



Figure 2.3: Reinforcement learning cycle. Figure credit: Sutton and Barto [Sutton and Barto, 2018]

An example of a reinforcement learning environment is the cart-pole environment, a classic problem in reinforcement learning [Barto et al., 1983]. The problem is to balance a pole on cart for as long as possible. The cart can be moved along the horizontal axis. A positive reward is given for every step that the agent is able to keep the pole upright. The episode terminates whenever the angle between the ground and the pole drops below a certain threshold, or whenever the cart moves too far from the centre. The action space in the cart-pole problem is discrete: the possibilities are to push the cart left or right with a fixed force. An observation of the cart-pole environment includes information on the cart position, the cart's velocity, the pole angle and the pole's angular velocity. All these four values are represented by scalar values within real-valued ranges. Therefore, the observation space is continuous.

The *cross-adaptive environment*, which will the object of study in this thesis, is a simplified model of musical performance scenario. The environment consists of two audio sources and a digital audio effect. Observations in the cross-adaptive environment are vectors with extracted features for the most recent audio frame from both audio signals concatenated together. These features are extracted by applying known digital signal processing algorithms that return information on the characteristics of the sound. The possible actions in the cross-adaptive environment are to choose the values of the effect parameters in the digital audio effect. Hence, both the observation space and action space are continuous in the cross-adaptive environment. The implementation of the cross-adaptive environment will be detailed in section 4.2.

### 2.2.1 Exploration and exploitation

At the beginning of a training process, the agent starts without any observations of the environment. Thus, all possible actions in the action space are equally probable. To start somewhere, the agent typically begins *exploring* the environment by sampling random actions from the action space and collecting its corresponding rewards. After some amount of steps, the agent will hopefully have collected enough state-action-reward tuples to infer patterns from them. At this point, the agent should start to *exploit* its policy to reinforce the actions that lead to positive rewards. However, the agent still needs to continue exploring. The trade-off between exploration and exploitation is a central problem in reinforcement learning. There are multiple strategies for how an agent should explore and exploit the environment. These strategies are typically embedded within a reinforcement learning algorithm. The exploration strategies used in this thesis project will be detailed in the description of the respective algorithms.

### 2.2.2 Learning a policy

The goal for an agent in reinforcement learning is to learn a policy $\pi$ that maximises the expected return. Such a policy can be called the *optimal policy* [Sutton and Barto, 2018]. The optimal policy, denoted $\pi^*$, can be defined by the following mathematical notation:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\ J(\pi)$$

where

$$J(\pi) = \underset{\tau \sim \pi}{E}[R(\tau)]$$

denotes the expected return $R$ for a sequence of state-action pairs $\tau$:

$$\tau = (s_0, a_0, s_1, a_1, ..., s_t, a_t)$$

The notation $\tau \sim \pi$ means that $\tau$ is sampled from the stochastic policy $\pi$.

A continuous action space implies that the number of actions is infinite. Thus, in environments where either the observation space or the action space is continuous, deriving

the optimal policy is a matter of approximation. There exist a variety of algorithms that detail how the agent can approximate an optimal policy. One family of these approximation algorithms is called *policy gradient algorithms*. Policy gradient algorithms seek to optimize a *parametrized policy* $\pi_\theta$, where $\theta$ denotes the parameters of the policy.

The objective in policy gradient methods is to maximize some performance measure by doing *gradient ascent* based on the gradient of the expected return $J(\pi_\theta)$:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\pi_{\theta_t})$$

where $\alpha$ is the learning rate.

Taking the gradient of $J(\pi_\theta)$ requires the parametrized policy $\pi_\theta$ to be differentiable with respect to its parameters $\theta$ [Sutton and Barto, 2018]. The gradient $\nabla J(\pi_\theta)$ is a way of measuring the change in performance w.r.t. to all the parameters that constitute the policy. $\nabla J(\pi_\theta)$ is useful because it allows tracking which parameters contribute to the increase or decrease in performance.

Since Mnih et al. were successful in playing Atari games with neural networks in 2013 [Mnih et al., 2013], recent advances in reinforcement learning typically use artificial neural networks (ANN) to parametrize the policy.

ANNs are probabilistic models inspired by the perceptual mechanism of the human brain and are made up of layers of neurons that are interconnected by weighted edges in a network-like structure [Rosenblatt, 1958][Nielsen, 2015]. The first and last layers are called the input and output layers, respectively, and all layers in between are called hidden layers. The input layer accepts a vector of numerical input whose size corresponds to the number of neurons in the input layer. In a feedforward neural network, which is the type of ANN used in this project, the input is forwarded through the network to produce an output at the output layer. Forwarding the input vector through the network means taking the dot product of the input layer and weigths of the next layer, then using the result of that operation to do the same for the next layer, and so on until the last layer is reached. Every dot product is activated by the neurons on the other side of the edges by some constant activation function, together with the neuron's bias. The point of the activation function is to be able to make non-linear mappings between the input and the output.

ANNs have the property of being universal function approximators, even with only a single hidden layer [Hornik, 1991]. This makes them very useful for scenarios where a function is

desired to perform a certain task that is difficult to express using traditional mathematical functions. The way that ANNs are designed to approximate functions is where the concept of machine learning comes in. The numerical value of the weights and biases of the neurons is what determines the functional capabilities of the ANN. Hence, these weights and biases have to be tuned to approximate the desired function. The process of tuning the weights and biases is an iterative learning process, where the ANN learns from pre-defined numerical success criteria, called a loss function, given example inputs. In practice, this happens by taking the gradient of the loss with respect to the weights and biases in the ANN, and nudging the values with incremental steps in the direction of the gradient (or sometimes in the opposite direction, depending on the objective). The most common algorithm for computing this gradient is called backpropagation, and essentially works by recursively taking partial derivatives of the loss w.r.t. all the weights and biases in the ANN, starting from the last layer and proceeding backward [Rumelhart et al., 1986][LeCun et al., 2012]. In order for an ANN to effectively learn anything, this process has to be repeated for many iterations, with the idea that the network will start to converge towards an optimum. In reinforcement learning, where the goal is to achieve as high a reward as possible, the objective is to find a local maximum for the gradient. This process is called gradient ascent. In the case of supervised learning, where the goal typically is to make the distance between two feature vectors as close as possible, the objective is to find a local minimum of the error between them, i.e., to minimize the loss. Conversely, this process is called gradient descent.

When an ANN is used to parametrize a reinforcement learning policy, the parametrization $\theta$ can be thought of as a matrix containing all the weights and biases of the network [Sutton and Barto, 2018]. Since it is possible to use backpropagation to calculate the partial derivative of the reward w.r.t. the weights and biases of the ANN (the parametrization of the policy), this does indeed fulfill the requirement of the policy to be differentiable w.r.t. its parameters.

Artificial neural networks are used to parametrize the policy also in this thesis. For most use cases, cross-adaptive processing involves mapping a set of audio features to a set of effect parameters. Using ANNs is a flexible way of doing just that. Jordal's work has also previously demonstrated that neural networks are good at finding mappings in cross-adaptive effects [Jordal, 2017]. In his case, using a framework of evolutionary computing.

As explained in the previous section, the cross-adaptive environment is an environment where both the observation space and the action space are continuous. Hence, learning a *cross-adaptive policy* requires the algorithm to work for continuous action spaces. In the results chapter, two state-of-the-art policy gradient algorithms are compared; Proximal

Policy Optimization (PPO) and Soft Actor-Critic (SAC). Both of these algorithms handle continuous action spaces. The rationale for choosing these two algorithms, among other algorithms that handle continuous action spaces, is that they have high scores on benchmarks in the reinforcement learning literature [Haarnoja et al., 2018a][Schulman et al., 2017], but approach the learning problem in different ways.

### 2.2.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a relatively simple on-policy algorithm that is not too far from the basic formulation of a policy gradient algorithm [Schulman et al., 2017]. On-policy algorithms sample all actions according to the latest version of the policy, meaning that they do not reuse experiences from the past [Achiam, 2018].

The idea behind PPO is to provide bounds on how much the policy can be updated from step to step, by only allowing updates within some trusted region in the proximity of the parameters of the current policy [Schulman et al., 2017]. The reason why one would want to do that is that vanilla policy gradient methods have shown that too large updates to the policy may be destructive for the overall evolution of the policy [Achiam, 2018]. The idea is similar to that of setting too high a learning rate for neural networks. If the learning rate becomes too high, then the model risk stepping over potential local optima.

The authors of the original PPO paper introduced two methods for how to set these bounds to the policy updates. The method chosen in this project clips the updates if they are outside the bounds.

Contrary to SAC, PPO does not have a built-in exploration strategy. Since PPO deals with stochastic policies, the amount of exploration is related to the randomness in action sampling from the policy. This randomness is determined by manually selected hyperparameters that schedule the amount of entropy in action selection throughout training.

### 2.2.4 Soft Actor-Critic

Soft Actor-Critic (SAC) is an off-policy algorithm [Haarnoja et al., 2018a]. Contrary to on-policy algorithms, off-policy algorithms aim to reuse experiences from past interactions in the environment. This generally makes off-policy algorithms more sample efficient [Haarnoja et al., 2018a].

The trade-off with off-policy methods is that they can be slower and more complex, as they require training an approximation for a so-called Q-function. A Q-function is a function that returns the value of taking a specific action in a given state. There is another family of reinforcement learning methods devoted to learning the Q-function, called Q-learning. In continuous action spaces, and for large discrete action spaces, it is practically impossible to provide an exact value of every action in every state, and this is why Q-functions often are approximated. Silver et al. introduced a method for learning a Q-function and a policy at the same time in their paper on Deterministic Policy Gradient algorithms [Silver et al., 2014]. SAC iterates on this idea, and the typical SAC implementations actually approximate two separate Q-functions to avoid overestimation in the approximation, which was found to be a problem for the Deterministic Policy Gradient algorithms [Achiam, 2018].

A key idea in SAC is its exploration strategy. SAC does not explore explicitly as PPO, but instead regularizes the amount of entropy in the policy as part of the reward function [Haarnoja et al., 2018a]. This strategy is called maximum entropy regularization. The intuition behind this strategy is that SAC tries to optimize its objective while acting as randomly as possible [Achiam, 2018].

SAC came out a year after PPO. According to the authors, its development was motivated by the sample inefficiency in state-of-the-art on-policy methods such as PPO and their sensitivity to hyperparameters [Haarnoja et al., 2018a].

## 2.3 Some related works

There has been a lot going on in music-related machine learning research over the last years. This section briefly describes a selection of related works that intersect the domains of machine learning and music, with a particular focus on reinforcement learning.

### Teaching an agent with human feedback

A suite of works carried out by Scurto, Bevilacqua, Caramiaux, and Van Kerrebroeck from 2018-2021 have investigated reinforcement learning for musical interaction [Scurto et al., 2021][Scurto et al., 2018][Van Kerrebroeck et al., 2018]. The main idea in their works is to include some form of human feedback in the learning loop of the reinforcement learning agent, for the agent to reinforce the desired behavior of a human. In one of the

papers [Scurto et al., 2021], the authors implement and perform user testing on a software that allows musicians to co-explore sound parameter spaces together with a reinforcement learning agent. The human user of the system interacts by shaping the parameter space and provide feedback on the suggestions of the agent. The user can also adjust the exploration behavior of the agent to let it explore less familiar parts of the parameter space. The different user interactions are assigned relative importance in the total reward function that is used to teach the agent how to proceed to explore the parameter space. The authors use Deep TAMER as a framework for interactive agent shaping, a framework that has been successful in leveraging interactive rewards for exploring high-dimensional state spaces [Warnell et al., 2018].

Visi extends Scurto et al.'s work to gesture-sound mappings [Visi, 2020]. A reinforcement learning agent proposes gesture-sound mappings from sensor data, e.g., accelerometers and Myo sensor armbands, to which the human user provides binary feedback. The feedback is then used to train the reinforcement learning agent. Visi lets the user first configure a number of synthesis parameters that act as an anchor for the agent. Then, the user tries out the default mapping and gives feedback to the agent. Iteratively, the user will refine their preferred mapping between the synthesis parameters and the gestures they produce. Visi writes that even though the binary feedback mechanisms are somewhat limiting to the training process, it functions as a creative tool for discovering interactions between gesture and sound.

Dahlstedt explored interactive artificial agents already in 2001 when he designed a system for parameter space exploration with interactive evolution [Dahlstedt, 2001a][Dahlstedt, 2001b]. His system lets users selectively evolve new combinations of sound parameters through controls in a user interface. The user interface allowed the user to control different aspects of the evolution process, such as combinations of sounds that should be further mutated and morphed in the next generation. Dahlstedt found that the system was successful as a compositional tool for exploring complex parameter spaces and structures within them in a rapid and simple way [Dahlstedt, 2001a].

The end goal of the cross-adaptive environment is to function as a tool for exploring new modes of musical interaction in live performances. Even though this thesis focuses on automatic mappings, a user of the system (a musician) would likely be interested in tuning the system to customize it for her musical expression. One way of letting a musician user interact with the model could be to further explore ideas about interactive agent shaping from the works of Scurto et al. [Scurto et al., 2021], Visi [Visi, 2020], and Dahlstedt [Dahlstedt, 2001a][Dahlstedt, 2001b]. By learning from how a musician interacts with the model, the musicians' role would be to *tame* the model for her specific

use case.

## The machine learning algorithm as a musical interface

Fiebrink et al. motivate an understanding of the machine learning algorithm as a human-computer interface, a mediator between the user's intentions and the computer's behavior [Fiebrink et al., 2016]. Similar to other human-computer interfaces, machine learning algorithms have their own set of affordances, such as the ability to represent structures in a dataset in the case of supervised machine learning. Affordance in this context refers to how an object, or a piece of software, can be used by a human actor. The term is attributed to the perceptual psychologist James J. Gibson [Gibson, 1977] and has been widely adopted in the human-computer interaction (HCI) community. General affordances of the machine learning algorithm as an interface is the ability to learn idiosyncratic style from whoever is interacting with the algorithm [Fiebrink et al., 2016]. Interactive machine learning involves the user of the algorithm in an interaction loop with the machine learning algorithm. Examples of interactions can be to iteratively define new data, tune relevant parameters or provide feedback to the model in the training process.

The usefulness of an interface is linked to its affordances [Fiebrink et al., 2016]. Fiebrink et al. focus mainly on affordances of supervised learning, e.g., the affordance of being able to build a model from examples instead of explicitly defining rule sets in code. Reinforcement learning is another machine learning branch, and has other affordances. Reinforcement learning agents learn from interaction with the environment, and as such, shaping the environment can be seen as an affordance of reinforcement learning. The previously mentioned work from Scurto et al. [Scurto et al., 2021] utilize another affordance of reinforcement learning; the ability to direct the teaching of the agent by incorporating human feedback as part of the reward.

## Symbolic music generation from audio features

RaveForce is a framework and a reinforcement learning environment for symbolic music generation based on evaluating features from an audio signal [Lan et al., 2019]. An agent is trained to imitate an audio sample of a drum break by deciding the correct steps in a MIDI sequence. A SuperCollider backend renders the symbolic representation to a new audio sample which is evaluated against the target. This process bears similarities with the cross-adaptive agent defined in this project, but the goals are different since RaveForce both use symbolic representation for the synthesis, and because the system is essentially

auto-adaptive instead of cross-adaptive.

## A live symbolic musical reinforcement learning agent

Collins developed a reinforcement learning agent to serve as a companion in improvised performances [Collins, 2008]. His goal was to build an agent for close and rewarding interactions during a concert performance. The resulting system was called Improvagent, a symbolic interactive system for MIDI piano improvisation. Improvagent learned from comparing successive frames of MIDI information such as onsets, note pitch, and velocity, and was partially rewarded based on its memetic success, i.e., how well the material it generated was picked up by the musician it was performing with. Collins was successful in developing a system for live improvisation with a reinforcement learning agent. He did not explore neural networks as function approximators, which he mentions as a drawback in the discussion. This might have been because neural networks were less popular in 2008 than they are today.

## Reinforcement learning as part of livecoding practice

Bernardo et al. implemented an interactive reinforcement learning agent as part of their live coding meta-language Sema [Bernardo et al., 2020a][Bernardo et al., 2020b]. The reinforcement learning agent is able to learn behavior from the person who is live coding, and is fully customazible for the user. Sema expects a relatively high degree of technical knowledge from its users in general, but interacting with the reinforcement learning agent in Sema is well integrated in the language. It is an innovative way of deploying a reinforcement learning agent.

## Reinforcement learning for speech enhancement

Fakoor et al. used reinforcement learning to enhance speech signals in phone calls [Fakoor et al., 2017]. They trained a reinforcement learning agent by using signal-to-noise measurements as a reward, and adapting the parameters of a noise-suppression algorithm by a reinforcement learning policy. This work is a form of auto-adaptive processing as defined in Verfaille et al. [Verfaille et al., 2006]. This demonstrates that adaptive audio effects also have applications outside the music domain.

## 2.4 Summary

This section has gone through relevant background material on cross-adaptive processing and reinforcement learning. These are the two fundamental areas of research that have made this thesis possible. The reader should by now be familiar with essential concepts for understanding the rest of the thesis, most notably how it is possible to model a simplified cross-adaptive processing scenario as a reinforcement learning environment. A small selection of related works in music-related machine learning has been briefly described to provide some more context.

# Chapter 3

# Methods

The research question of this thesis was to study how a simplified version of a cross-adaptive performance scenario can be modeled as a reinforcement learning environment with the goal of creating mappings in cross-adaptive audio effects. Finding answers to this question require software that models a cross-adaptive audio processing scenario as a reinforcement learning environment. To the author's knowledge, no such environment already exists for a similar use case, so it was deemed necessary to develop it as part of the thesis. Both Lan's Raveforce [Lan et al., 2019] and Collins' Improvagent [Collins, 2008] are previous works that have modeled musical agents with reinforcement learning, but for very different musical applications. The development of the cross-adaptive reinforcement learning environment is a large part of the effort gone into this thesis. As the development of the environment is integral to the research process, the research method has been evaluation through prototyping.

Even though a significant portion of the workload behind this thesis has been to develop software, it is not a typical software engineering project. The software itself is not an object of study and mainly serves to produce results used to reflect on the research topic. Hence, common evaluation criteria from software engineering, such as usability or stability, are not as relevant to measure the success of this thesis. The cross-adaptive environment developed as part of this thesis was implemented to examine the research topic from different angles. It would likely need to be redesigned from a user's perspective if it were to be used directly in a performance scenario. A possible continuation of this thesis could be to integrate a reinforcement learning agent into digital audio workstations (DAW). This continuation would resemble a typical software engineering project in which the topic of research could be the human-computer interaction between musicians and the software. Such a project would likely benefit from a redesign, focusing more on the user

experience of using a reinforcement learning agent in cross-adaptive processing. However, for the purpose of evaluating the research question of this thesis, user testing was a bit out of scope, considering the early stage of the project.

## 3.1 Evaluation

A series of test cases were developed to evaluate the research question. The test cases are meant to represent the capabilities and limitations of the software at its current stage. The test cases produced results in the form of figures and audio samples. The results have been evaluated with a hybrid of qualitative and quantitative observational methods. The reason for that is because a complete evaluation of the research question is twofold. On one side, evaluating the performance of a reinforcement learning agent is a matter of measuring the expected return of the agent's policy after training. The better the reward, the better the agent. This does, however, assume that factors such as the reward function or feature extractors are kept constant. On the other side, evaluating the successfulness of a given method for creating mappings for cross-adaptive audio effects is not as easily quantifiable. Cross-adaptive audio effects are applied in various musical contexts, in which the rules for good and bad performance often are more complex than a numerical signal can represent. The near-infinite range of audio effects also makes it difficult to evaluate performance numerically on a general level. Some effects are better at shaping the harmonic content in sounds, and others are better at accentuating transients. The theoretical processing capabilities of an audio effect with many parameters applied on a given audio signal are complex to establish quantitatively due to the vast amount of combinations of effect parameters. However, a trained human ear may use background knowledge of the audio effect to create heuristics that allow for qualitative evaluation, i.e., determining if the agent should be able to perform better or worse than it currently is doing. Besides, the dependence that the results have on both feature extraction methods and the choice of audio effect makes it difficult to make general quantitative comparisons between cross-adaptive scenarios. Even in the ideal example of a universal audio effect with the capabilities of shaping any sound into any other, the agent would not be able to learn interesting cross-adaptive mappings if the feature extraction was poor or consisted of features unable to capture the relevant nuances of the target sound.

Some of the test cases were set up as controlled experiments, in which there was only one independent variable. The controlled experiments use metrics from the training process as success criteria, such as mean reward. In supervised machine learning, the convergence of the loss is typically used as a signal to measure the model's performance. However,

in reinforcement learning, the loss function does not necessarily relate to the model's performance. This is partly because the data distribution is dependent on the most recent version of the policy, contrary to supervised learning where the data distribution usually is fixed before the training starts [Achiam, 2018].

For the four first test cases, those that compare different models to one another, three different trials were run with different random seeds. Since there is a lot of randomness in how policies are initialized, e.g. the weights of the ANN, random seeds can potentially play a big role in the performance of the algorithms and thus reduce the certainty of the results. Running three separate trials was an attempt to filter out potential outliers and thus increase the validity of the results. The most representative models among the three trials were selected and used for the comparisons. The judgement of which models were most representable was done qualitatively by inspection of the mean reward graph and trying to determine which model seemed to be closest to the average among the three. Henderson et al.'s comparison of various state-of-the-art reinforcement algorithms found that in several of the papers they examined, results were reported with few trials in environments in which performance results were prone to variations in random seeds [Henderson et al., 2018]. Even though it is possible that three trials may have been too few regarding the results in the cross-adaptive environment, none of the models in the trials produced any extreme outliers, which is a positive sign.

## 3.2 Constraints of the test cases

Cross-adaptive processing is a broad domain when considering all the possible combinations of effects and feature extraction methods. Since the emphasis of this thesis was to investigate reinforcement learning in cross-adaptive processing, audio effects and feature extraction methods were mainly reused from previous projects in the cross-adaptive research project, most notably from Brandtsegg's toolkit [Brandtsegg, 2015]. After initial tests with various audio effects, the one chosen to use in the test cases was a resonant low-pass filter with additional distortion and post-gain controls. Jordal was also able to produce interesting results with that effect in his thesis [Jordal, 2017].

The source and target sounds were kept constant during training. The source sound is a recording of white noise, and the target sound is a variation of the famous drum break known as the "Amen break"[1]. Both samples were trimmed to last precisely five seconds.

[1]The Amen break is a drum break performed by Gregory Coleman in 1969 that has been widely sampled in hip hop and electronic music

Six features were extracted from the audio signals in total: RMS, estimated pitch, spectral centroid, spectral spread, spectral flatness, and spectral flux. RMS is a direct measurement of the audio signal's amplitude, which is perceived as the loudness in the output signal for the human ear. Estimation of pitch tries to find the fundamental frequency in the audio signal. Many frames of audio signals will not have a defined pitch, so this feature's signal-to-noise ratio is highest for harmonic sounds. The spectral centroid relates to the perceived brightness of the sound. Spectral spread is associated with the amount of deviation from the spectral centroid. Spectral flatness describes the tone-to-noise balance of an audio signal. White noise has a very high value of spectral flatness since there are no distinct peaks in its spectrum. Conversely, a pure sine wave has a very low spectral flatness. Spectral flux is a measure of the rate of change in the spectrum by looking at the difference between two consecutive audio frames.

These six features can not fully represent the timbre of any given audio sample, but they give insight into some basic perceptual characteristics of the sound. There are many more features that could be added, and possibly some that may be deemed redundant. For example, the spectral flatness will have some similarity to the spectral spread, in that they both respond to broader energy distribution in the spectrum. Similarly, spectral flatness and spectral flux are overlapping, since they both respond to perceived noisiness of the sound.

Since the possibilities in cross-adaptive processing are so many, the selection of target and source sounds, the feature extraction methods, and the audio effect bear the risk of biasing the results. For example, white noise, which is used as a source sound, has equal intensity for all frequencies, giving it the property of being a highly shapeable sound. Even though these biases are unfortunate for the generalizability of the results, they are an inevitable part of the domain of cross-adaptive processing.

# Chapter 4

# Implementation

## 4.1 Functional requirements

One challenge in designing a machine learning system for real-time use is that the requirements are different during training and during performances. When training the model, it is practical that the reinforcement learning loop is synchronous since that both makes debugging easier and allows the training process to happen as fast as possible for the respective machine(s) running the process. In the digital audio context, an implication of this synchronous training procedure is that the rendering of the audio happens non-real-time, i.e., that it is not submitted to a time constraint. Therefore, *synchronous non-real-time* rendering implies that the audio used in the training process has to exist before launching the training process. Ideally, the system should also allow the user to train a model using real-time input, similar to the Wekinator [Fiebrink and Cook, 2010]. This would allow musicians to adapt the configurations of the system, such as the effect and the audio features, in real-time, which in turn would have additional affordances in a performance context. Supporting the training process in real-time proved to be a little too complex for the current prototype of the project but will be discussed in more detail later on. During performances, however, asynchronous calls to the machine learning module are desired as not to disturb the tight timing requirements of real-time audio processing routines. The ideal situation is that the performer's system can demand a new mapping from the machine learning module whenever it needs it. One way of achieving that is to use a mediator pattern. In the context of this cross-adaptive environment, the mediator is a shared interface between the signal processing module and the machine learning module, which provides a way of communicating with both ends asynchronously through the Open Sound Control (OSC) protocol. When the mediator receives new audio features,

it forwards those to the machine learning module. This initiates a forward pass in the machine learning module that generates a mapping for the audio effect. The mapping is sent back to the mediator over OSC, which relays the new effect parameters to the signal processing module. Since the audio features are rendered and transmitted in real-time over OSC, this process is referred to as *asynchronous real-time rendering*.

The cross-adaptive environment must be submitted to a time constraint to be employed by a musician in real-time. As soon as the mediator receives an observation from the audio processing system, it should initiate a forward pass of the neural network to produce the effect parameter mapping corresponding to the observation. Suppose the system is already processing an observation. In that case, the system should restart its computation with the new audio frame since it makes the most sense always to prioritize whatever is received latest, since that corresponds to the most recent audio frame of the performance. However, if the interval between successive observations is too small, it is possible that the system arrives in a state where it constantly throws out half-done calculations, never finishing processing any single audio frame. When that happens, every second frame (or more often, if necessary) should be discarded, effectively reducing the control sample rate of the machine learning module.

Running one forward pass of a neural network is a deterministic process since the number of operations is constant. This makes it possible to scale the size of the neural network to run a forward pass in real-time, respecting the system specifications of the computer it runs on. The disadvantage of reducing the size of the neural network is a potential loss in complexity. Hence, the optimization trade-off becomes latency vs. complexity.

## 4.2 Implementation of the cross-adaptive environment

The reinforcement learning environment was implemented according to the convention of OpenAI's *gym*[1]. *gym* is a library that provides standardized environments for comparing reinforcement learning algorithms. Since the inception of *gym* in 2016, research in reinforcement learning tends to use *gym* environments as benchmarks for measuring performance on various tasks. Examples of *gym* environments are classic control tasks, such as the cart-pole problem, or continuous physics tasks, such as teaching a two-legged humanoid to walk. Most of the environments used for benchmarking advances in the field of reinforcement learning research rely on some third party, such as pybullet[2] or

[1] `https://gym.openai.com/`
[2] `https://pybullet.org/`

MuJoCo[3]. Hence, the role of the *gym* library is to provide a uniform developer interface between reinforcement learning agents and reinforcement learning environments. *gym* enables developers to evaluate an agent's performance on many different benchmarking tasks without the need to change anything in their algorithm. However, the characteristics of certain environments typically require tuning the hyperparameters accordingly, e.g., learning rate and parameters related to exploration.

Due to the role that *gym* has obtained as a benchmarking tool in current research, libraries that implement reinforcement learning algorithms typically interface well with *gym* environments as well. Having *gym* as an interface between the agent and the environment is practical since it makes it relatively straightforward to deploy state-of-the-art algorithms to custom environments. In practice, developing a new environment requires the environment to inherit a base class in Python (`gym.Env`) to follow the basic steps of the reinforcement learning cycle (see fig. 2.3). Most importantly, the developer has to define a *gym*-compatible action space and observation space, as well as `step()` function that returns a reward and the subsequent observation for the environment.

### 4.2.1 The cross-adaptive learning loop

A step at time $t$ in the custom *cross-adaptive environment* works according to the following pseudocode, where $S$ is the source sound (dry), $T$ is the target sound, and $P$ (wet) is the sound processed by the effect $E$. $F$ denotes features, $s$ is state, $a$ is action, and $\pi_\theta$ is the parametrized policy of the agent.

**def** `step`$(S_t, T_t)$**:**
  $F_{S_t} \leftarrow$ `Analyze`$(S_t)$;
  $F_{T_t} \leftarrow$ `Analyze`$(T_t)$;
  $s_t \leftarrow [F_{S_t}, F_{T_t}]$;
  $a_t \leftarrow \pi_\theta(s_t)$;
  $P_t \leftarrow E_{a_t}(S_t)$;
  $F_{P_t} \leftarrow$ `Analyze`$(P_t)$;
  $r_t \leftarrow$ `Reward`$(F_{S_t}, F_{T_t}, F_{P_t})$;
  **return** $r_t$;

At each time step $t$, one audio frame is rendered from the unprocessed (dry) source audio signal and the target audio signal. Audio features corresponding to that frame are extracted. Together, the features from both the source and the target make up the

[3] `http://www.mujoco.org/`

observation that the agent receives. The agent's policy uses the observation to produce an action, i.e., a mapping from features to effect parameters. This action is applied to render the processed (wet) counterpart of the source audio frame. Audio features are then extracted from the processed source audio frame. The audio features from the unprocessed audio frame, the target audio frame, and the processed audio frame at time $t$ are then used to calculate the reward. However, which parts of the observations are used in the reward calculation depend on the reward function. In total, this constitutes a step in the cross-adaptive reinforcement learning environment. The reward is used to update the policy, as described in the background on policy gradient optimization 2.2.2, but this is not part of the environment's step cycle, strictly speaking.

### 4.2.2 Digital signal processing

One of the goals of this project was to design a framework that was agnostic of the software used for digital signal processing (DSP). Consequently, it was decided to perform all DSP in a module separate from the machine learning backend, which could be replaced by another audio processing software of choice if desired. Even though Jordal argued that performing the feature extraction in Python is a faster option for static audio [Jordal, 2017], through libraries like Librosa[4] or Essentia[5], the advantage gained from letting an external music computing system do this is that it interfaces better with other types of audio processing software. Csound, which was used for DSP in this project, can easily be packaged in a Virtual Studio Technology (VST) plugin and then be used in a DAW. Additionally, music computing systems such as Csound or SuperCollider are primarily used by people who work in the music technology domain. In that sense, using a music computing system to do DSP brings the project closer to the territory in which potential users of the system are likely to be familiar.

Brandtsegg developed *featexmod*[6] as part of the cross-adaptive research project. *featexmod* is a package and a toolkit for cross-adaptive processing. It enables analyzing features and using them as control signals in other processes. The toolkit is a revision of the previously released *interprocessing* which is well described in his 2015 paper [Brandtsegg, 2015]. *featexmod* was developed in Csound, which also has been the language of choice for the entire cross-adaptive research project. Integrating the toolkit in this reinforcement learning framework was a nice way of leveraging advances in the research project and using an audio processing software that is widely used both by music technology researchers

[4] `https://librosa.org/`
[5] `https://essentia.upf.edu/`
[6] `https://github.com/Oeyvind/featexmod`

and musicians. Various components from the toolkit have been graciously borrowed to perform analysis of audio features.

Csound has convenient bindings for Python through *ctypes*, a library for binding functions from external languages to Python code. The main advantage of using the Python bindings for Csound versus controlling it as an external process is that it enables rendering audio frame by frame from within the Python program. Being able to render and analyzing one frame at a time allowed evaluating the model every timestep. Another advantage with the Python bindings is that they enable the Python process to read and write to Csound's control channels synchronously. These channels are used internally in Csound to set effect parameters and as communication buses between Python and Csound. When considering the model for live real-time interaction, the synchronous channel-based communication is replaced by OSC messages sent and received asynchronously of the machine learning process.

### 4.2.3 Reward function

A reinforcement learning agent learns from rewards. The agent tries to reinforce actions that lead to positive rewards and prevent reproducing actions that lead to negative rewards. Rewards are calculated from a reward function. The reward function is a function of the current state of the environment. The state that is used to calculate the reward usually contains more information about the environment than what the agent is able to observe. For example, the rules of chess are ultimately what determine the reward a player would get by moving a piece, but an observation of the game only reveals the position of the current pieces. In the cross-adaptive environment, the agent only observes the two incoming audio signals, but also the processed audio signal is used to calculate the reward.

The design of the reward function is ultimately about what one wants the agent to achieve. In a cross-adaptive processing context, the desired outcome is interesting effect parameter mappings. What constitutes an interesting mapping is highly subjective, and it can also vary between scenarios. In order to be able to measure degrees of success, a reproducible and quantifiable goal has been used in the current work: to process the source audio in such a way that it has the greatest possible similarity with the target audio, using the same feature analysis methods for comparison as was used for training. To achieve this, the reward function has to be designed to reward the agent for taking actions that make the processed source audio sound more similar to the target audio than the original, unprocessed source audio does.

Similarity can be measured on the basis of the extracted audio features, where every feature is represented as a dimension in a vector. The most straightforward method for measuring similarity between two vectors is by taking the euclidean distance between them. This function takes two vectors $A$ and $B$ of equal length as input and outputs a number that says how far two points are from each other in an n-dimensional space:

$$EUC(A, B) = \sqrt{(A_1 - B_1)^2 + (A_2 - B_2)^2 + ... + (A_n - B_n)^2}$$

The shorter the distance, the more similar the features. Hence, the euclidean distance needs to be scaled in a way so that shorter distances lead to higher rewards and vice versa. In the results section, the following two ways of converting euclidean distance into a reward function are compared. The notation uses $R$ for reward function, end $EUC$ for euclidean distance. Dry and wet correspond to the unprocessed and processed signal, respectively.

**Inverse scale**

The first reward function is the simpest.

$$R_{INV}(wet, target) = \frac{1}{1 + EUC(wet, target)}$$

This gives a clean reward signal. When the euclidean distance is 0, the reward is 1, which is the maximum. The reward approaches zero as the distance becomes larger and larger.

**Relative gain**

The second method is motivated by the fact that the audio signals behave very differently over time. Some segments of the source audio signal will be easier to shape in a way that makes the processed signal sound similar to the target. This reward function uses the ratio of the euclidean distance between the dry signal and the target, and the euclidean distance between the wet signal and the target.

$$R_{REL}(dry, wet, target) = \frac{EUC(dry, target)}{EUC(wet, target)}$$

This function rewards the agent for coming up with a parameter combination that is able to do better than the unprocessed signal, and so measure the relative gain of applying the given parameter combination. Since the agent is able to control the mix parameter of the audio effect (the balance between dry and wet), it should theoretically be able to learn to avoid using the effect for segments where the dry signal already is similar to the target signal.

### 4.2.4 Signal flow

Figure 4.1 illustrates the signal flow of the implemented system. The upper half shows how the system works during training, and the bottom half during performance. The major difference is whether the communication happens synchronously through Csound's channels, or asynchronously over OSC. The agent works similarly in both cases: the agent observes the environment and uses its policy to find actions corresponding to the observations.

During training, rendering and feature extraction is performed synchronously. All frames are rendered through Csound's Python bindings, which enables setting effect parameters and extracting features through Csound's channel-based communication. Since all processes happen synchronously, the set of extracted features from the two signals arrive at the same time. Observations are then used to calculate the reward, which in turn is used to tune the weigths of the parametrized policy.

In performance mode, a pre-trained model is used. Since there is no learning happening at this point, there is no need to calculate rewards. For performance mode, it is assumed that features are extracted from two separate audio streams in real-time, i.e., two musicians have their own set of instrument plugins and a analyzer plugin at the end that is sending features over OSC to a mediator. The mediator's role is therefore to aggregate the features that are arriving asynchronously from the two audio streams, and forward them as complete observations to the machine learning model. Internally, the mediator quantizes the OSC messages since they may arrive with slightly different timing. The mediator is also in charge of receiving the action from the machine learning model, also over OSC, and to forward it to the source audio stream. Note that the target audio stream does not have to receive the action, since it is only the source that is going to be processed.

As a side note, this signal flow shows the way that the current prototype works. Ideally, one would allow users to train machine learning models in real-time, which would require

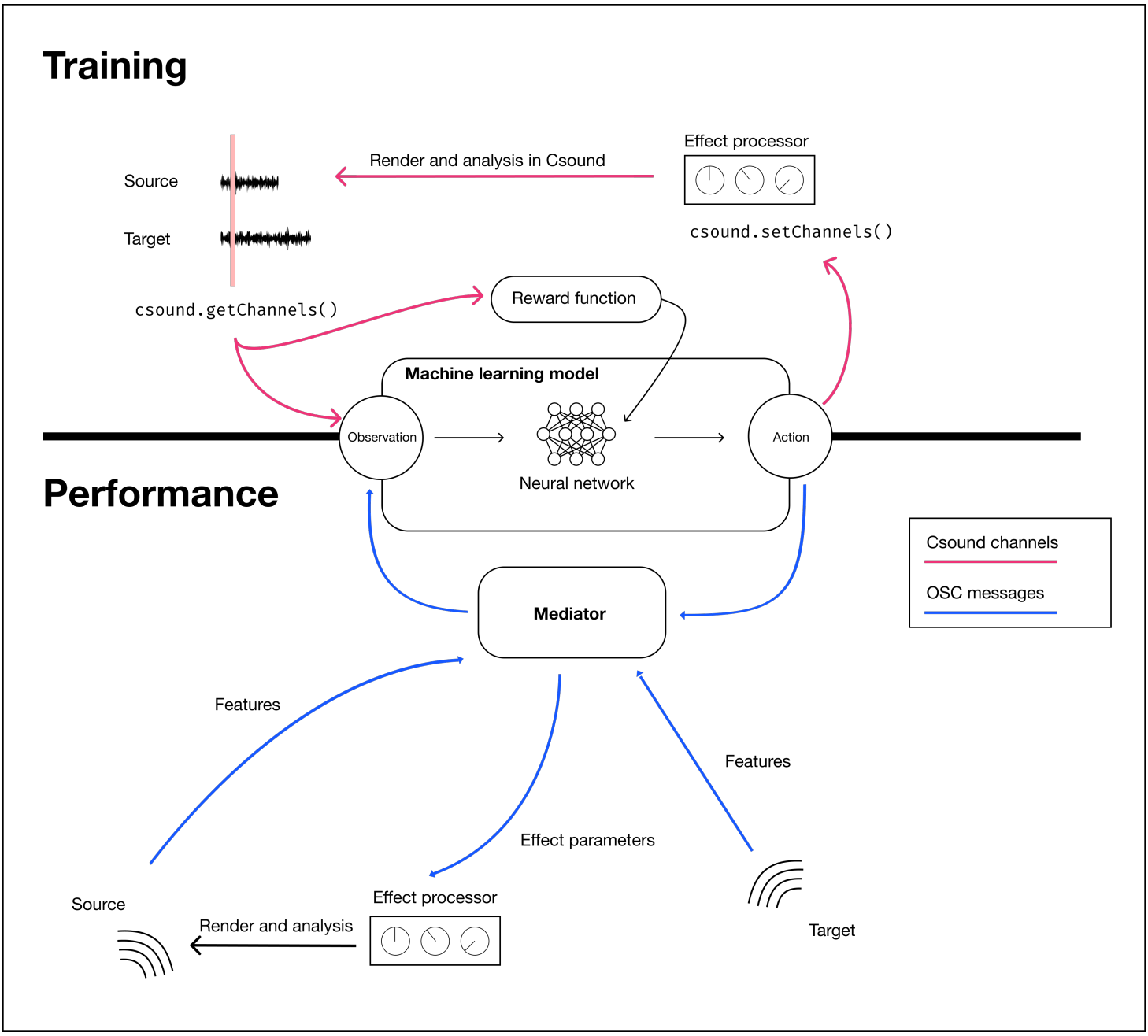



Figure 4.1: Signal flow during training and performance

revisions to the existing architecture. Real-time training will be elaborated upon in the discussion later on.

### 4.2.5 Standardization and normalization

The range of effect parameters varies largely, from 0 to 1 for the case of the mix parameter and from 20 to 10000 (Hz) for the cutoff of a low-pass filter. The activation function used in the neural network, the hyperbolic tangent function, outputs values within the range

of -1 to 1. To integrate smoothly with the activation function, the actions that are taken by the policy are scaled from [-1, 1] to the range of the respective effect parameter. The actions are then mapped by the following function:

$$f(x) = a + (b - a) * e^{\frac{log(\frac{x+1}{2})}{s}}$$

where $x$ is an action between -1 and 1, $[a, b]$ is the minimum and maximum range of the respective effect parameter, and $s$ is a *skew factor*. $\frac{(x+1)}{2}$ is done to shift from [-1, 1] to [0, 1] prior to scaling, since all effect parameter values are in the positive range.

Our ears perceive tones of 220, 440 and 880 Hz as linearly and evenly spaced notes in three octaves, although the relation between the three frequencies is non-linear. Consequently, some effect parameters, such as cutoff frequency, makes more sense to scale exponentially rather than linearly. The skew factor is a way to account for both linear and exponential parameter mappings from the neural network output. A skew factor of 1 corresponds to a linear mapping. A skew factor of 0.5 corresponds to an exponential mapping. This scaling idea was borrowed from Walsh's GUI framework for Csound, Cabbage [Walsh, 2008], and was also used in Jordal's project [Jordal, 2017].

All the audio features are normalized in Brandtsegg's toolkit. However, the mean and standard deviation vary largely from feature extractor to feature extractor and naturally depend on the audio samples. After the features are extracted from Csound, they are therefore standardized by subtracting the mean and dividing by the standard deviation. This helps the neural network learn faster [LeCun et al., 2012]. Since all observations are collected at runtime, the standardization is based on a running buffer of previous observations, from which the mean and standard deviation is calculated.

For convenient experimentation with different combinations of feature extractors and effects, Jordal's implementation of templating Csound files with Jinja[7] [Jordal, 2017] has been partially reused. Using a template engine is a fast and modular way of writing Csound files to disk from within a Python program.

### 4.2.6 Important hyperparameters

Hyperparameters play a large part in machine learning. Hyperparameters that are specific to the learning algorithm will be mentioned in the relevant part of the results section.

[7] `https://jinja.palletsprojects.com/`

Three general hyperparameters, learning rate, activation function, and the neural network size, were set the same for all test cases in the results section, and as such are part of the implementation.

### Learning rate

The learning rate of a neural network controls how much the model will change in relation to the reward that is given to the agent in every episode. Too low a learning rate may result in a model that never will be able to converge towards a local maximum. If the learning rate is set too high, the model might weigh every reward too much, and potentially skip important optima along the way that would yield better results.

Finding the correct learning rate often requires trial and error. Figure 4.2 shows a grid search on multiple learning rates for 1500 iterations[8]. Clearly, too low a learning rate prevents the model from learning. The green line shows the possible pitfalls of setting the learning rate set too high, as the performance worsens towards the end of the training. Among the four, $3e-4$ performed the best and was therefore used in all the test cases.


Figure 4.2: Results from testing multiple learning rates

Soft Actor-Critic (SAC), the algorithm used in this grid search, samples a number of episodes before it starts learning, which is why the graph is shifted by 100 iterations in figure 4.2. This is also true for subsequent SAC plots. SAC can do this because it is an off-policy algorithm, whereas Proximal Policy Optimization (PPO) cannot because it is on-policy.

[8]One iteration in the training procedure corresponds to 100 evaluations of successive observations because the batch size was set to 100. The audio frames are of length 128 samples, and the length of the source and target samples are each five seconds long, sampled at 44100 Hz. Thus, 1500 iterations corresponds to roughly 87 full evaluations of the audio files in their entirety by the following calculation: $(128 * 100 * 1500)/(44100 * 5) = 87.07$

### Activation function

In ANNs, the activation function controls how the input to the neurons gets activated, i.e., what output the neuron will produce. Non-linear activation functions are used to enable neural networks to be able to learn non-linear mappings between input and output. Jordal tested out multiple activation functions and found the sigmoid function to work best for his project [Jordal, 2017]. The sigmoid function is a non-linear activation whose curve looks like an "S", hence the name. Jordal used a type of sigmoid function called the logistic function with an output range of 0 to 1:

$$f(x) = \frac{e^x}{e^x + 1}$$

Another type of sigmoid function, the hyperbolic tangent function, has an output range of $-1$ to 1:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \tanh(x)$$

Some publications argue that the hyperbolic tangent function is more stable for learning in neural networks since it is centered around zero [LeCun et al., 2012]. For this reason, the hyperbolic tangent function was chosen for this project

### Neural network size

The size of the ANN was set in a similar way to the learning rate, by running multiple trials with different configurations of hidden layers and amount of nodes in these layers. The trials were constructed "bottom-up", by first training small neural networks and then iterating by increasing the size until it did not seem to yield better results. Increasing the size of the ANN increases the number of operations for every forward and backward pass of the ANN, thus increasing the time it takes to train and evaluate the ANN. The trade-off is that larger neural networks allow for more complex function mappings from input layer to output layer. Hence, finding the right neural network size is a balance of latency versus complexity.

After running multiple trials with various configurations of hidden layers and neurons per layer, one hidden layer of 16 neurons was deemed sufficient for the task at hand. The trials were all run with the same six features (RMS, pitch and four spectral features) and

with the distorted low-pass filter, which has six effect parameters. Thus, the total size of the ANN was $[6, 16, 6]$. This size was used to produce the results.

# Chapter 5

# Results

This chapter consists of six test cases which will be presented and evaluated. The test cases highlight the research's main findings in studying reinforcement learning as a method for creating mappings in adaptive audio effects. The results will be the basis for the discussion in chapter 6, which will reflect on the results in a broader context.

In all test cases, a model was trained with the objective of shaping a source signal, noise, into a target signal, a drum break with either of the reward functions defined in 4.2.3.

It is highly recommended to listen to the audio samples in the digital appendix[1] while reading about the results. The audio samples are placed under headers that correspond to the section headers in this chapter, but will also be referenced in parentheses as they are discussed.

## 5.1 Comparing Soft Actor-Critic and Proximal Policy Optimization

The choice of a reinforcement learning algorithm can have a high impact on obtained results. In this experiment, two modern policy optimization algorithms are compared, namely Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO). SAC and PPO have different strategies for learning a policy, as described in the background section 2.2.2. The two approaches have different affordances, which makes them suitable for comparison.

[1] `https://ulrikah.github.io/thesis/`

To attempt a fair comparison between the two algorithms, the common hyperparameters were fixed throughout the experiment. Since SAC and PPO work a bit differently, they both have their own set of algorithm-specific hyperparameters. The difference in hyperparameters makes the comparison prone to bias due to how the algorithm-specific hyperparameters were set. This bias hurts the generalizability of the results.

SAC and PPO differ in two major ways in terms of configurable hyperparameters specific to the respective algorithm. For one, SAC is an off-policy algorithm. In practice, this means that there are hyperparameters related to how SAC should sample from its sample bank of previous episodes (called a *replay buffer*). The hyperparameters related to SAC's replay buffer were set to the default in *rllib* as early informal testing validated the default values to be stable. As PPO's optimization happens on-policy, PPO does not operate with a replay buffer. The other major difference is the strategies that the two algorithms use to handle exploration in the environment. SAC uses the mentioned entropy regularizing approach (see section 2.2.4), where the objective is to maximize not only the expected return, but also the amount of entropy in the policy. The intuition behind this approach is that SAC always rewards the agent more for exploring the less familiar of two actions if they are rewarded the same by the environment. PPO also operates with a notion of entropy, but this entropy is related to how randomly PPO samples from its stochastic policy. Typically, the entropy is set high at the beginning of the training process to let the agent explore a lot, and then it decays by a factor throughout training. This strategy is very similar to the $\epsilon$-greedy exploration strategy that is often used in Q-learning.

Early informal testing showed that the cross-adaptive environment necessitated more exploration than the typical benchmark environments. Experiment 5.3 goes into detail on exploration in the cross-adaptive environment. For that reason, the hyperparameters related to exploration were set relatively high, with the intuition of letting the agent explore approximately four times as much as it would have with the default setting.

Table 5.1 lists all hyperparameters that were set for the comparison of SAC and PPO other than those that have already been described in section 4.2.6. To produce the results, two agents were trained for 1000 iterations with the respective algorithms and hyperparameters. The features that were used for training were RMS, pitch, and various spectral features. The following section shows that an agent equipped with these feature extractors can create mappings that validate reinforcement learning as a method for creating mappings in cross-adaptive audio effects.

Figure 5.2 shows the trajectory of the mean reward over 1000 iterations for two tuned models trained on SAC and PPO, respectively. The result shows that SAC outperforms PPO in the cross-adaptive environment, with a mean reward that is approximately 25%

| **Common** | |
|---|---|
| Optimizer | stochastic gradient descent |
| Batch size | 100 |
| **SAC** | |
| Target entropy | 24 |
| Replay buffer size | $10^6$ |
| **PPO** | |
| Entropy coefficient | $10^{-4}$ |

Figure 5.1: Hyperparameters

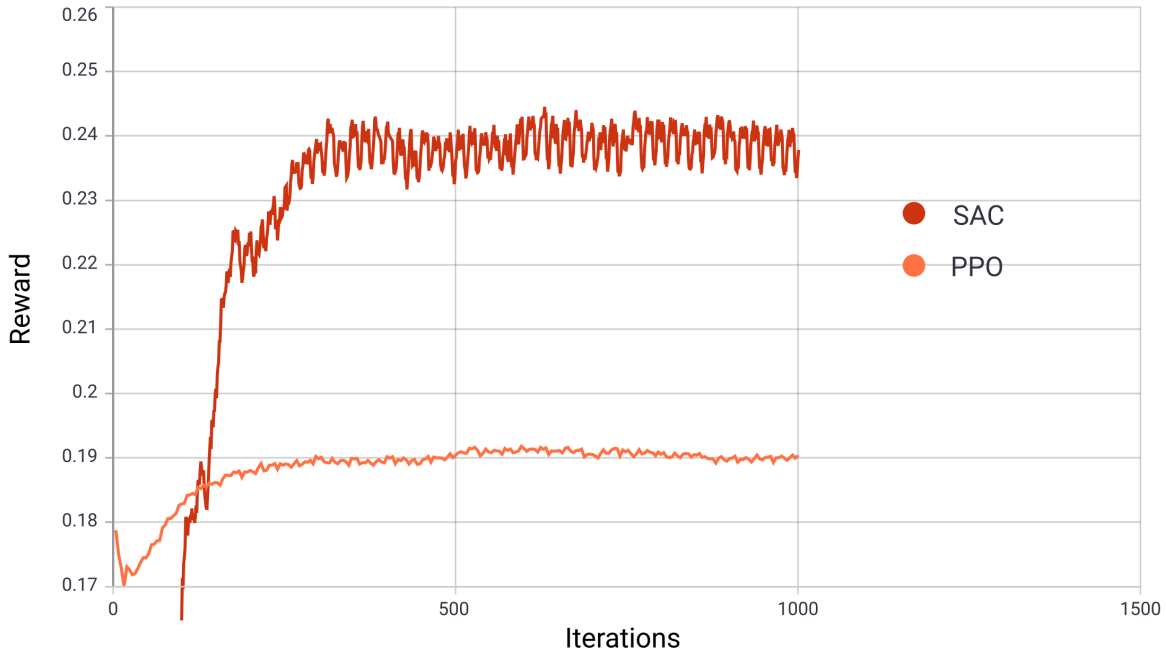



Figure 5.2: Mean reward - SAC vs. PPO

higher. As SAC was developed as a successor to PPO, partially to provide an algorithm that was less brittle to hyperparameters changes than existing ones [Haarnoja et al., 2018a], it is not very surprising that SAC performs better. The audio examples (1.1 and 1.2) in the digital appendix reflect the results from the training process.

The relative performance of reinforcement learning algorithms is not independent of the environment. This is exemplified in the original SAC paper from Haarnoja et al. [Haarnoja et al., 2018a], where the authors' implementation of SAC is compared to four other algorithms on six benchmark environments. The results show that SAC achieves a significantly higher average return on most of the environments, but for some (namely *Hopper* and *Ant*), SAC performs similarly to PPO. Keeping this in mind, there may be specific characteristics of the cross-adaptive environment that amplify SAC's success. For instance, the stochasticity of the cross-adaptive environment, i.e., the fact that it's near

impossible for the agent to predict the next observation, may be more suitable for SAC's entropy regularizing strategy than for PPO's exploration strategy.

If PPO is more brittle to hyperparameter changes than SAC, as stated in Haarnoja et al. [Haarnoja et al., 2018a], this opens up for the possibility that PPO's hyperparameters were set a bit out of tune in the comparison. A hypothetical perfect tuning of PPO's hyperparameters could potentially yield a better mean reward and close parts of the performance gap between the two algorithms. Still, a difference of approximately 25% is sufficient enough to declare SAC the winner of this comparison. Consequently, the following test cases all use SAC for policy optimization.

## 5.2 Feature selection

The role of feature selection is essential. It is the only representation the agent has of the environment and determines its learning abilities and the auditory output.

In the following experiment, two models have been trained for 1000 iterations with different sets of feature extractors. The choice of feature extractors defines the observation space of the agent. Hence, the hypothesis is that the models should behave differently with respect to the features they have been trained on. Figure 5.3 displays the control parameters over time for both models, aligned with the waveform generated after running inference on the models. The control parameters have been normalized from their respective ranges to $[0, 1]$. The control parameters are upsampled to audio rate in Csound[2] to align them with the audio samples. The target and source waveforms are displayed for reference.

The first model (model A, displayed with red color in the plots) uses only RMS to compare the two audio signals' similarity. The second model (model B, displayed with a blue color in the plots) uses estimated pitch and spectral features in addition to RMS for comparison. The spectral features that have been analyzed are the same features described in the methods chapter 3.2, i.e., spectral centroid, spectral spread, spectral flatness, spectral flux. The hyperparameters were set the same for both models, similar to the hyperparameters in the previous experiment.

#### Amplitude peaks

All of model A's parameters are closely related to the amplitude in the target waveform.

[2] `http://www.csounds.com/manual/html/upsamp.html`

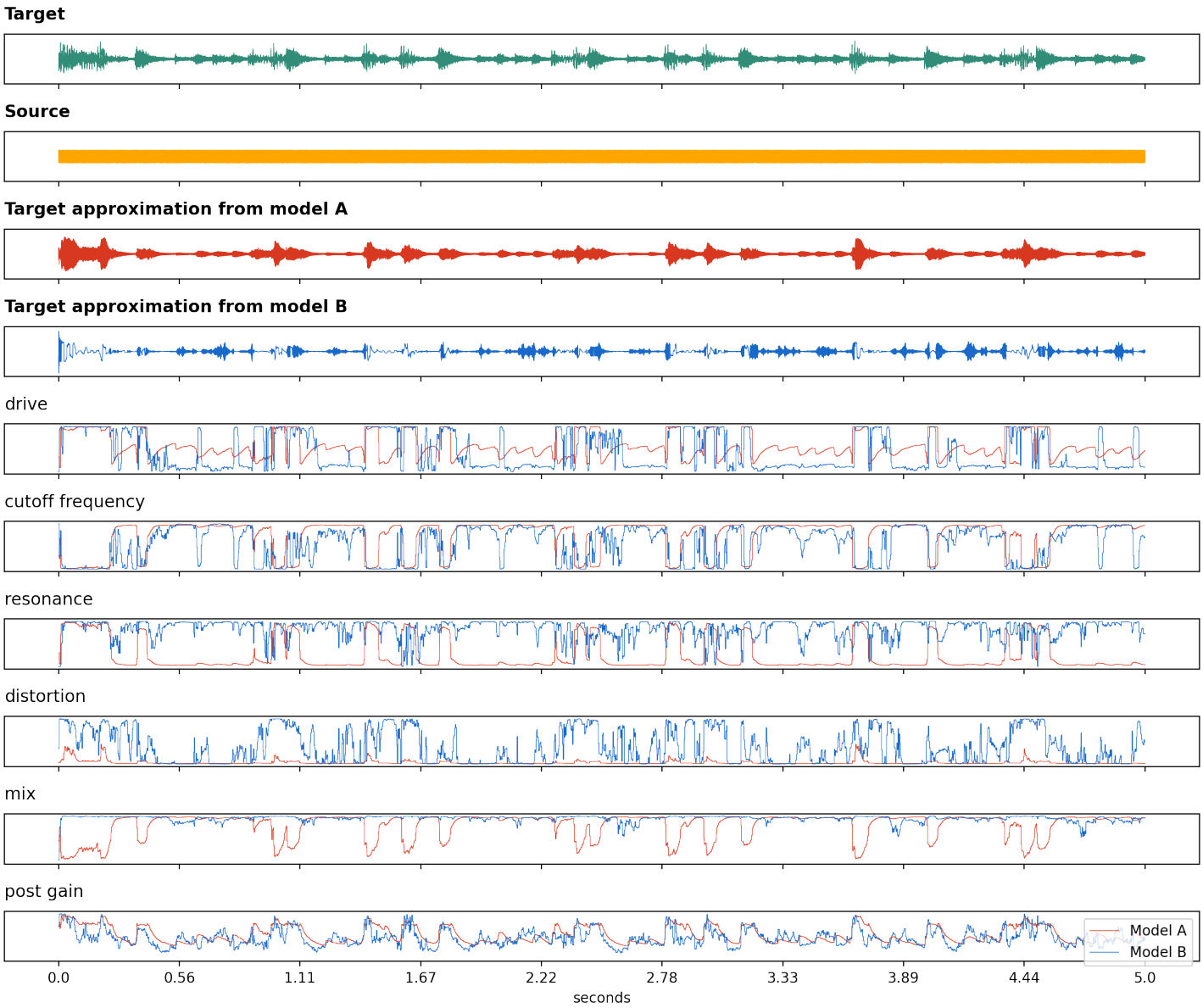



Figure 5.3: Control parameters of model A (RMS) and B (RMS, pitch, spectral)

Resonance, drive, distortion and post gain increases according to the amplitude, whereas the frequency and mix parameters are inversely correlated, i.e. they decrease as the amplitude goes up. This observation fits with model A's objective of optimizing for similarity in RMS values, as RMS extraction is a measurement of the absolute value of the amplitude of an audio signal. However, the sharp cuts in the mix parameter that happen at the same time as the amplitude peaks in the target signal, indicate that model A is mainly using the dry signal to represent the amplitude peaks, and the wet signal for the periods between the transients.

Comparing model B with model A, one notable difference is how model B sets the mix parameter for the rendering. Interestingly, the mix parameter in model B remains at a

constant high value for almost the whole period. This means that model B hardly uses any of the dry signal to represent the amplitude peaks, which conversely seems to be model A's strategy. By listening to the audio samples (2.1 and 2.2), it is clear that B's approximation to the target sound is more accurate. The contrasting control of the mix parameter shows that model B has learned to represent the amplitude peaks by making use of other effect parameters. This result is positive because the amplitude peaks correspond to different drum hits with varying timbral characteristics, and thus it makes sense that effect parameters have to be combined to represent them. Model A, in contrast, uses mainly the mix parameter to represent these peaks. The result is that it is difficult to differentiate the drum hits from each other.

Several of model B's parameters have a similar relation to the amplitude peaks as model A, but with many more nuances. Drive and frequency increase and decrease according to the amplitude peaks, respectively, and are inversely correlated. Some of the high-pitched transients (hi-hat hits) between the kick and the snare hits trigger peaks in the control signal of the low-pass filter cutoff frequency, effectively letting some of the high-frequency content come through.

Similar for both models are the slopes of the post gain parameter. The post gain parameter controls the gain (amplitude) of the audio signals after the sound has been processed by the low-pass filter. One interpretation from this observation is that both models learn that post gain is an effective parameter for affecting the RMS feature of the signal. The relation between the RMS feature and the post gain parameter is likely one of the more linearly connected mappings in the environment.

**Jitter in control parameters**

A notable difference between model A and model B in figure 5.3 is the amount of jitter in the control parameter values. More nuances in the control parameter signals are expected, given that model B has a more complex observation space than model A, but ideally not at the cost of more jitter. The goal is always to have a high signal-to-noise ratio as possible for all parameters. Figure 5.4 shows a close-up view of half a second of the parameter mappings of the post gain parameter. Even though the two waveforms follow a similar trajectory, model A's curve is smoother than model B's curve. Model B has a substantial amount of jitter in the parameter values, which may indicate that the model is too prone to subtle changes in the observations.

A possible explanation for the parameter jitter in model B is that it is more difficult to learn precise mappings for multiple features at the same time. The addition of the estimated pitch and spectral features increases the size of the observation space from

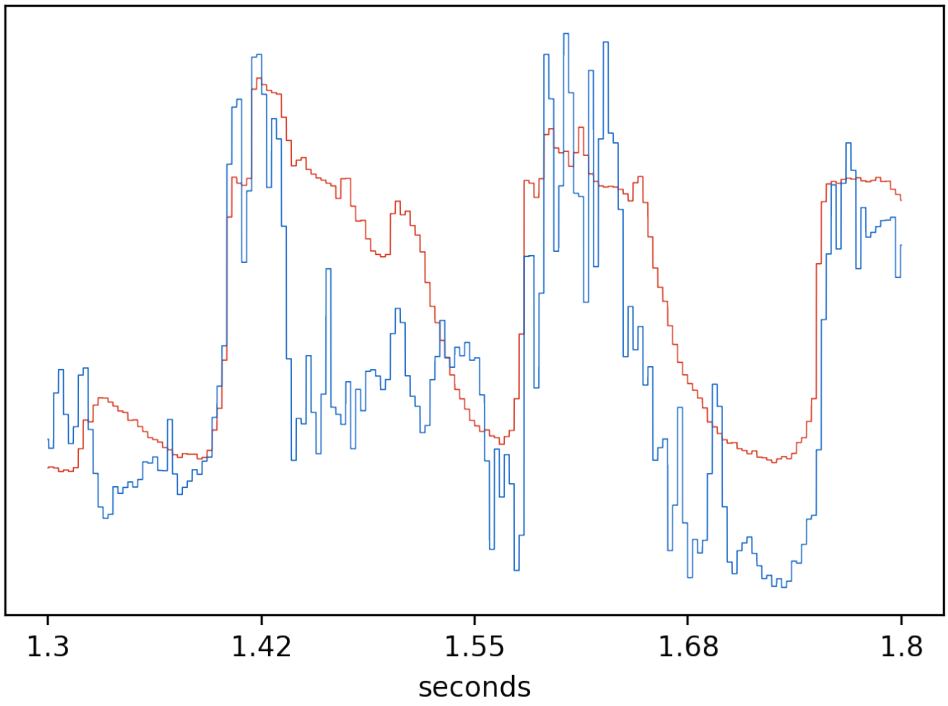



Figure 5.4: Jitter in post gain parameter

one single feature to six features (RMS, pitch, spectral centroid, spectral spread, spectral flatness, spectral flux). Mapping six features to six parameter values is a harder task than mapping one feature to six parameter values. However, neural networks have been able to do more complicated tasks in the past, so the jitter may also be a sign of a model that has converged to a non-optimal maximum. This may, in turn, be due to non-optimal hyperparameter configuration.

In a performance setting, some of this jitter could be smoothed out by interpolating between the output values of the model. However, such a smoothing only serves as a band-aid and comes at the cost of highly articulated effect parameters.

Note that there is no reward plot in this comparison of audio features. Since model A's observation space is only a subset of model B's observation space, the scale of the euclidean distance, which is used to calculate the reward, varies so much that it will be unfair to compare model A to model B. Consequently, it does not make sense to use reward directly as a signal for comparing the performance of model A and model B.

## 5.3 Balancing the exploration temperature

Finding a good trade-off between exploration and exploitation is a vital part of reinforcement learning. As environments vary considerably in shape and form, the strategy for finding this trade-off is empirical testing based on heuristics, as is the case for many other hyperparameters in machine learning.

The SAC algorithm uses an exploration strategy that aims to maximize the action selection entropy during training. As entropy is a measure of randomness in the policy, this strategy is essentially a way of optimizing a policy while acting as randomly as possible. In SAC, entropy is added as bonus in the expression for expected return:

$$J(\pi) = \underset{\tau \sim \pi}{E}[R(\tau) + \alpha H(\pi|\tau)]$$

$H(\pi|\tau)$ represents the entropy of the policy $\pi$ given the sequence of $(state, action)$ pairs $\tau$. The trade-off coefficient, $\alpha$, controls the relative weight of the entropy coefficient in the expression. Determining how much the bonus for maximizing entropy should be weighted relative to the environment reward is a matter of tuning as it may vary from environment to environment. This process of weighting the entropy maximization objective is referred to as tuning the *temperature* of the learning policy. In a preprint that was published as a follow-up to the original algorithm [Haarnoja et al., 2018b], the SAC authors introduced a mechanism that allows treating the desired entropy value, called the *target entropy*, as a constraint. This was an improvement to the previous version, in which the temperature was set to a static value, which made the algorithm less brittle to the temperature tuning. In practice, this means that the temperature of the entropy maximization can vary between episodes while approaching a target entropy, instead of being fixed at a constant value. Haarnoja et al. explain that the intuitive improvement of this mechanism is that the agent will explore more in parts of the action space where the policy is less optimal and exploit more in regions where the policy is more optimal [Haarnoja et al., 2018b].

The official implementation of SAC[3] sets the default target entropy to the size of the action space. In the cross-adaptive environment, the size of the action space corresponds to the number of parameters in the given audio effect. The distorted low-pass filter used in this experiment has six effect parameters, which yields a default target entropy of $\alpha_D = |A| = 6$.

[3] `https://github.com/rail-berkeley/softlearning`

An experiment was carried to study the exploration temperature in the cross-adaptive environment. Five models with different magnitudes for target entropy were trained for 1000 iterations. For the sake of reference, the models will be labeled *C*, *D*, *E*, *F* and *G*. The target entropy values were $\{3, 6, 12, 24, 48\}$ for model *C*, *D*, *E*, *F* and *G*, respectively. However, the interesting aspect to compare is not the numerical value of the target entropy directly, but the relative difference between them. These values correspond to $\{\frac{1}{2}\alpha_D, \alpha_D, 2\alpha_D, 4\alpha_D, 8\alpha_D\}$ where $\alpha_D$ is the default value for target entropy following the heuristic in the original implementation. The interpretation of an agent being trained with a target entropy value of $4\alpha_D$ is thus that it explores four times as much as an agent being trained with a target entropy value of $\alpha_D$. Figure 5.6 shows their target approximations after training for 1000 iterations and 5.5 shows a plot of the control parameters over time for all five models.

This section contains many graphs and models, so it is advised to use the appendix actively while reading. The relevant audio samples are found in the digital appendix (3.1, 3.2, 3.3, 3.4, and 3.5).

A qualitative evaluation was done by listening to the audio samples and comparing them to the target. The purpose of the evaluation was not to define the objectively best performing model, but to serve as a supplement to the numerical plots when comparing the models. The evaluation resulted in the following rating of the models relative to each other:

1. Model F
2. Model C
3. Model G
4. Model E
5. Model D

The most apparent result from figure 5.5 is how model F, with the next to highest target entropy value, uses only two out of the six effect parameters to approximate the target sound: post gain and filter frequency. Model F keeps the drive and mix parameters at a constant high and the distortion and resonance parameters at a steady low. All the other models have learned to use all effect parameters to create the same mappings. Model F has a mean reward that is approximately 10-20 % higher than the other models, as seen in figure 5.7. From the listening test, model F is perceptually closest to the target,

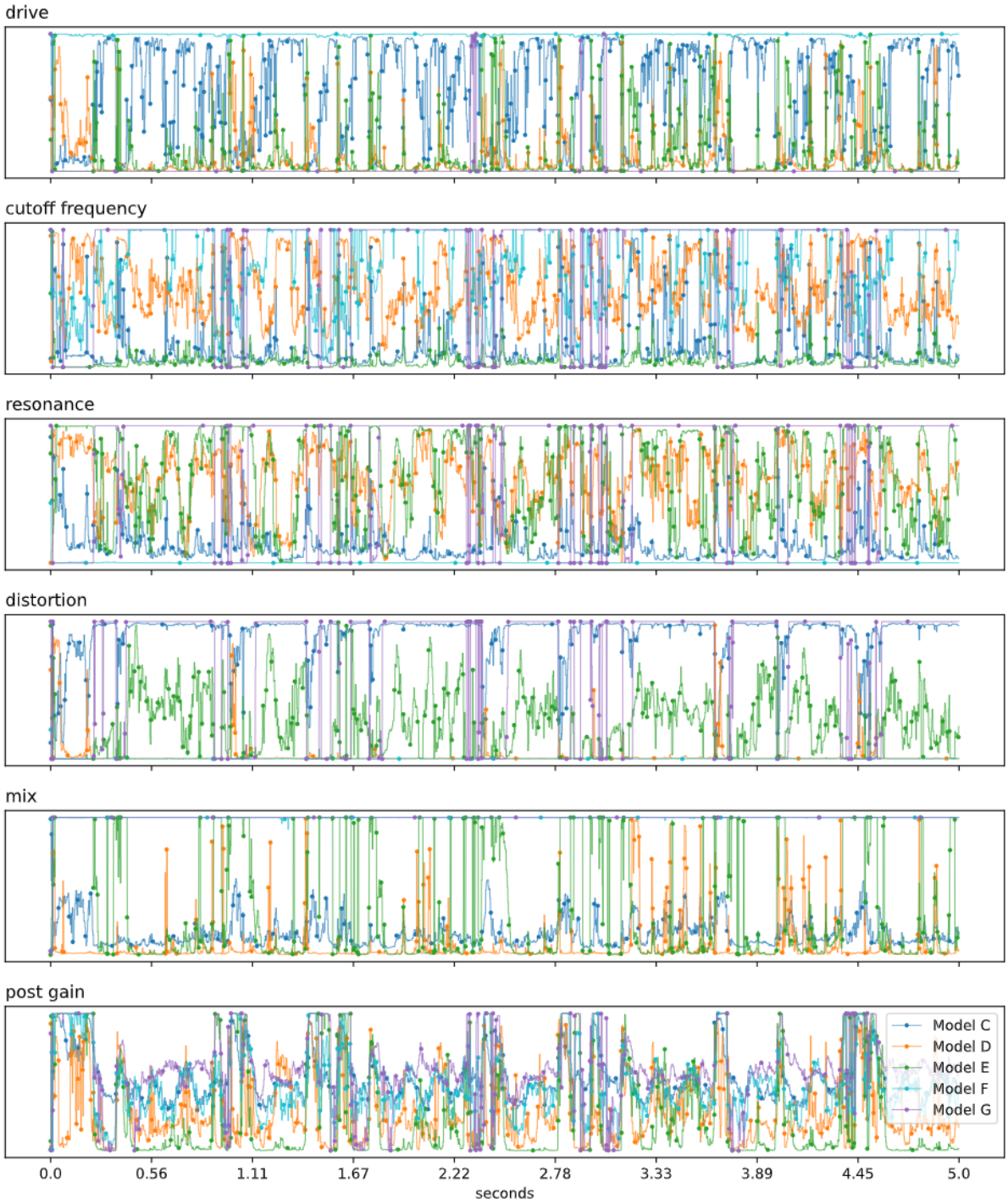



Figure 5.5: Control parameters of five models with different target entropies.

particularly when it comes to the low-frequency content in some of the transients (kick and snare hits).

Interestingly, the auditory output of models C, D, E, and G all have unique characters. Model C, performing close to worst in terms of mean reward, shares more characteristics with model A from the feature selection experiment (5.2), which was trained only on RMS, than with the other models in this experiment. Model D manages to capture

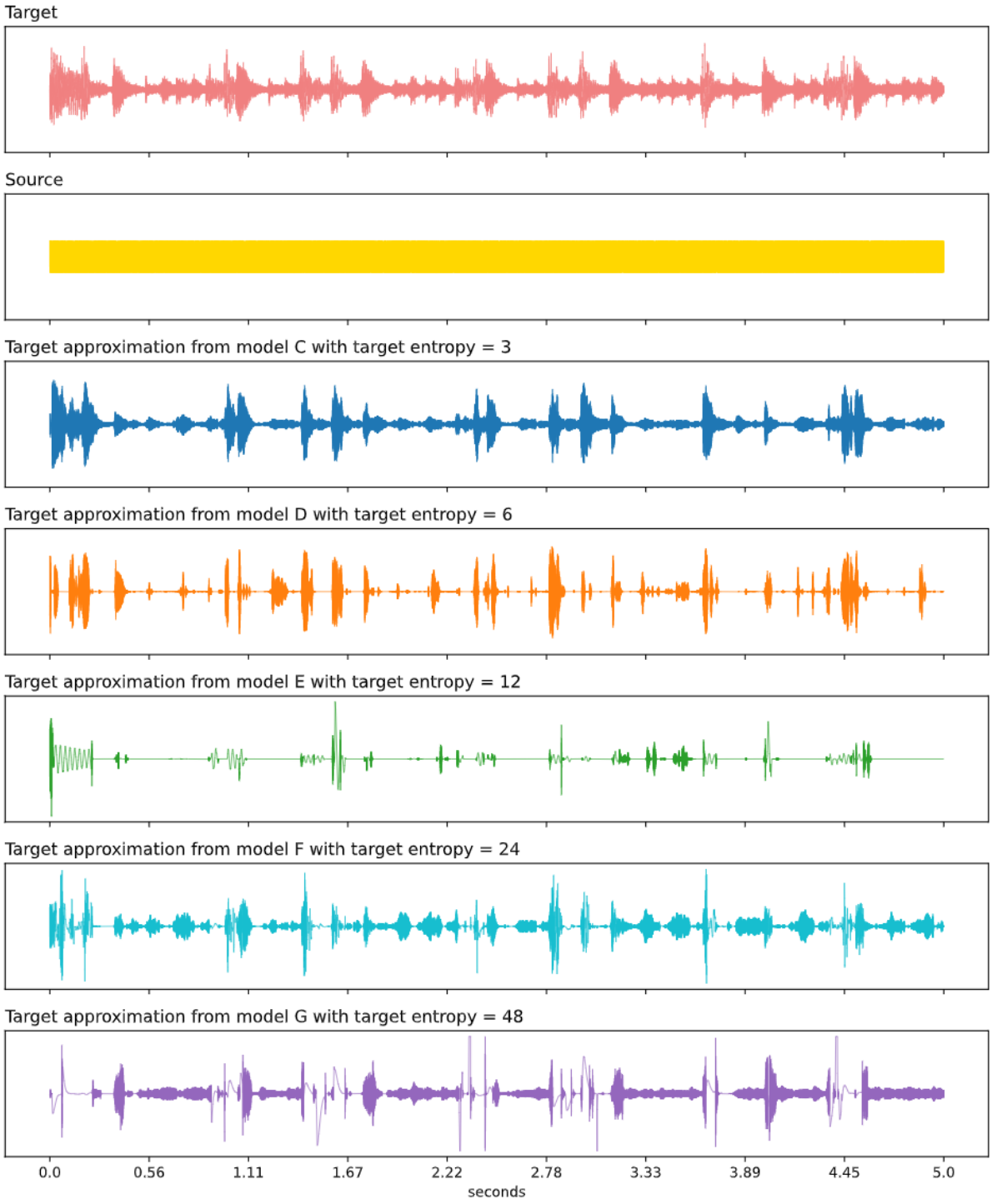



Figure 5.6: Target approximations of five models with different target entropies.

the transients' structure in the target sound but fails to approximate the variances in the frequency content between the high-pitched hi-hat hits and the low-pitched kick and snare hits. The result makes it difficult to separate the various drums from one another in model D's output. The shape of the frequency parameter in model D is less distinct than it is for model F. Except for the post gain control, model F only uses the frequency parameter to shape the source sound. Since one would not expect a gain control to affect

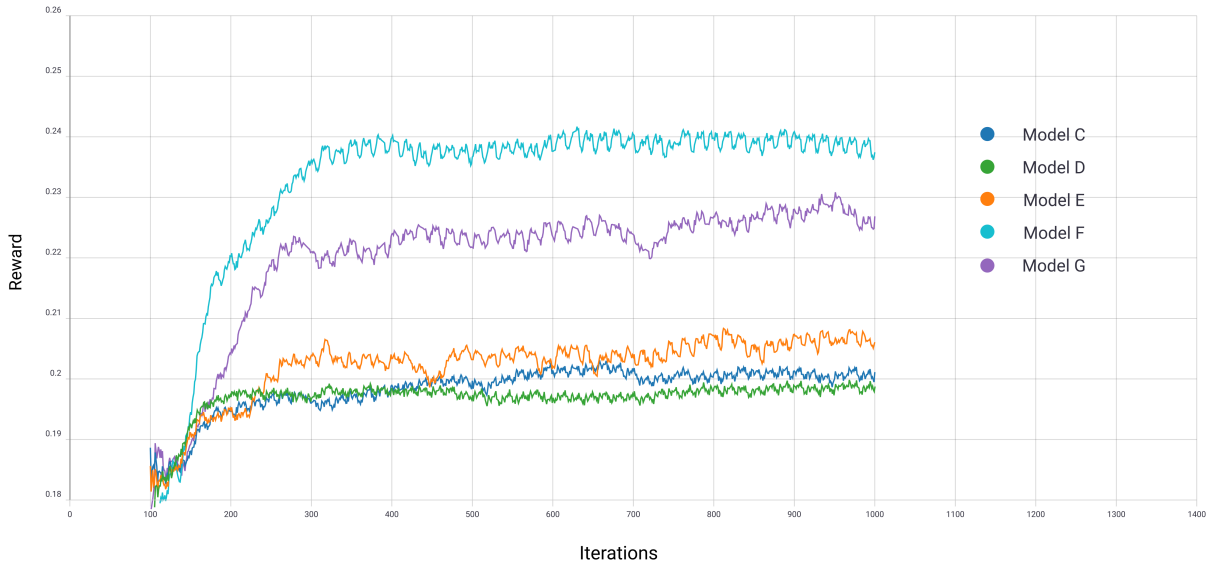



Figure 5.7: Mean reward over 1000 iterations

spectral features at all, it is possible that making efficient use of the frequency parameter is what makes model F perform so much better than the other models.

Model E's performance is the most peculiar among the four models. Listening to the output of model E, it seems like it does the opposite of model C. Model E achieves better at attaining the frequency content of the transient hits, but the transients are less pointed out. Figure 5.9d shows several periods where the spectral flux is very low in model E compared to the other models. The spectral flux is a measure of how quickly the spectrum changes over time. The static periods in the spectral flux plot may signify that model E has not learned the spectral variations as well as the other models.

Model E seems to represent the transients by rumbling low-frequency energy bursts. Human ears generally perceive low-frequency sounds as less loud than mid-to-high-frequency sounds [Suzuki and Takeshima, 2004], whereas the spectral feature extractors used to train these models do not take the human auditory system into account. This may explain why model E achieves higher mean reward scores than model C and D, but perceptually is further away from the target sound. Using feature extraction methods that account for the human auditory system, such as the mel scale or the bark scale, could potentially mitigate such a bias.

Even though the qualitative evaluation of model G happens to be rated in the midst of the models, the output from model G has some characteristics that none of the other models have produced. Model G manages to better separate some of the high-frequency content in the drum hits, such as the difference between how the snare and the hi-hats sound. The snare hits are particularly well accentuated compared to the snare hits produced by

the other models. Digital snare hits can be produced by sending a noise burst through a linearly shaped envelope, and one could expect a well-performing neural network to be able to learn this maneuver. However, only model G seems to have learned a way of synthesizing a snare hit, likely due to a high number of explorations.

Inspecting the mean reward plots in figure 5.7, the mean reward of model G is still increasing after the 1000 iterations for which all models were trained. The training of model G may have been ended before it started to converge properly. The most promising model G from the three trials was therefore trained for 1500 more iterations in the hope of further improvement. Even though the model continued to improve slightly for a little while, it ultimately did not influence the output sound too much.

Only a few of the feature plots (see figure 5.9a-5.9f) are good indicators of the model performances. Several of them change rapidly over time and contain so much jitter that it is difficult to conclude much based on them. In the spectral flatness plot in figure 5.9e, several of the models have periods of "clipped" values. The same behavior is somewhat apparent for spectral centroid and spectral spread as well. However, the better-performing models are more similar to the target feature plot than the worse-performing models, which is a good sign. The "clipping behavior" is more understandable when looking at figure 5.8, which shows the feature plots of the source signal compared to the target signal. The mismatch between the feature plots and the qualitative evaluation of the sounds speaks to a broader problem: there is a gap between the way humans perceive sounds and the individual feature extractors we use to analyze them. Furthermore, this gap makes it non-trivial to precisely describe what we want our machine learning models to achieve. Such a description ultimately boils down to some form of numerical comparison between audio features, which may not always be as close to the way humans compare sounds as desirable.

It is noteworthy that the curves for some parameters show a lot of activity near the extremes of their range (see, for instance, model F and G's distortion parameters in figure 5.5). This activity may indicate that the respective models have failed to learn the effect of using that particular parameter, but the exact rationale for this behavior is uncertain and is an area for improvement in future work.

## Summary

Overall, the results from this experiment show that the cross-adaptive environment required more exploration than what is suggested by the heuristic of the SAC authors. A

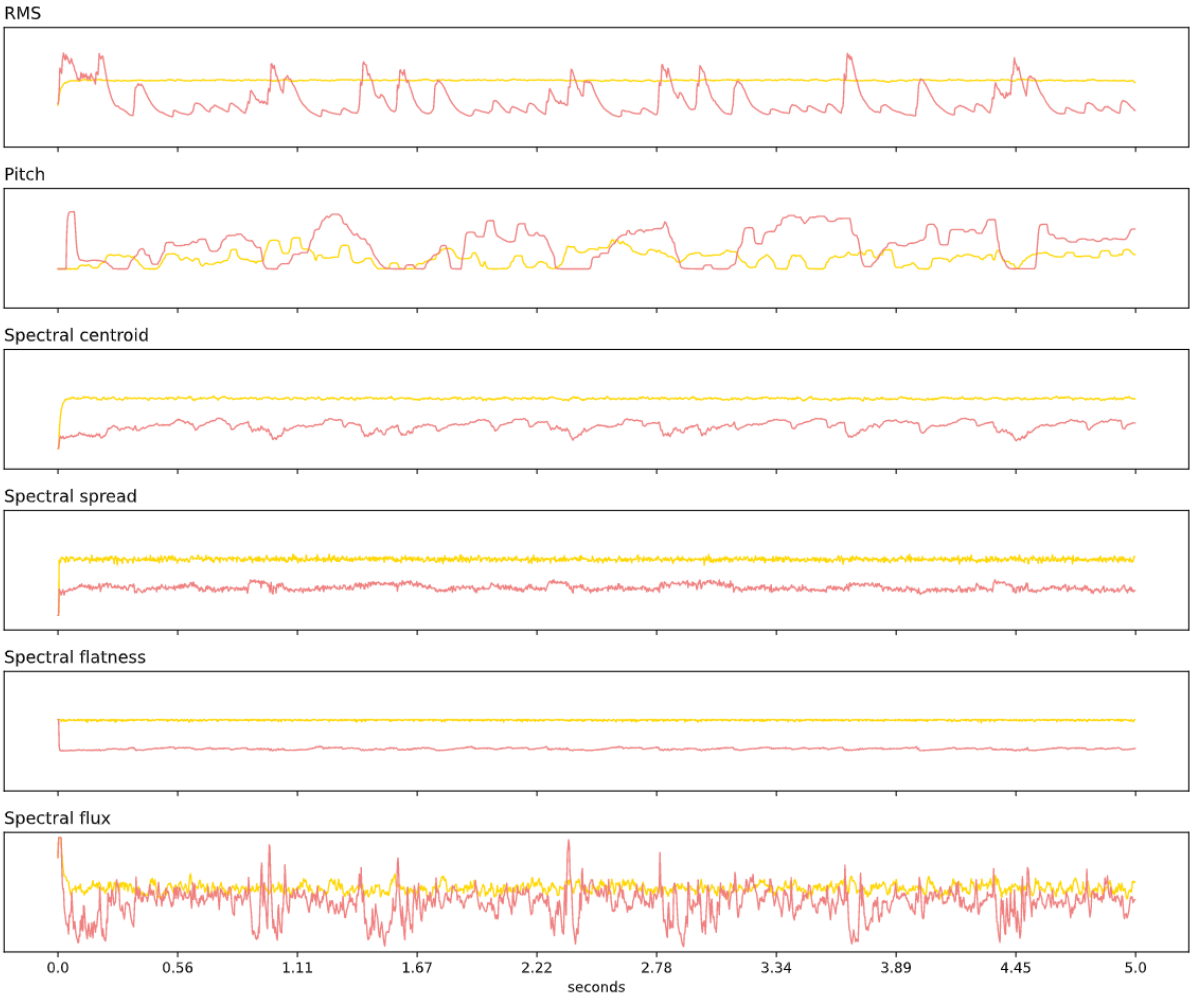
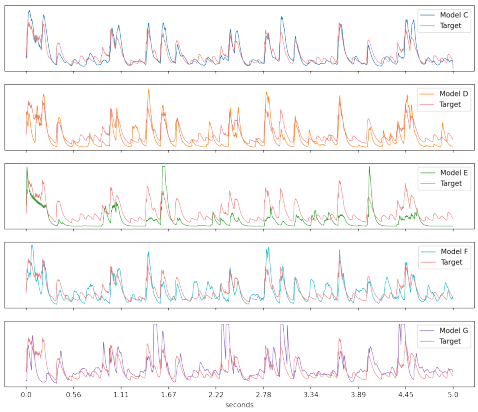



Figure 5.8: Feature plots for the source (yellow) and target (red) audio

target entropy four times the default performed the best, both perceptually and in terms of the highest mean reward. However, there is not necessarily a correlation between mean reward and perceptual similarity. More exploration creates more perceptually odd results, but not necessarily worse in terms of similarity. A concluding remark is that the cross-adaptive environment presumably requires more exploration than benchmark reinforcement learning environments, at least if the default settings in *rllib* and in the official SAC implementation is to be taken as a point of reference.

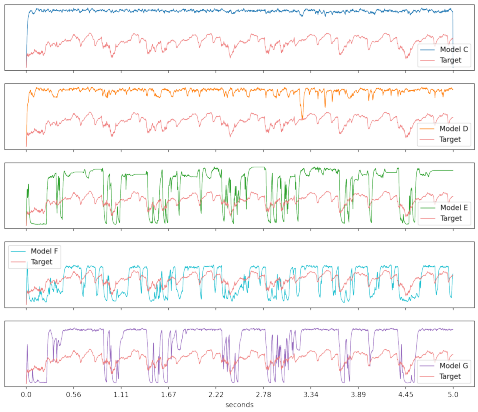
## 5.4 Reward function design

In this experiment, the two reward functions $R_{INV}$ and $R_{REL}$ are compared (see 4.2.3 for an explanation of the functions). Two models were trained for 1000 iterations with SAC, and with exploration hyperparameters set according to the findings in the results on exploration. The two functions output values in different ranges. Thus, it does not



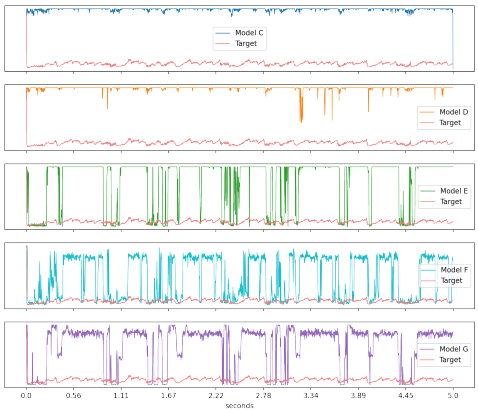

(a) RMS


(b) Pitch

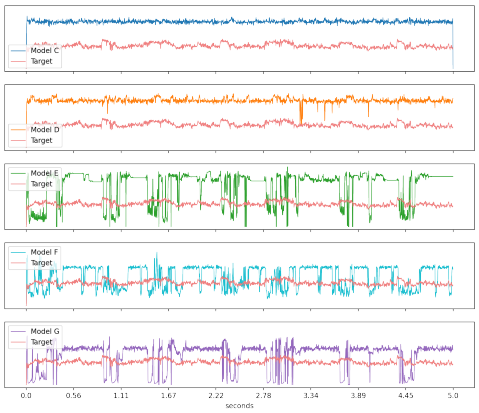



(c) Spectral centroid


(d) Spectral flux




(e) Spectral flatness

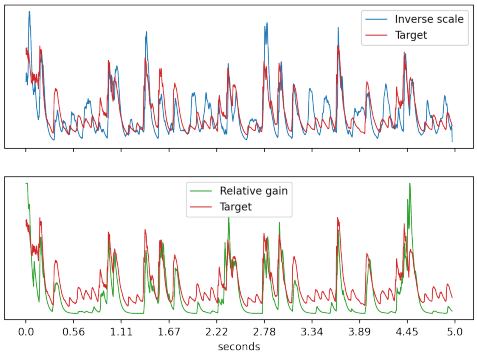



(f) Spectral spread

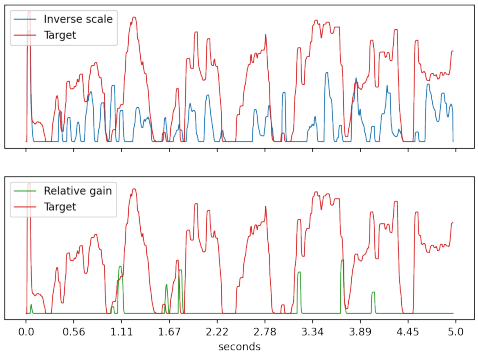
Figure 5.9: Feature plots for models C, D, E, F and G

make sense to compare the mean reward over time. The comparison is therefore done on

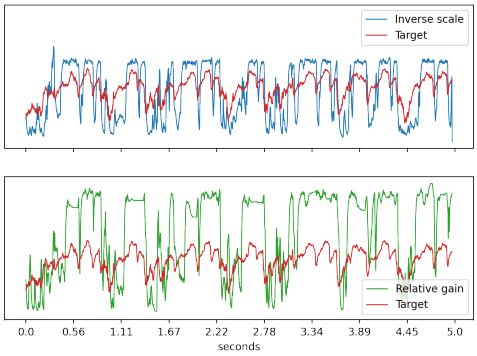
the basis of feature plots and their corresponding audio samples. The feature plots are displayed in figure 5.10 and the audio samples can be found in the digital appendix (4.1 and 4.2).



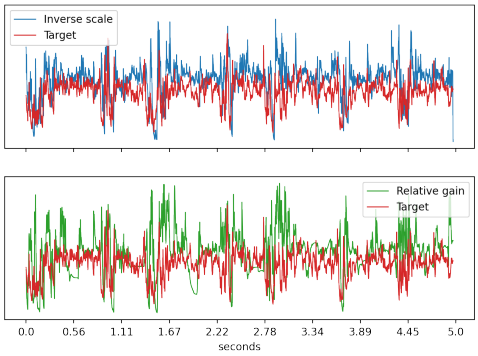

(a) RMS



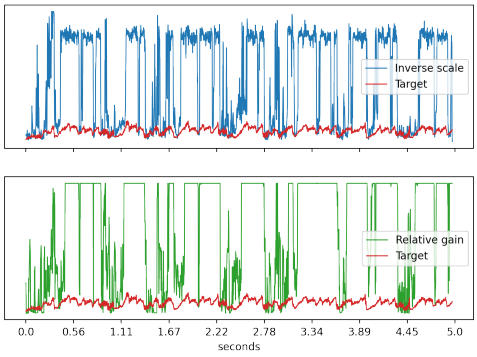

(b) Pitch



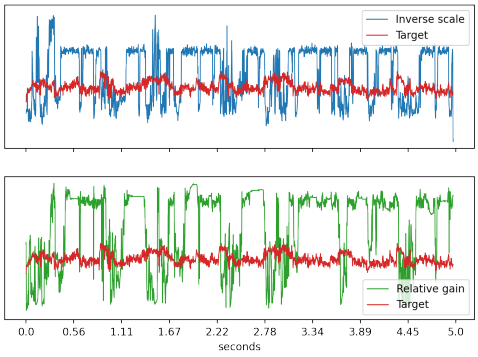

(c) Spectral centroid




(d) Spectral flux




(e) Spectral flatness




(f) Spectral spread

Figure 5.10: Feature plots from using different reward functions

The feature plots reveal that the output of the model trained with the inverse scaling function $R_{INV}$ slightly more similar to the target than the model trained with the relative gain function $R_{REL}$. However, the audio samples have different characteristics, and determining which one of them is the perceptually most similar to the target is debatable.

The conceptual difference between the two functions is that $R_{REL}$ can exploit the advantage of applying a certain mapping, whereas $R_{INV}$ only considers that absolute similarity. It is interesting to note that the function $R_{REL}$ looks like it performs slightly worse from the feature plots since it is a "smarter" measurement of the impact of the agent's actions. One possibility for this somewhat surprising result is that the rewards distributed by $R_{REL}$ will decline over time as the model gets better. This happens because the difference between the euclidean distances will get smaller and smaller as the model starts to converge. This is unfortunate because the agent should naturally still be incentivized to leave the signal as is when the dry signal is close to the target signal.

The discovery motivates a new reward function, one that combines objective similarity as does $R_{INV}$ with the relative advantage of using a mapping over not using one at all, as does $R_{REL}$. Since the output of $R_{REL}$ is a value $> 1$ for mappings that increase the similarity, and $< 1$ for mappings that perform worse than the dry signal, it functions well as a modulator. A new reward function $R_{MIXED}$ was therefore tested:

$$R_{MIXED} = R_{INV} * R_{REL}$$

The results with $R_{MIXED}$ were not as good as either $R_{INV}$ or $R_{REL}$. The resulting output of the model trained with $R_{MIXED}$ can be listened to in the digital appendix (4.3). Even though this test did not succeed, there is a potential that other reward function designs are more tailored to the task at hand than $R_{REL}$, $R_{INV}$, and $R_{MIXED}$ are.

One specific strategy that could help to counteract the decline in reward distribution of $R_{REL}$ as it converges is reward normalization. In many reinforcement learning environments, the reward distribution is unknown in advance, and may vary largely throughout training. Therefore, it is difficult to assert a fixed mean and standard deviation before training starts that can be used to standardize all future rewards. This problem was addressed by van Hesselt et al. who developed a method to adaptively normalize rewards over time [van Hasselt et al., 2016]. This proved to work for normalizing the rewards in many Atari environments quite well overall, without losing too much in terms of performance. Employing this technique to the cross-adaptive environment could help to provide a cleaner reward signal with less periodicity (the periodicity can for instance be observed in figures 5.2 and 5.7), and also mitigate the declining behavior of $R_{REL}$ as the model converges.

## 5.5 Real-time performance

The ultimate goal of this project is to apply the implementation in a cross-adaptive session, whether it is for performance or music production. In any case, the system is not of much use if it does not support real-time. This part of the results is more of a demonstration of the prototype's current capabilities and limitations than it is an experiment. A pre-trained model will be tested in real-time and non-real-time, i.e., asynchronously over OSC and synchronously over Csound channels, respectively, as showed in the signal flowchart in figure 4.1.

Figure 5.11 displays the waveform of audio samples generated by a pre-trained model when running inference real-time (top) and when running inference non-real-time (bottom) with the system at its current status. The audio files are found in the appendix (5.1 and 5.2).

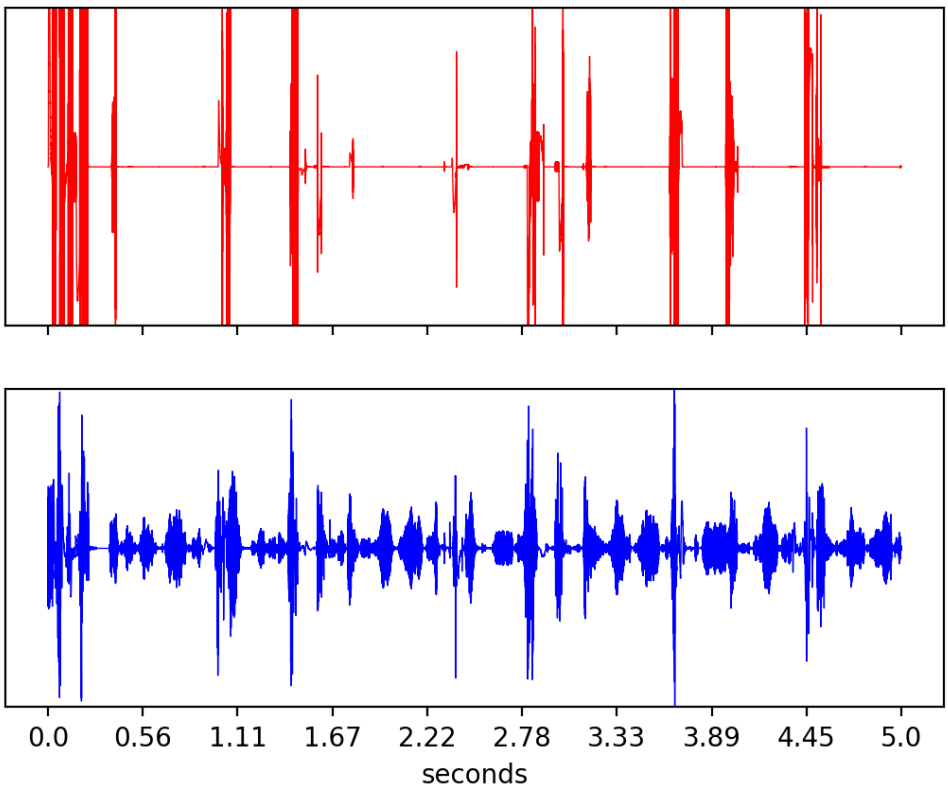



Figure 5.11: Real-time inference (top) vs. non-real-time inference (bottom)

The two waveforms show that the non-real-time synchronous inference works better, which is audible when listening to the audio files as well. However, the timing of the main transients are well defined in both scenarios. This is positive because it shows that the timing of the real-time asynchronous inference works approximately as it should.

A latency test was carried out to determine how long time the different operations in the cross-adaptive environment took during inference. The results show that the forward pass of a neural network takes around 0.12 to 0.15 milliseconds and that generating a mapping, rendering one frame of audio and retrieving analysis from it in total took around 0.15 to 0.22 milliseconds. In total, both operations, which account for the main bulk of one step in the cross-adaptive environment, took less than 0.5 milliseconds. Both operations were averaged over 10000 runs. The trials were run on an Intel i5 core with 16 GB of RAM. Even though there are many personal computers that are slower than that, the forward pass of the neural network and the rendering of the audio should be manageable for most personal computers. The code for the time trial is accessible in the code repository[4].

In 2002, Wessel and Wright suggested that musical instruments should have an upper latency bound of 10 milliseconds [Wessel and Wright, 2002]. However, McPherson et al. [McPherson et al., 2016] recognized that the latency requirements of musical instruments vary depending on the musical context and that some systems can tolerate higher latencies. The overall latency is acceptable for the current implementation, but further work in synchronizing the streams should be cautious of not increasing above the bounds discussed by Wessel and Wright [Wessel and Wright, 2002] and McPherson et al. [McPherson et al., 2016].

Debugging the source code when running asynchronous inference reveals that the OSC messages arrive with uneven intervals at the mediator. This is not surprising since there is no synchronization happening between the two processes. In practice, this means that the observations include source features from one frame at time $t$ and target features from another frame, e.g. $t-1$ or $t+3$, which does not make sense for the objective that the agent is set out to achieve. However, since features of two or more subsequent frames can be relatively similar in audio signals, for instance in periods of silence, some mappings can still be reasonable, and as such produce a waveform that looks approximately similar to the target.

Further work on the project should find a good way to synchronize the two OSC streams from the source and target audio signals. A first iteration of implementing this synchronization could be to timestamp the OSC messages from both streams, and implement a way of correlating packets from the two sources. Techniques from audio and video streaming could also be applied, such as network jitter compensation by buffering incoming packets for a short amount of time. For all methods that aim to synchronize these two streams, careful consideration must be put into how long one can afford to offset the incoming packets to avoid latency issues. To avoid latency issues, it is also possible to

[4] `https://github.com/ulrikah/rave/blob/main/rave/timer.py`

use a lower control rate, or to adjust it dynamically. Halving the control rate will result in half as many forward passes in the neural network, and thus largely reduce the latency in the total system. The downside of decreasing the control rate is that mappings will potentially be less accurate. Finding the right control rate is a matter of trial and error, and may vary between the contexts in which the system is used.

To sum up, live inference does not work ideally yet. Luckily, the current problems are related to synchronization, for which there exist known solutions. Implementations of these solutions should be considerate of latency requirements found in the literature.

## 5.6 Generalizability to new sounds

Measuring how well a machine learning model generalizes to new examples is a vital part of supervised machine learning, where the goal is to develop a model that most accurately is able to represent the patterns in the dataset. The concept of *overfitting* in supervised learning refers to a model that is too closely tied to the specific characteristics of the training dataset, and is unable to generalize to new data. In reinforcement learning, however, the goal is to develop an agent that performs as well as possible in a given environment, but it is not expected that the agent should perform well outside that environment. Therefore, the concept of generalization in reinforcement learning is tied to how robust the agent is to changes in the environment. This is partially why there is such a strong emphasis on exploration in reinforcement learning, since that is eventually how the agent can continue to find new and better solutions. As an example, the policy of an agent trained on an Atari game environment should ideally be indifferent to the colors in the background of the game, as the background do not impact the reward or future states in any way. Similarly, the cross-adaptive agent should ideally be resilient to minor changes in the source and target sounds that are irrelevant to how humans perceive these sounds.

The results in figure 5.12 show how the best-performing model of section 5.3 (model F) performs when being tested with two new sounds[5]. One of the new sounds is an 80's styled drum beat, and was chosen because it has a similar structure to the drum beat that was used as the original target during training. The other new sound is an arpeggiated synth sequence, and was chosen because it is different to the original source and target. All the audio samples are available in the digital appendix (6.1, 6.2, 6.3, 6.4, and 6.5).

The first experiment, where the 80's styled drum beat is used as the new target, shows

[5]Both samples are taken from Legowelt's sample pack of sounds from the Yamaha PSS795 synthesizer

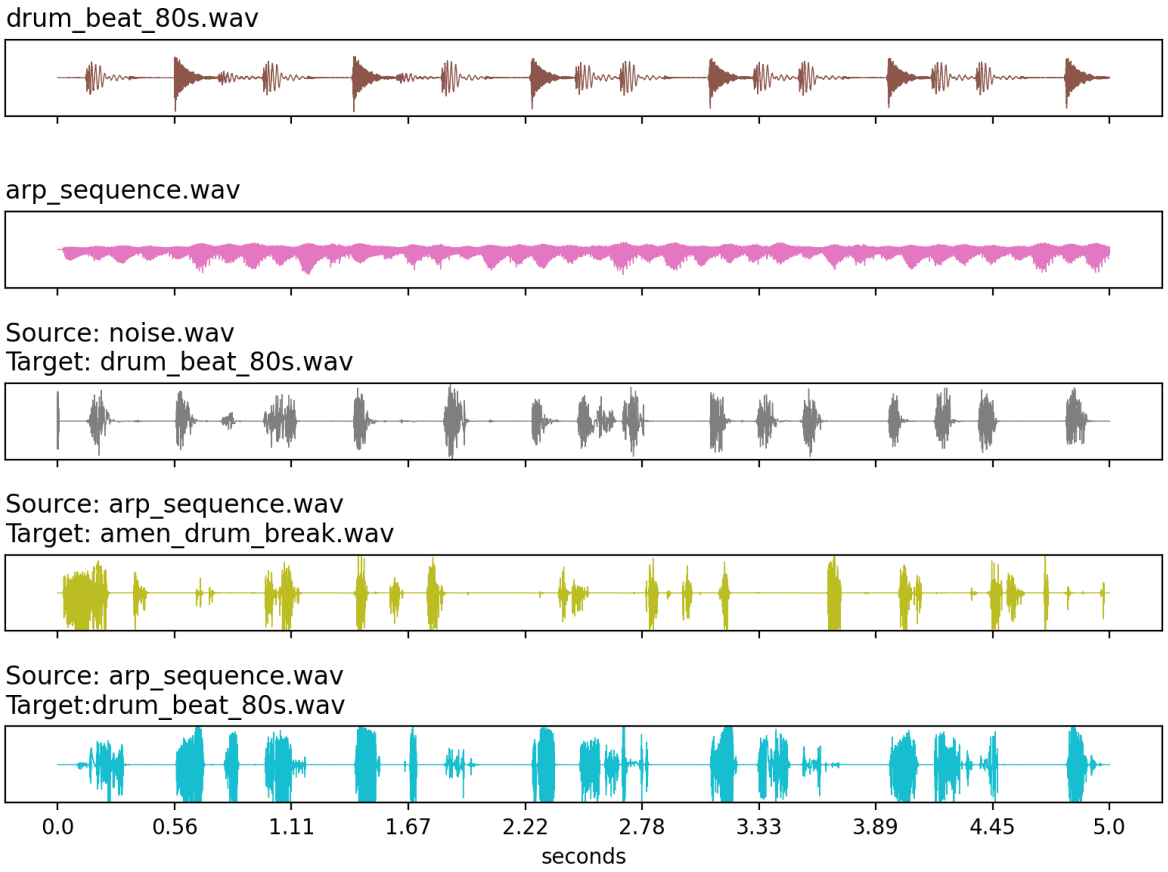



Figure 5.12: Generalizability to new sounds

that the model manages to retrieve the structure of the source audio. Compared to how the model performed on the original target (the Amen drum break), the drum hits are less distinct from one another than what they were for the original target.

In the second experiment, a drastically different sound is used as the source input to the model while the target is unchanged. The resulting audio has the structure of the Amen break, but with a timbre that is more similar to the source sound. It is difficult to hear the characteristics of the drum hits in the target sound, except for the rhythmic structure.

The third experiment combines the two new sounds. The arpeggiator sequence is used as source, and the 80's drum beat is used as target. The result is similar to the second experiment, where the structure of the drum beat is clearly audible, but the timbre is derived from the source sound.

Summing up, it seems that the model is able to generalize better on the structural elements, such as transient hits, but less so for the timbral characteristics. This is not very surprising, as the model is trained with a highly rhythmic audio sample as target. Additionally, the distorted low-pass effect limits the possibilities of the model to transform

melodic elements of the audio.

Since the current system implementation only supports training an agent on a specific pair of sounds, it is not expected that it should learn to generalize well to new sounds. The system architecture should be improved by allowing training with multiple audio files, or even with real-time audio streams. An additional possibility is to slightly modify the source and target during training with data augmentation. Jordal showed that data augmentation increased the robustness of the models he trained [Jordal, 2017]. Possible data augmentation techniques are, for instance, to alter the samples' speed slightly or add more noise or distortion.

# Chapter 6

# Conclusion

This chapter contains reflections on the results by considering their limitations, and it provides potential directions for future work. At the very end, a summary will conclude the thesis.

Overall, the results from the previous chapter show that it is possible to model a simplified version of a cross-adaptive performance scenario as a reinforcement learning environment. Consequently, reinforcement learning can be used as a method for creating mappings in cross-adaptive audio effects. From the comparison of the two reinforcement learning algoritms, SAC was found to perform better than PPO in the cross-adaptive environment. Furthermore, the results in section 5.3 show that exploration seems to be more important in the cross-adaptive environment than for the typical benchmark environments found in the literature. The results also demonstrate that both feature selection and reward function design alter the agent's learning objective, and thus impact what the agent is able to learn.

## 6.1 Limitations

Considering all the possible aspects of a cross-adaptive performance session, the scope of this thesis has been narrow. All the results were discussed using only a limited set of sounds, features, and effects. Hence, it is difficult to generalize too much based on the results in this project. For instance, all test cases use noise as source sound and a drum break as target sound. The study of cross-adaptive effects to shape highly melodic sounds remains unexplored. Such a study would also require a closer look into other digital audio effects, in order to make sure that the effect would be capable of transforming the relevant

features of the sound.

Time-varying effects such as delay and reverb have purposefully been avoided in this project. This effect category complicates the current model of the cross-adaptive scenario because the result of processing a single frame of audio is not necessarily measurable in the subsequent frame. As an example, imagine a delay effect with a delay line of 200 ms. The result of sending an audio frame through that delay line would be observable first 200 ms later. With a frame size 128 of samples, 200 ms corresponds to roughly 68 audio frames with a sample rate of 44100 Hz ($\frac{44100*0.2}{128} = 68.9$ frames). From the perspective of the reinforcement learning agent, it would be challenging to know that the action it picked 68 steps ago is what yielded the reward in the most recent state. The difficulty in learning from time-varying effects is a drawback in the current design of the model of the cross-adaptive scenario. This drawback could be mitigated by leveraging the power of specially designed neural networks that can learn from data with spatial or temporal relationships. Long-short-term-memory (LSTM) networks [Hochreiter and Schmidhuber, 1997], a form of recurrent neural networks, are often applied to such use cases and could be interesting to apply to the cross-adaptive environment as well, to account for time-varying effects. Convolutional neural networks (CNN) with one-dimensional kernel convolutions is another possibility, with which Brandtsegg and Tidemann were successful in learning mappings between gesture input and synthesis parameters [Brandtsegg and Tidemann, 2020]. They were able to use CNNs because the gestural input had a spatial relationship, thus achieving the same effect as with a recurrent neural network.

Both reward functions described in section 4.2.3 use euclidean distance to measure the similarity between features. The results show that some features may be more perceptually significant than others, which the euclidean distance measurement does not take into account. This can and probably should be refined if further work with the system is to be done. Feature masking is a potential way of assigning relative value to some feature extractors and could be relevant to use for performances. However, assigning relative weights to the features is a direct way of introducing bias in the model. This hurts the generalizability of the results and therefore has been avoided in this thesis.

## 6.2 Reflections and future work

The point of the cross-adaptive research project was to explore cross-adaptive processing in music production and performance. Therefore, it is worth reflecting on the possibilities and implications of applying the cross-adaptive reinforcement learning agent to a performance and production scenario. Fiebrink et al.'s idea of the machine learning algorithm

as a musical interface [Fiebrink et al., 2016] is a good framework for understanding the cross-adaptive environment as an interface.

Traditional human-computer interfaces for audio processing found in digital audio workstations (DAW) usually allow interaction with control parameters. The control parameters are often tightly coupled to the audio processing that happens in the backend of the interface, e.g., controlling the cutoff frequency of a digital audio filter or the room size of a digital reverb effect. In a machine learning model employing an ANN, the direct coupling between input and output is the network's weights and biases. The relatively simple ANN used for most of the test cases in the previous section has an input layer of 6 nodes, a hidden layer of 16 nodes, and an output layer of 6 nodes, which sums up to 192 weights in total ($6 * 16 + 16 * 6 = 192$). Allowing the user to control all the parameters directly would likely constitute a poor musical interface since the impact of one single weight on shaping the audio signal is modest. The beauty of using neural networks is that it is possible to create high-level mappings between input and output. Determining how to best interact with such an interface should therefore try to leverage the potential in these high-level mappings.

Consider the hypothetical scenario where the cross-adaptive agent's policy is the optimal policy, where the agent knows perfectly how to map a vector of audio features to an effect parameter combination to achieve the maximum expected return. If the agent were trained to optimize for similarity between the source and target sound, the agent would be outstanding at making two sound sources sound as similar as possible to each other with a given audio effect. At this point, the technical goal of creating automatic mappings in cross-adaptive audio effects would be accomplished. However, from the perspective of a musician who was to perform with this model as a musical interface, she would now have a black box effect that allowed her to imitate another sound source. Even though the limitations of this black box could be entertaining to investigate as part of a creative practice, it seems unlikely that it would be an interesting tool to use for various musical styles and all kinds of different timbres without any human interaction.

In the context of musical applications of machine learning, Fiebrink and Caramiaux motivate an understanding of the machine learning algorithm as any other human-computer interface whose affordances are the properties of the algorithm that align with the goals and abilities of the musician [Fiebrink et al., 2016]. Since the cross-adaptive agent essentially is a machine learning algorithm, it is also possible to consider the cross-adaptive agent as a musical interface. Building on Fiebrink and Caramiaux's perspectives, for the cross-adaptive agent to be usable as an interface for a human musician, its affordances have to align with the goals of the musicians that are taking part in cross-adaptive perfor-

mances [Fiebrink et al., 2016]. Bearing in mind the subtitle of the cross-adaptive research project, "exploring radically new modes of musical interaction in live performance"[1], it is fair to assume that the goals of the musicians performing in a cross-adaptive session would be to explore new directions in terms of musical interaction and collaboration with their peers. The cross-adaptive agent as a musical interface should thus afford exploration of the unknown, but at the same time give the user a sense of control and freedom. Determining exactly how to best frame the cross-adaptive agent for these affordances is a work on its own and would require close collaboration with performing musicians in the target group. However, it is still possible to reflect on the system in its current shape and speculate on how it could be framed as a musical interface.

A policy can only learn from observations it receives from the environment and select corresponding actions within the action space. Therefore, one of the most unmediated ways of interacting with the cross-adaptive agent is to decide what these observations should be and which actions the agent should be allowed to take. In practice, this means that a musician should be able to choose which features and effects should be part of training the cross-adaptive policy. This would grant the user total control of the learning capabilities of the system and the control to shape the agent for their musical taste. The interface should support basic functionality such as saving presets of feature extractors and effects, as well as saving pre-trained models, for later reuse. Additionally, designing custom reward functions could be another way of personalizing the agent for the user's aesthetic preferences. A user could, hypothetically, design a reward function that distributes high rewards for when the estimated pitches relate harmonically, but when the transient density differs, effectively creating a correlation between harmonic content and an inverse correlation between rhythmic content. Also, the stochastic policy of the cross-adaptive agent allows sampling actions with a specified degree of randomness. Tuning the temperature of the action sampling could be a direct way of letting the user control exploratory aspects of the cross-adaptive agent during a live performance.

For a live performance, it would be interesting to let the performers fully interact with the cross-adaptive agent by also enabling training it in real-time. The current system design only supports real-time inference of a pre-trained model. Allowing real-time training puts stricter performance requirements on the system and would possibly affect the model's accuracy. In turn, training in real-time has other affordances, most notably allowing the performer to make more spontaneous choices amid a performance.

There are also other benefits to reinforcement learning in an interactive context, such as when training during a performance. Scurto et al. showed that it is possible to use

[1] `http://crossadaptive.hf.ntnu.no/`

direct human feedback in the reward function design to let people interactively guide a reinforcement learning agent to learn their personal preferences for parameter space exploration [Scurto et al., 2021]. Leveraging human feedback during the training process would allow for a performer to shape the agent while at the same time performing with it. In a more experimental concert configuration, one could also envision the audience giving feedback to the agent, effectively contributing to the musical expression of the performers on stage.

Even though it would be ideal to be able to train the cross-adaptive agent in real-time, it would take a longer time for the agent to converge due to the lowered frequency of the policy updates. This is a potential problem of real-time training for live settings. A trade-off between a real-time and non-real-time training approach could be to use some form of hybrid between supervised learning and reinforcement learning. This way, the agent could be trained on some dataset of audio samples provided by the user, with the idea that one typically would like the agent to inherit some base knowledge of the cross-adaptive processing that could be supplied with further context-specific training examples. The training process could then continue in real-time, e.g., during a performance rehearsal, to reflect the distinctive characteristics of the specific cross-adaptive performance session. Kumar et al. proposed an efficient way of leveraging collected datasets in reinforcement learning models with conservative Q-learning [Kumar et al., 2020]. Conservative Q-learning tackles the problem of distributional shifts between datasets and the episodes collected in real-time by providing a lower bound on the value estimation of $(state, action)$ pairs (the Q-function). According to the authors, this can guarantee theoretical improvements to further policy updates [Kumar et al., 2020]. In that respect, conservative Q-learning could be one possible way of combining synchronous and asynchronous real-time training.

Lastly, packaging the cross-adaptive agent within an audio processing plugin would allow combining the machine learning algorithm with other types of audio effects and modulators. This would be ideal for supporting any desirable digital musical workflow. Commercial DAWs like Ableton Live and Logic Pro have a number of native plugins, i.e., audio effects, modulators, feature extractors, and also have support for third-party VST and Audio Units (AU) plugins. ADEPT, the framework for adaptive digital audio effects mentioned in the background [Campbell et al., 2016], has taken this route and is packaged as an audio plugin. In ADEPT, users can manually choose how feature extractors on one side should adapt audio effects on the other side. ADEPT has a range of feature extractors built-in and automatically discovers the ranges of the effect parameters in the DAW in which it is used. In that way, ADEPT can use its internal feature extractors to modulate external effects. Embedding the cross-adaptive agent within a framework such as ADEPT would be a convenient way of deploying the cross-adaptive agent in existing

infrastructure for music production and performance.

## 6.3 Summary

This thesis has investigated the research question of how a simplified version of a cross-adaptive performance scenario can be modeled as a reinforcement learning environment to create mappings in cross-adaptive audio effects. It builds upon work from the cross-adaptive research project, and more broadly, on a legacy of research and inventions in music technology and machine learning.

The contributions of this thesis are the research and development of an open-source system capable of creating mappings in cross-adaptive audio effects with techniques from reinforcement learning. Even though the implemented system should be considered a prototype, this thesis demonstrates that reinforcement learning is a viable technique for creating mappings in cross-adaptive audio effects. The crude implementation of asynchronous real-time inference shows that it should be feasible to use the system in real-time over OSC.

There is a lot left to be done in figuring out how the cross-adaptive environment best is brought into a performance and production context. This work would require systematic user testing. To further explore the affordances of reinforcement learning, the cross-adaptive reinforcement learning environment could be extended to incorporate human feedback in the learning process. Future work should also look into the generalizability of the results by combining other effects and feature extractors.

# References

[Achiam, 2018] Achiam, J. (2018). OpenAI - Spinning up in deep reinforcement learning. https://spinningup.openai.com/.

[Barto et al., 1983] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846.

[Bernardo et al., 2020a] Bernardo, F., Kiefer, C., and Magnusson, T. (2020a). Designing for a Pluralist and User-Friendly Live Code Language Ecosystem with Sema. In *Proceedings of the 2020 International Conference on Live Coding (ICLC2020)*, pages 41–57, Limerick, Ireland. University of Limerick.

[Bernardo et al., 2020b] Bernardo, F., Kiefer, C., and Magnusson, T. (2020b). A signal engine for a live coding language ecosystem. *Journal of the Audio Engineering Society*, 68(10):756–766.

[Brandtsegg, 2015] Brandtsegg, Ø. (2015). A toolkit for experimentation with signal interaction. In *Proceedings of the 18th International Conference on Digital Audio Effects (DAFx-15)*, pages 42–48.

[Brandtsegg et al., 2018] Brandtsegg, Ø., Engum, T., and Wærstad, B. I. (2018). Working methods and instrument design for cross-adaptive sessions. *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

[Brandtsegg and Tidemann, 2020] Brandtsegg, Ø. and Tidemann, A. (2020). {Shape: An adaptive musical interface that optimizes the correlation between gesture and sound. *Proceedings of the International Conference on Live Interfaces (ICLI)*.

[Campbell et al., 2016] Campbell, O., Roads, C., Cabrera, A., Wright, M., and Visell, Y. (2016). ADEPT: A framework for adaptive digital audio effects. In *2nd AES Workshop on Intelligent Music Production (WIMP)*.

[Chourdakis and Reiss, 2016] Chourdakis, E. T. and Reiss, J. D. (2016). Automatic control of a digital reverberation effect using hybrid models. In *Audio Engineering Society*

*Conference: 60th International Conference: DREAMS (Dereverberation and Reverberation of Audio, Music, and Speech)*. Audio Engineering Society.

[Chourdakis and Reiss, 2017] Chourdakis, E. T. and Reiss, J. D. (2017). A machine-learning approach to application of intelligent artificial reverberation. *Journal of the Audio Engineering Society*, 65(1-2):56–65.

[Collins, 2008] Collins, N. (2008). Reinforcement Learning for Live Musical Agents. In *International Computer Music Conference*, Belfast.

[Dahlstedt, 2001a] Dahlstedt, P. (2001a). Creating and exploring the huge space called sound: Interactive evolution as a composition tool. In *Proceeding of Music Without Walls, Music Without Instruments Conference*.

[Dahlstedt, 2001b] Dahlstedt, P. (2001b). A MutaSynth in parameter space: Interactive composition through evolution. *Organised Sound*, 6(2):121–124.

[Eldhuset, 2016] Eldhuset, A. W. (2016). *Using Genetic Algorithms to Find and Evaluate Parameters for Adaptive Digital Audio Effects*. Master's Thesis, NTNU.

[Emmerson et al., 2018] Emmerson, S., Baalman, M., and Brandtsegg, Ø. (2018). Instrumentality, perception and listening in crossadaptive performance. *Proceedings of the International Conference on Live Interfaces (ICLI)*.

[Engel et al., 2020] Engel, J., Hantrakul, L. H., Gu, C., and Roberts, A. (2020). DDSP: Differentiable digital signal processing. In *International Conference on Learning Representations (ICLR)*.

[Fakoor et al., 2017] Fakoor, R., He, X., Tashev, I., and Zarar, S. (2017). Reinforcement Learning To Adapt Speech Enhancement to Instantaneous Input Signal Quality. In *Neural Information Processing Systems (NIPS) Machine Learning for Audio Signal Processing Workshop*.

[Fiebrink et al., 2016] Fiebrink, R., Caramiaux, B., Dean, R., and McLean, A. (2016). *The Machine Learning Algorithm as Creative Musical Tool*. Oxford University Press.

[Fiebrink and Cook, 2010] Fiebrink, R. and Cook, P. R. (2010). The Wekinator: A system for real-time, interactive machine learning in music. In *Proceedings of The Eleventh International Society for Music Information Retrieval Conference (ISMIR)*, Utrecht.

[Gibson, 1977] Gibson, J. J. (1977). The theory of affordances. *Hilldale, USA*, 1(2):67–82.

[Haarnoja et al., 2018a] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018a). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic

actor. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR.

[Haarnoja et al., 2018b] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2018b). Soft Actor-Critic Algorithms and Applications. *CoRR*, abs/1812.05905.

[Henderson et al., 2018] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In McIlraith, S. A. and Weinberger, K. Q., editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 3207–3214. AAAI Press.

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

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

[Jordal, 2017] Jordal, I. (2017). *Evolving Artificial Neural Networks for Cross-Adaptive Audio Effects*. Master's Thesis, NTNU.

[Kumar et al., 2020] Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative Q-learning for offline reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 33, pages 1179–1191. Curran Associates, Inc.

[Lan et al., 2019] Lan, Q., Tørresen, J., and Jensenius, A. R. (2019). RaveForce: A Deep Reinforcement Learning Environment for Music Generation. In *Proceedings of the SMC Conferences*, pages 217–222. Society for Sound and Music Computing.

[LeCun et al., 2012] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–48. Springer.

[McPherson et al., 2016] McPherson, A. P., Jack, R. H., and Moro, G. (2016). Action-Sound Latency: Are Our Tools Fast Enough? *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. In *Neural Information Processing Systems (NIPS) Deep Learning Workshop*.

[Nielsen, 2015] Nielsen, M. A. (2015). *Neural Networks and Deep Learning*, volume 25. Determination press San Francisco, CA.

[Perez-Gonzalez and Reiss, 2009] Perez-Gonzalez, E. and Reiss, J. (2009). Automatic equalization of multichannel audio using cross-adaptive methods. In *Audio Engineering Society Convention 127*. Audio Engineering Society.

[Reiss and Brandtsegg, 2018] Reiss, J. D. and Brandtsegg, Ø. (2018). Applications of cross-adaptive audio effects: Automatic mixing, live performance and everything in between. *Frontiers in Digital Humanities*, 5:17.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, USA.

[Salimans et al., 2017] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv:1703.03864 [cs, stat]*.

[Sarkar et al., 2017] Sarkar, S., Reiss, J. D., and Brandtsegg, Ø. (2017). Investigation of a drum controlled cross-adaptive audio effect for live performance. In *Proceedings of the 20th International Conference on Digital Audio Effects (DAFx-17)*.

[Schedl et al., 2014] Schedl, M., Gómez Gutiérrez, E., and Urbano, J. (2014). Music information retrieval: Recent developments and applications. *Foundations and Trends in Information Retrieval*, 8:127–261.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347.

[Scurto et al., 2018] Scurto, H., Bevilacqua, F., and Caramiaux, B. (2018). Perceiving Agent Collaborative Sonic Exploration In Interactive Reinforcement Learning. In *Proceedings of the 15th Sound and Music Computing Conference (SMC 2018)*, Limassol, Cyprus.

[Scurto et al., 2021] Scurto, H., Kerrebroeck, B. V., Caramiaux, B., and Bevilacqua, F. (2021). Designing Deep Reinforcement Learning for Human Parameter Exploration. *ACM Transactions on Computer-Human Interaction*, 28(1):1:1–1:35.

[Silver et al., 2014] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, pages 387–395, Beijing, China.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.

[Suzuki and Takeshima, 2004] Suzuki, Y. and Takeshima, H. (2004). Equal-loudness-level contours for pure tones. *The Journal of the Acoustical Society of America*, 116(2):918–933.

[van Hasselt et al., 2016] van Hasselt, H. P., Guez, A., Guez, A., Hessel, M., Mnih, V., and Silver, D. (2016). Learning values across many orders of magnitude. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 29. Curran Associates, Inc.

[Van Kerrebroeck et al., 2018] Van Kerrebroeck, B., Bevilacqua, F., and Scurto, H. (2018). *Deep Reinforcement Learning for the Design of Musical Interaction*. Master's Thesis, Sorbonne.

[Verfaille and Arfib, 2001] Verfaille, V. and Arfib, D. (2001). A-DAFx: Adaptive digital audio effects. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-01)*.

[Verfaille et al., 2006] Verfaille, V., Zolzer, U., and Arfib, D. (2006). Adaptive digital audio effects (A-DAFx): A new class of sound transformations. *IEEE Transactions on audio, speech, and language processing*, 14(5):1817–1831.

[Visi, 2020] Visi, F. G. (2020). Towards Assisted Interactive Machine Learning: Exploring Gesture-Sound Mappings Using Reinforcement Learning. *Proceedings of the International Conference on Live Interfaces (ICLI)*.

[Walsh, 2008] Walsh, R. (2008). Cabbage, a new GUI framework for Csound. In *Proceedings of the Linux Audio Conference*, volume 21, Cologne, Germany. Citeseer.

[Warnell et al., 2018] Warnell, G., Waytowich, N., Lawhern, V., and Stone, P. (2018). Deep TAMER Interactive agent shaping in high-dimensional state spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[Wessel and Wright, 2002] Wessel, D. and Wright, M. (2002). Problems and prospects for intimate musical control of computers. *Computer music journal*, 26(3):11–22.

[Zölzer et al., 2002] Zölzer, U., Amatriain, X., Arfib, D., Bonada, J., De Poli, G., Dutilleux, P., Evangelista, G., Keiler, F., Loscos, A., and Rocchesso, D. (2002). *DAFX-Digital Audio Effects*. John Wiley & Sons.

# Appendix A

# Digital appendix

This appendix only serves as a container for links to the three digital appendices: the repository containing the source code, the webpage containing the audio samples, and the summarizing blog post.

- Source code repository: `https://github.com/ulrikah/rave`
- Audio samples: `https://ulrikah.github.io/thesis`
- Blog post: `https://mct-master.github.io/masters-thesis/2021/05/15/ulrikah-rave.html`