NTNU

Norwegian University of
Science and Technology

Evolving Artificial Neural Networks for Cross-Adaptive Audio Effects

Iver Jordal

TDT4501 Computer Science, Specialization Project

Submission date:	June 2016
Supervisor:	Gunnar Tufte
Co-supervisor:	Øyvind Brandtsegg

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Cross-adaptive audio effects have many applications within music technology, including for automatic mixing and live music. The common methods of signal analysis capture the acoustical and mathematical features of the signal well, but struggle to capture the musical meaning. Together with the vast number of possible signal interactions, this makes manual exploration of signal mappings difficult and tedious. This project investigates Artificial Intelligence (AI) methods for finding useful signal interactions in cross-adaptive audio effects. A system for doing signal interaction experiments and evaluating their results has been implemented. Since the system produces lots of output data in various forms, a significant part of the project has been about developing an interactive visualization tool which makes it easier to evaluate results and understand what the system is doing. The overall goal of the system is to make one sound similar to another by applying audio effects. The parameters of the audio effects are controlled dynamically by the features of the other sound. The features are mapped to parameters by using evolved neural networks. NeuroEvolution of Augmenting Topologies (NEAT) is used for evolving neural networks that have the desired behavior. Several ways to measure fitness of a neural network have been developed and tested. Experiments show that a hybrid approach that combines local euclidean distance and Nondominated Sorting Genetic Algorithm II (NSGA-II) works well. In experiments with many features for neural input, Feature Selective NeuroEvolution of Augmenting Topologies (FS-NEAT) yields better results than NEAT.

Table of Contents

1 Introduction

For decades, music technology has made music more appealing by applying audio effects, which are processing techniques that alter audio so that it sounds different. For example, one common audio effect in rock music is distortion on the electric guitars, which will make them sound "fuzzy" instead of "clean". Some audio effects become more appealing when their parameters are changed over time. For example, there is the auto-wah effect, which essentially is a peaking filter, which amplifies a specific frequency and cuts off other frequencies. The volume of the input is used to dynamically control the cutoff frequency of the filter. When the cutoff frequency is swept from low to high, it sounds like "wah", hence the name auto-wah. The auto-wah effect is an example of an *adaptive audio effect* (Verfaille & Arfib, 2006). Since meaningful interactions between musical elements can make music more interesting and appealing, *cross-adaptive audio effects* were invented. In this class of audio effects, parameters are dynamically informed by features of other sounds. In the 1960s, Stockhausen presented one of the first cross-adaptive audio effects in his composition "Hymnen" (Moritz, 2003), where he, amongst other things, modulated the rhythm of one anthem with the harmony of another anthem. Sidechain compression is a more recent example of a cross-adaptive audio effect. In that technique, the amplitude of one sound controls one or more parameters in the compressor that is applied to a different sound. In electronic music, sidechain compression is often used to let the volume of the bass drum turn down the volume of the bass synth. This is done to avoid conflicts between the bass drum and the bass synth, and also provides a pulsating, rhythmic dynamic to the sound. Further, cross-adaptive audio effects have been used in algorithms for mixing multichannel audio (Reiss, 2011) and voice-controlled synthesizers (Cartwright & Pardo, 2014). Generally, cross-adaptive audio effects can be applied in a wide range of research fields, including live music performance and audio mastering. Current research at the Music Technology department at Norwegian University of Science and Technology aims at exploring radically new modes of musical interaction in live music performance. In 2015, Øyvind Brandtsegg presented a toolkit for experimenting with signal interaction. This toolkit enables one to find musically interesting signal interactions by empirical experimentation. However, this can be tedious due to the vast number of combinations. Also, while most low-level audio features are mathematically and acoustically well defined, it's hard to use them for musically interesting cross-adaptive audio effects. One often needs to combine several audio features in complex ways. An audio feature can be linked to any effect parameter, and the mapping function can be anything. A setup can have many instruments, lots of audio effects, and the ordering of the effects may vary. Indeed, Brandtsegg's suggestions for future work includes "practical and musical exploration of the technique, and the mapping between sound features and effects controls". As cross-adaptive audio effects are relatively uncharted territory, methods to evaluate various cross-couplings of features have not been formalized. There is a need for a tool that can help efficiently search for useful mappings in those huge search spaces. That was the spark of this specialization project.

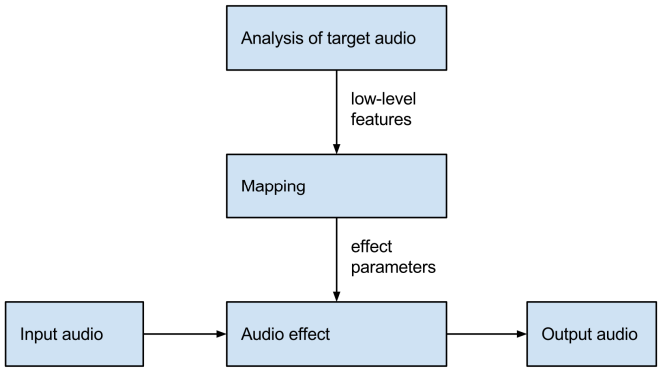


Figure 1.1: Cross-adaptive audio effect process with two audio streams: input audio and target audio

In practice, a music performance that uses cross-adaptive audio effects can be a very complex, dynamic system with many signal interactions. However, to limit the scope and complexity of the problem, this project will study signal interactions between only two sounds at a time: an input sound and a target sound. A single audio effect is applied to the input sound. The parameters of this effect are dynamically informed by features from the target audio. The goal of the tool implemented in this project is to make interesting mappings from target audio analysis to audio effect parameters. Brandtsegg (2015) suggested that machine learning can be useful in this context. Generally, for a machine learning problem to be well-defined, a performance measure is needed. A good performance measure would be "how appealing does it sound?". However, what generally sounds appealing to humans is tacit knowledge and cannot be simply described mathematically. Because there are no good objective measures of what a good cross-adaptive audio effect is, an assumption has been made in this project: If a cross-adaptive audio effect makes features of one sound audible in the other sound, then it is considered interesting. Therefore it has been decided that the objective of the system in this project should be to make the input sound similar to the target sound.

The mapping (figure 1.1) is a function that maps n-dimensional vectors (audio features) to m-dimensional vectors (effect parameters). Artificial neural networks are apt for this, as they can approximate a wide variety of continuous functions (Hornik, 1991). Backpropagation (Werbos, 1982; Lecun, Buttou, Orr, & Müller, 1998) is an efficient algorithm for training neural networks (i.e. finding a set of useful weights for it), but it requires target values for the output nodes to compute error signals. Since no sets of target values are available for this project, and there is generally no good way of telling exactly how the mapping affects the resulting sound, it makes sense to use evolutionary computation to train the neural networks instead. That is feasible because it is possible to construct a fitness function that returns a score based on how similar two sounds are. All in all, this project is about developing a toolkit that explores evolutionary computation in various ways to evolve artificial neural networks that act as mappings in cross-adaptive audio effects.

As the developed system has many components and deals with lots of numbers in the form of audio features, neural network weights, audio effect parameters, output sounds and fitness values, it's hard to understand what the system really does. To alleviate this, a significant part of the project has been about developing a comprehensive interactive visualization system as a part of the toolkit. One motivation for this is that one cannot improve the optimization process if one does not know where it fails. It was also made in order to make the toolkit more human-understandable and to make evaluation of results more efficient.

2 Background information

2.1 Genetic Algorithms

Genetic algorithms (Goldberg, 1989) are iterative algorithms that can approximate solutions to optimization problems. In such problems, one usually doesn't know how to construct a good solution, but it is possible to measure how good a solution is. The methods used in genetic algorithms are inspired by Darwin's principle of natural selection. In the algorithm, a population of individuals is simulated through generations of "life". Each individual is a candidate solution to the optimization problem. The fittest individuals, as determined by a fitness function, are the individuals that are most likely to survive and reproduce (either asexually or sexually). Individuals that are deemed less fit are more likely to die young, and don't get to pass their genes on to future generations. During reproduction, crossover and mutation occurs. Crossover is a genetic operator that combines two parents to produce an offspring. Mutation is a genetic operator that alters an individual slightly.

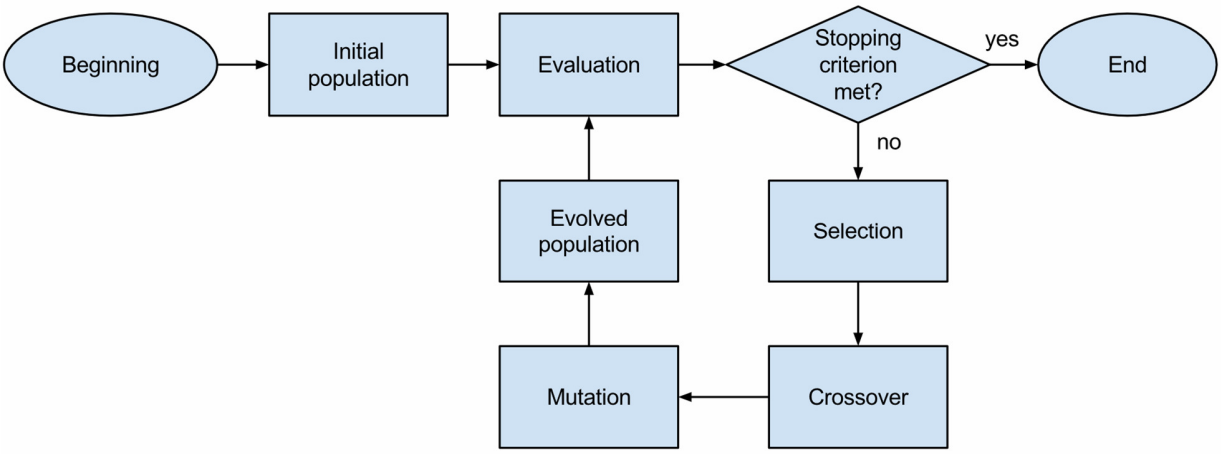


Figure 2.1: Genetic algorithm cycle

2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are systems of interconnected "neurons", or nodes (Caudill, 1987). A connection from a node A to a different node B means that the activation level of node A influences the activation level of node B based on the numerical weight of the connection. The activation level of a node is calculated by adding up all incoming signals to that node and running that number through the node's activation function. An ANN can be thought of as a function that transforms n-dimensional vectors to m-dimensional vectors.

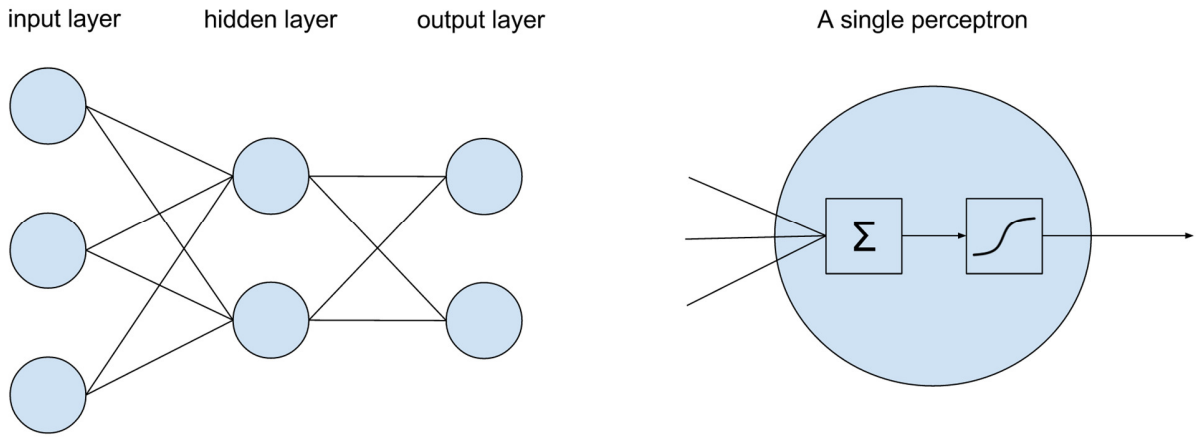


Figure 2.2: To the left: Illustration of a small neural network with one hidden layer. To the right: Illustration of a perceptron with sigmoid activation function

2.3 Neuroevolution

Neuroevolution is a technique that uses evolutionary algorithms to train artificial neural networks. It differs from supervised learning algorithms such as backpropagation in that it does not require a set of correct input-output pairs. Instead, only a performance measure (fitness function) is needed.

2.4 NeuroEvolution of Augmenting Topologies (NEAT)

NEAT (Stanley & Miikkulainen, 2002) is a neuroevolution technique that evolves neural networks with genetic algorithms. Not only the weights of the ANN are evolved, but also the structure. The NEAT approach begins with a feed-forward approach with input nodes and output nodes that are fully connected. The ANN can then grow larger by having nodes and links added to it. NEAT can also remove nodes and links.

2.5 Multi-Objective evolutionary algorithm

As real-life problems often have more than one objective, there is a need for ways to deal with multiple objectives effectively. A multi-objective evolutionary algorithm (MOEA) is an algorithm for solving mathematical optimization problems involving more than one objective function to be optimized simultaneously (Veldhuizen & Lamont, 2000). One well-known algorithm of this kind is Nondominated Sorting Genetic Algorithm II (NSGA-II) (Deb, Pratap, Agarwal, & Meyarivan, 2002). In this algorithm, the performance measure is based primarily on rank and secondarily on crowding distance. Rank is calculated by running the fast non-dominated sort algorithm. This algorithms assigns a rank to each individual. If the rank of individual A is better than the rank of another individual B, it means that A dominates B. Individual A dominates B if both the following conditions are true:

- The solution A is no worse than B in all objectives.
- The solution A is strictly better than B in at least one objective.

Crowding distance is a way to measure how crowded the search space around the individual is. Crowding distance is quantified by forming a cuboid with the nearest neighbours as vertices, and then taking the average of the side lengths of the cuboid. Large crowding distances are encouraged because it preserves diversity in the population (Deb, Pratap, Agarwal, & Meyarivan, 2002).

2.6 Audio feature extraction tools

Audio feature extraction is the process of computing a compact numerical representation that can be used to characterize a segment of audio. Low-level features such as spectral centroid and Mel-Frequency Cepstral Coefficients (MFCC) (Mermelstein, 1976; Logan, 2000) are computed directly from the audio signal, frame by frame. A frame is a slice of audio and can consist of for example 1024 samples. In an audio signal with sampling rate 44.1 kHz, the duration of such a frame would be approximately 23 ms.

Audio features can be used in many different ways, such as music information retrieval and musical genre classification. In this project, they are used for similarity measures and for controlling the parameters of audio effects. Three audio feature extraction tools were used in this project: Aubio (Brossier, 2003), Essentia (Bogdanov et al., 2013) and LibXtract (Bullock, 2007).

2.7 Audio effects

Audio effects are processing techniques that alter audio so it sounds different. In this project, two audio effects are used: Modified hyperbolic tangent and resonant low-pass filter.

Modified hyperbolic tangent is a waveshaping function that can model the characteristics of analog distortion, and especially tube distortion. Modified hyperbolic tangent differs from hyperbolic tangent in that one can model the positive and the negative slopes differently. This distortion effect makes the sound “fuzzier” by adding harmonic components in higher frequencies.

$$mtanh(x) = \frac{e^{ax} - e^{-bx}}{e^{cx} + e^{-dx}}$$

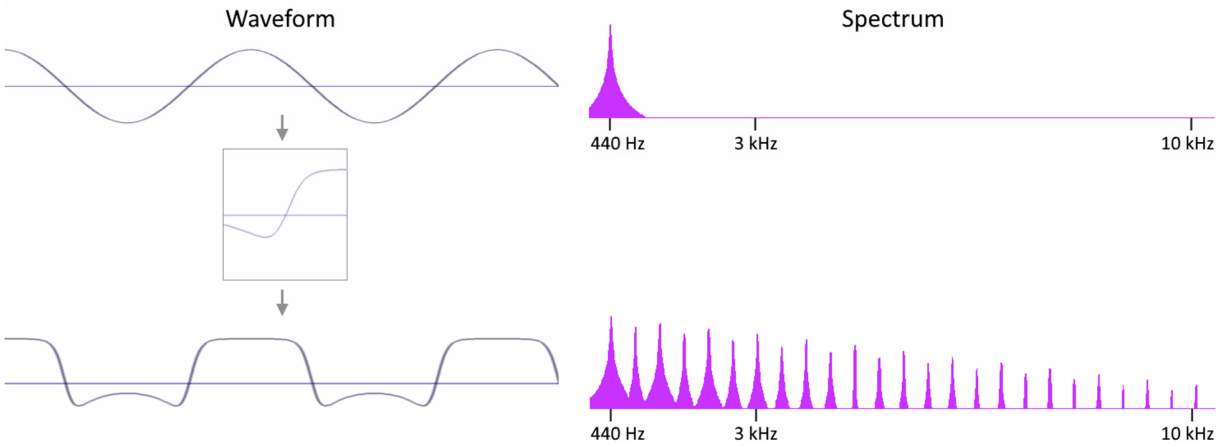


Figure 2.3: Harmonic frequency components are added to a 440 Hz sine wave by means of waveshaping

A low-pass filter attenuates high frequencies and retains low frequencies unchanged. It can be used to make a sound “darker” or “smoother” in timbre. A *resonant* low-pass filter is a low-pass filter that has a peak in the response curve at the cutoff frequency. This quality can be used to boost a single tone in a sound with a rich frequency spectrum. The width of the resonant peak is described by a parameter called Q. As Q increases, the resonance becomes more pronounced.

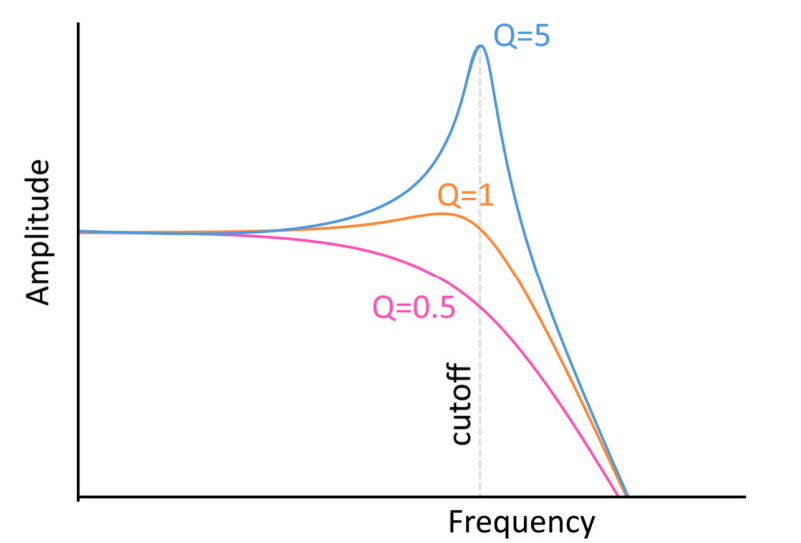


Figure 2.4: Frequency response of a resonant low-pass filter with various Q values

2.8 Audio processing tools

Audio processing is the alteration of audio signals, typically through audio effects. One popular audio processing tool is Csound (“Csound: Sound and music computing system”, n.d.). This is both an audio programming language and a program that runs Csound code. The Csound program takes in a text file of code. This code is executed by the Csound program. The output is sound that is directed to either an audio interface (live) or to a file (non-real-time processing). Csound is used by musicians and composers, typically in experimental electroacoustic music. Traditionally, it has been an offline tool, due to lack of computational power. Today, computational power is sufficient for Csound to run in real-time, so it can be used in live settings such as concerts and sound installations. Csound can not only run on desktop computers, but can also be used as audio processing engine in mobile applications for the operating systems Android and iOS.

3 Methods and implementation

3.1 Evolving neural networks

There are several ways to map sound analysis to effect parameters. The initial idea was a modulation matrix, where each effect parameter becomes a linear combination of the audio features. This idea was quickly iterated upon. Why not use an artificial neural network? An artificial neural network can do the same as a modulation matrix (when the ANN has linear activation functions and no hidden layers), but it can also do more. A neural network can have hidden nodes with various activation functions, which makes more complex signal interactions possible. This widens the scope, and allows for learning higher-level features, such as “the snare and the bass drum are hit simultaneously”.

NEAT (Stanley & Miikkulainen, 2002) is suggested as a technique for evolving neural networks. This is a technique which evolves not only the weights of the neural network, but also the topology, i.e. the number of nodes and the connections between the nodes. HyperNEAT, a technique that is based on NEAT, has been used with great success in a project called Picbreeder (Secretan et al., 2008) and later with some success in a project called SoundBreeder (Ye & Chen, 2014). The purpose of those projects were to evolve visually and aurally appealing art, respectively. Since both those two projects and this project are within the field of creative computation, the hypothesis is that NEAT or HyperNEAT will work well in this project.

3.2 Implementation

Csound has been selected to be the audio processing tool in this project. Since Csound interoperates nicely with Python, and Python is a popular language for artificial intelligence (AI) applications, that was the language of choice for the neuroevolution application. Further, it has been decided that the application should be written in a style compatible with both Python 2 and 3, for the sake of compatibility with various Python libraries. Also, it should work on both Windows and Ubuntu. This way, experiments can not only run on Windows desktop computers, but also on Ubuntu instances in the cloud. The most important dependency is MultiNEAT (Chervenski & Ryan, n.d.), which is one of Kenneth Stanley’s recommended neuroevolution libraries (Stanley, n.d.). It features several neuroevolution algorithms, including NEAT, FS-NEAT and HyperNEAT. It is written in C++, but it has Python bindings, so it is fit for this project. Further, the audio features extraction tools selected for this project are Aubio, Essentia and LibXtract, which are all free open source software written in a compiled language. A link to the author’s implementation of the toolkit, which is open source, is included in appendix A. A complete list of dependencies is included in appendix B.

3.2.1 Performance

A. Eldhuset has previously implemented a program that uses Csound in signal interaction experiments with genetic algorithms. He concluded that his implementation was slow, taking around 5 seconds per individual (Eldhuset, 2015). Hence an experiment with a population size of 20 and 20 generations would take approximately half an hour. The author has analyzed the weaknesses of Eldhuset's approach and come up with a number of techniques to alleviate performance issues:

- Use a templating engine to generate Csound files. It writes csound code to different files, one for each individual in the population. This allows many Csound instances to be run in parallel.

- While a Csound instance runs, it does not have to communicate with another program (a host) via an API. All data needed for the run, including effect parameter values over time, are included inside the csd file
- Use dedicated, compiled audio feature extraction tools such as Aubio instead of Csound for audio feature extraction
- Use the standard streams (stdout) instead of file I/O in audio feature extractors that support this
- Let the host program, Csound and audio feature extractors write files to a RAM disk to avoid slow disk I/O activity
- Use the concept of pipelining to shorten critical paths and enable more parallelism
- Sensible handling of duplicate individuals: when two or more individuals are equal (i.e. their neural networks are equal), evaluate only one of them, and apply the same result to the identical individuals

Incorporating all these techniques, the author's implementation spends around 0.11 seconds on average per individual, given that the dist_lpf effect and the aubio mfcc analyzer is used, the duration of the input sound is 7 seconds, and that the program is run on a modern, high-end laptop with two CPU cores. Hence an experiment with population size 20 and 20 generations may take approximately half a minute.

3.2.2 Neuroevolution routine

The neuroevolution program is called from the command line, with a number of arguments for configuring the experiment. The program then performs roughly these steps:

1. Check sanity of arguments
2. Analyze input sound file and target sound file
3. Initialize a population
4. For each generation, evaluate all individuals, write their data to json files and then advance to the next generation

The evaluation of an individual involves several operations:

1. An artificial neural network is created from the genotype of the individual
2. All feature vectors of the target sound are run through the neural network
3. The neural outputs are scaled to appropriate ranges for the various audio effect parameters
4. Csound runs the input sound through the audio effect that is controlled by the audio effect parameters
5. The resulting sound is run through the audio feature extraction tool(s)
6. The audio features are standardized with the same mean and variance as in the standardization of the target sound audio features
7. The audio features of the target sound and the output sound are used in the fitness function
8. The resulting fitness value is assigned to the individual.

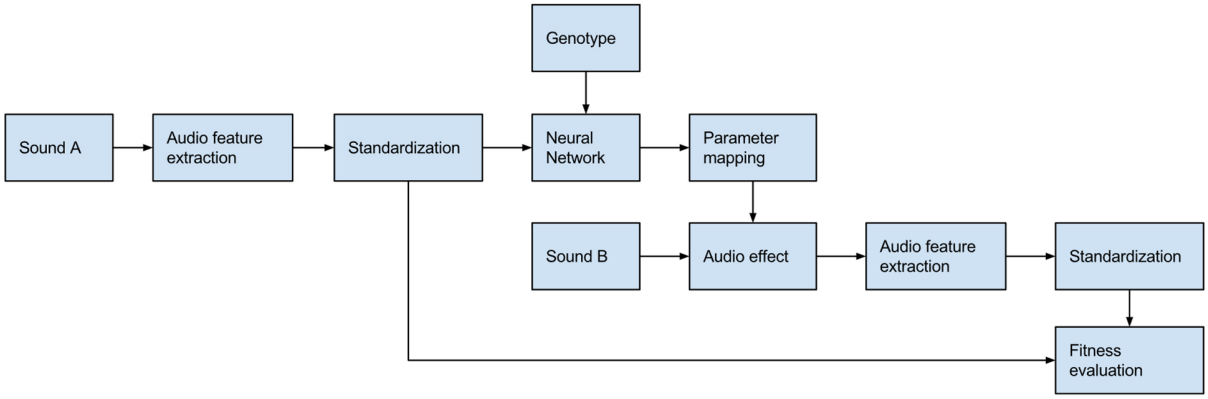


Figure 3.1: Flowchart for the evaluation of an individual

3.2.3 Input standardization

To make the audio features suitable as input to a neural network, they need to be scaled. A simple, but good technique is to standardize them by is subtracting the mean and dividing by the standard deviation (Sarle, 2014). For each audio feature there is one sequence of numbers for the input sound and another sequence for the target sound. The mean and standard deviation for an audio feature is calculated from the input sound's sequence for that feature concatenated with the target sound's sequence for that feature. This mean and standard deviation is then used in all further input standardization. This gives the series the quality of being centered around zero and having a standard deviation of 1 with respect to the input sound and the target sound. Additionally, to avoid extreme values, values are clipped to the range [-4, 4].

3.2.4 Parameter mapping

The output of the neural network is in the range [0, 1], due to the sigmoid activation function in the output layer. These normalized values need to be scaled and skewed appropriately for each effect parameter. The author has used the same mapping function as in Cabbage (Walsh, 2008), a GUI framework for Csound, where slider values are mapped from 0 to 1 to the target range by the following function:

$$f(x) = m_{min} + (m_{max} - m_{min}) * e^{log(x)/s}$$

Where m_{min} and m_{max} are the endpoints of the target range and s is the skew factor. The default skew factor is 1, which will yield a linear mapping. A skew factor of 0.5 will cause the mapping function to output values in an exponential fashion. This is useful for effect parameters such as cutoff frequency.

3.3 Visualization

In very early versions of the neuroevolution program, the author found it hard to evaluate all the data produced during experiments. Therefore an interactive web application for visualizing results was developed. This tool has been very important for being able to understand the strengths and weaknesses of the neuroevolution program during development and research. When the author gained a good understanding of the inner workings and the output of the neuroevolution program, he was able to improve the weak points of the implementation.

The visualization system is a single-page web application written in AngularJS, with various JavaScript libraries for visualizing data. For a complete list of JavaScript libraries that were used in the web

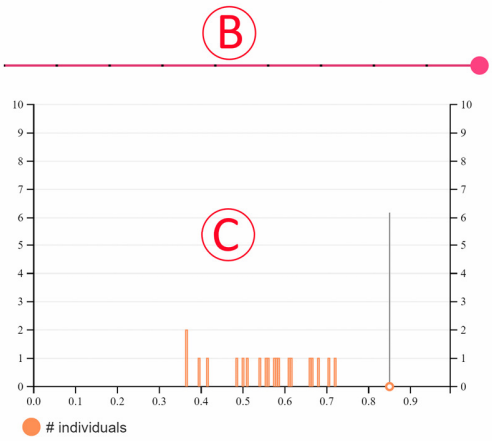
application, see appendix C. The application server is written in NodeJS. The neuroevolution program writes data after each generation, and the NodeJS server listens for these data updates. Whenever new data is available, the updated data is sent via WebSockets to the web application, which then updates its views. Figure 3.2 shows a screenshot of how the web application may look.

Fitness plot

Interaction: Drag with the mouse to pan. Hold alt and drag to zoom. Double-click to reset zoom/pan. Click series in the legend to toggle visibility.

Select generation for histogram

Interaction: While slider is focused, use the arrow keys to navigate



Select individual

sorted by fitness (best to the right)

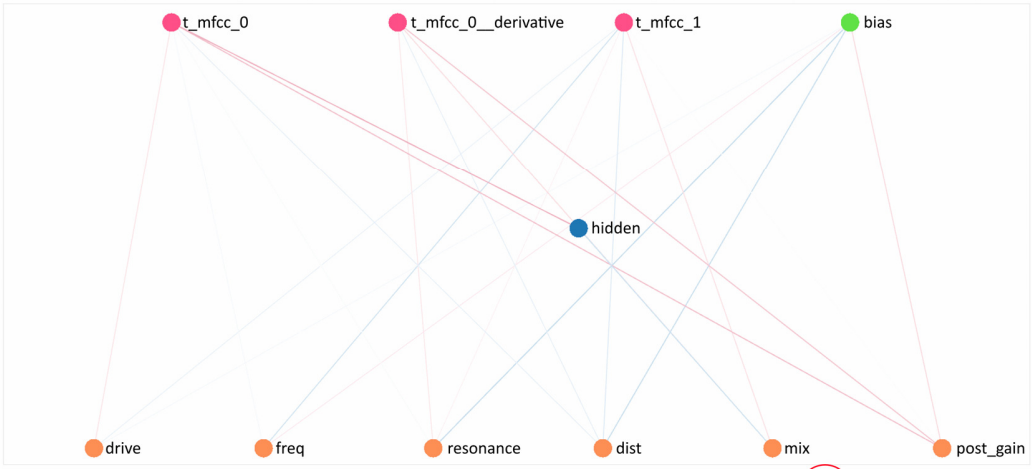
D

Individual f3317d47

Fitness: 0.6144

Born in generation 8

E

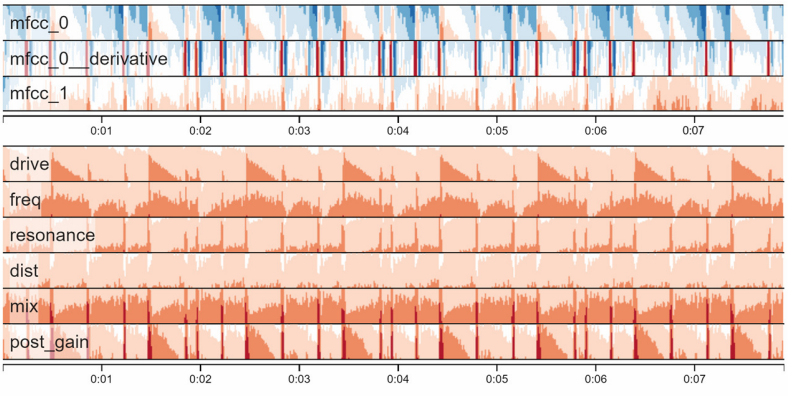


Target (A) Input (B) Output (C) Diff(B, A) Diff(C, A) Diff(C, B)

F

G

PLAY DOWNLOAD Playback rate: 1.00



H

I

Figure 3.2: Annotated screenshot of the interactive visualization system. **A**: Fitness plot (line chart, shows the progress of the GA over the generations). **B**: Interactive slider for selecting a specific generation and visualizing its population. **C**: Fitness histogram (shows fitness distribution in a population). **D**: Interactive slider for selecting a specific individual in the population of the selected generation. **E**: Neural network visualization with edge colors according to weight. Light color means small magnitude while strong color means large magnitude. Red means positive weight while blue means negative. **F**: Select which sound and corresponding data series to visualize. **G**: Audio player with waveform visualization. Useful for playing back the output sound of the selected individual. **H** and **I**: Horizon charts for visualizing neural input and neural output, respectively. Horizon charts make better use of vertical space than standard area charts, allowing you to see many more metrics at-a-glance. Larger values are overplotted in successively darker colors, while negative values are offset to descend from the top. The horizon charts are aligned with the audio player.

4 Experiments and discussion

Three experiments have been run in an attempt to measure the feasibility of using neuroevolution for finding useful mappings from audio features to audio effect parameters. Also, in each experiment, different neuroevolution configurations are compared to find what works better. Each configuration gets 20 runs (with different Pseudo Random Number Generator (PRNG) seeds), and the fitness plots show aggregated values. Table 4.1 shows a rough overview of the experiments and their configuration.

Experiment	Input sound	Target sound	Effect	Parameters compared
1	White noise	Drums	Distortion and resonant low-pass filter	Output activation functions
2	White noise	Drums	Distortion and resonant low-pass filter	Fitness functions
3	White noise	Sine sweep	Resonant low-pass filter	NEAT vs. FS-NEAT

Table 4.1: Overview of the three experiments

4.1 General configuration

Table 4.2 shows the parameters used unless otherwise stated in individual experiments

Parameter	Value
Add neuron probability	0.03
Remove neuron probability	0.03
Add link probability	0.03
Remove link probability	0.06
Elite fraction	0.1
Survival rate	0.25
Allow clones	True
Selection method	Tournament selection
Hidden activation function	Hyperbolic tangent
Output activation function	Sigmoid
Effect parameter low-pass filter cutoff frequency	50 Hz
Population size	20
Number of generations	20

Number of runs	20

Table 4.2: General configuration

4.2 Experiment 1

In this experiment, the toolkit is used to make white noise sound like a drum loop that consists of bass drum and snare drum hits. The mfcc_0 feature is used as neural input and similarity measure. mfcc_0 measures the spectral energy, so it is expected that the amplitude of the output sound will resemble the amplitude of the drum loop sound. Further, three different output activation functions will be compared.

4.2.1 Configuration

Parameter	**Value**
Number of generations	10
Fitness function	Local similarity
Target sound	Drum loop
Input sound	White noise
Effect	Distortion and resonant low-pass filter
Audio features	mfcc_0
Output activation function	[Sigmoid, Linear, Sine]

Table 4.3: Configuration for experiment 1

4.2.2 Fitness function

The local similarity fitness function is based on the average euclidean distance between the feature vector of the target sound and the output sound in the k frames of the two sounds.

```
Function LOCAL_SIMILARITY(target, individual):
       total_euclidean_distance = 0
       for each k in range(num_frames):
           A = target.get_feature_vector(k)
           C = individual.get_feature_vector(k)
           total_euclidean_distance += EUCLIDEAN_DISTANCE(A, C)
       avg_euclidean_distance = total_euclidean_distance / num_frames
       return 1 / (1 + avg_euclidean distance)
```

where EUCLIDEAN_DISTANCE is $d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2}$

4.2.3 Evaluation of configurations

Figure 4.1 shows that the sigmoid activation function yields better results than sine and linear.

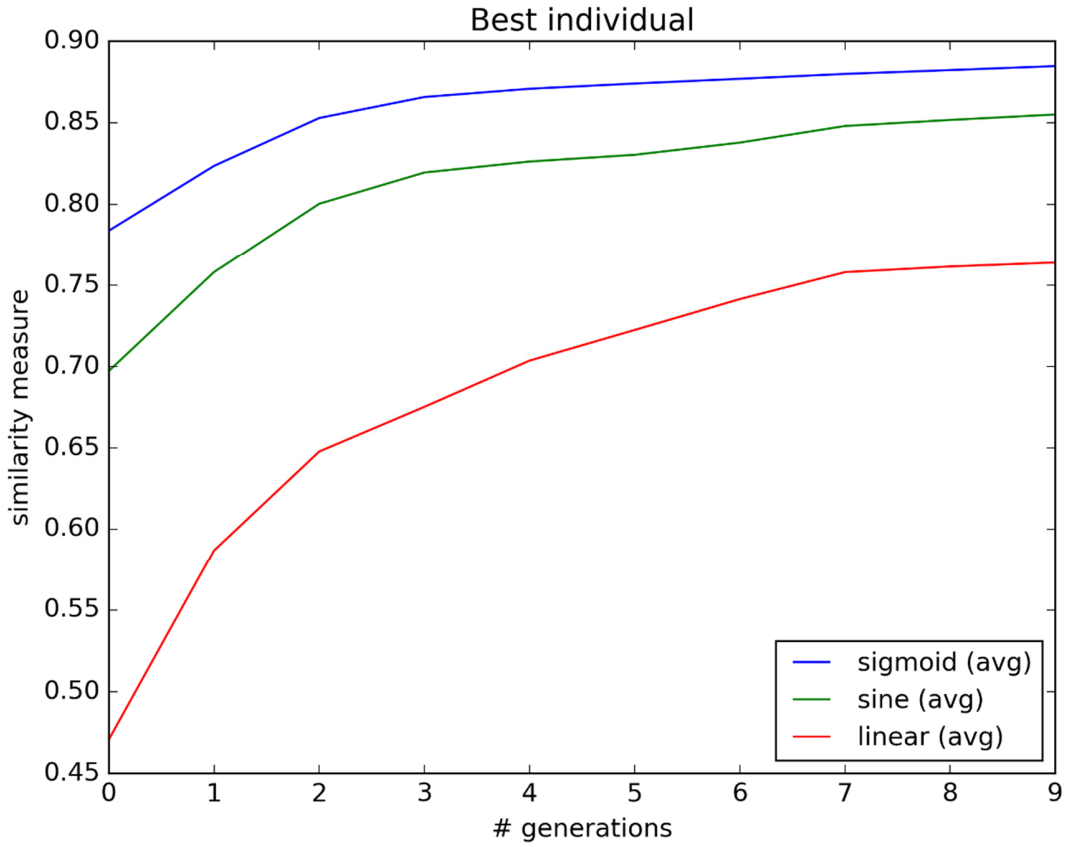


Figure 4.1: Average similarity over 20 runs. Three different output activation functions are compared.

4.2.4 Evaluation of output sound

The results are as expected: The amplitude of the best output sound is quite similar to the amplitude of the target sound. Figure 4.2 shows this in that waveform C looks like waveform A. However, the bass drum hits in the output are not quite as strong as in the target sound. This is because the mfcc_0 feature is not a pure amplitude measure and yields lower values for the bass drum hits than for the snare hits. Figure 4.3 shows this. Further, the distinction between bass drum and snare drum in the output sound is nonexistent. That will be dealt with in experiment 2, by adding more features for neural input and similarity measure.

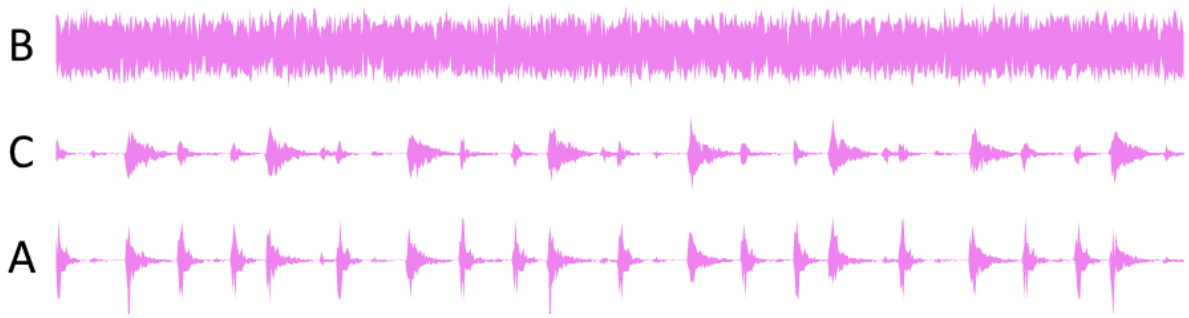


Figure 4.2: Waveform visualizations of input sound (B), output sound (C) and target sound (A).

Figure 4.3: The mfcc_0 feature series of the drums sound visualized in a horizon graph

4.3 Experiment 2

This experiment aims to produce output sounds that have clear distinction between bass drum hits and snare drum hits. This is done by adding more spectral features than in experiment 1. The other goal of the experiment is to try various fitness functions and see which one of them works best.

4.3.1 Configuration

Parameter	Value
Fitness function	[Local similarity, multi-objective, hybrid, novelty, mixed]
Target sound	Drum loop
Input sound	White noise
Effect	Distortion and resonant low-pass filter
Audio features	mfcc_0, mfcc_1, spectral_centroid, bark_2, bark_10, bark_18

Table 4.4: Configuration for experiment 2

4.3.2 Fitness functions

Local similarity

This is defined in experiment 1.

Multi-objective optimization

This fitness function is inspired by NSGA-II (Deb, Pratap, Agarwal, & Meyarivan, 2002). It incorporates two measures: rank and crowding distance. These are concepts taken directly from the NSGA-II paper, and they are then used in a math expression that satisfies these two constraints that are used in NSGA-II:

rank(A) > rank(B) => fitness(A) > fitness(B)

rank(A) = rank(B), crowding_distance(A) > crowding_distance(B) => fitness(A) > fitness(B)

The ranks of the individual are calculated by doing non-dominated sort. Crowding distance is computed between individuals in a given rank. The multi objective fitness function works like this:

```
Function MULTI_OBJECTIVE(target, individuals):

    for individual in individuals:

        CALCULATE_OBJECTIVES(individual, target)

    fronts = FAST_NON_DOMINATED_SORT(individuals)

    for rank in fronts:

        CALCULATE_CROWDING_DISTANCE(fronts[rank])  # assigns individual.crowding_distance

        for individual in fronts[rank]:

            individual.fitness = 1.0 / (rank + (0.5 / (1.0 + individual.crowding_distance)))



Function CALCULATE_OBJECTIVES(individual, target):

    individual.objectives = {}

    for feature in similarity_features:

        individual.objectives[feature] = EUCLIDEAN_DISTANCE(target.analysis[feature], output.analysis[feature])
```

Pseudocode for `FAST_NON_DOMINATED_SORT` and `CALCULATE_CROWDING_DISTANCE` can be found in the NSGA-II paper (Deb, Pratap, Agarwal, & Meyarivan, 2002).

Hybrid

While NSGA-II is good at optimizing for non-dominated individuals, these individuals may be extreme tradeoffs and therefore not necessarily feasible solutions in practice. In order to reward good tradeoffs more, the author developed the hybrid fitness function. This fitness function returns the average of `MULTI_OBJECTIVE` and `LOCAL_SIMILARITY`.

Novelty search

Novelty search (Lehman & Stanley, 2008) ignores the objective and optimizes for novelty instead. The reason that this may work well is that in some problems the intermediate steps to the goal do not resemble the goal itself. When it comes to implementation, MultiNEAT has novelty search built-in, but Python bindings for it are missing, so the author could not use it in his Python application. However, novelty search can be implemented on top of most evolutionary algorithms, by using a fitness function that rewards novelty (Lehman & Stanley, 2015), so that is what the author did. First, each individual needs to be represented as a vector that describes its characteristics. This vector is constructed by concatenating all audio feature series of the individual. The implemented fitness function assigns high fitness values to the individuals that have long euclidean distances from the 3 nearest neighbours, where the neighbours are individuals that have been evaluated earlier. The very first population gets random fitness values, because there are no earlier individuals to measure distance from.

Mixed

This fitness function is simple: For each generation, one of the following fitness functions is chosen randomly and applied: local similarity, multi-objective, hybrid, novelty. The idea behind this fitness

function is to create a dynamic fitness landscape, where the individuals that get good scores from all fitness functions have the greatest chance of survival over time.

4.3.3 Evaluation of configurations

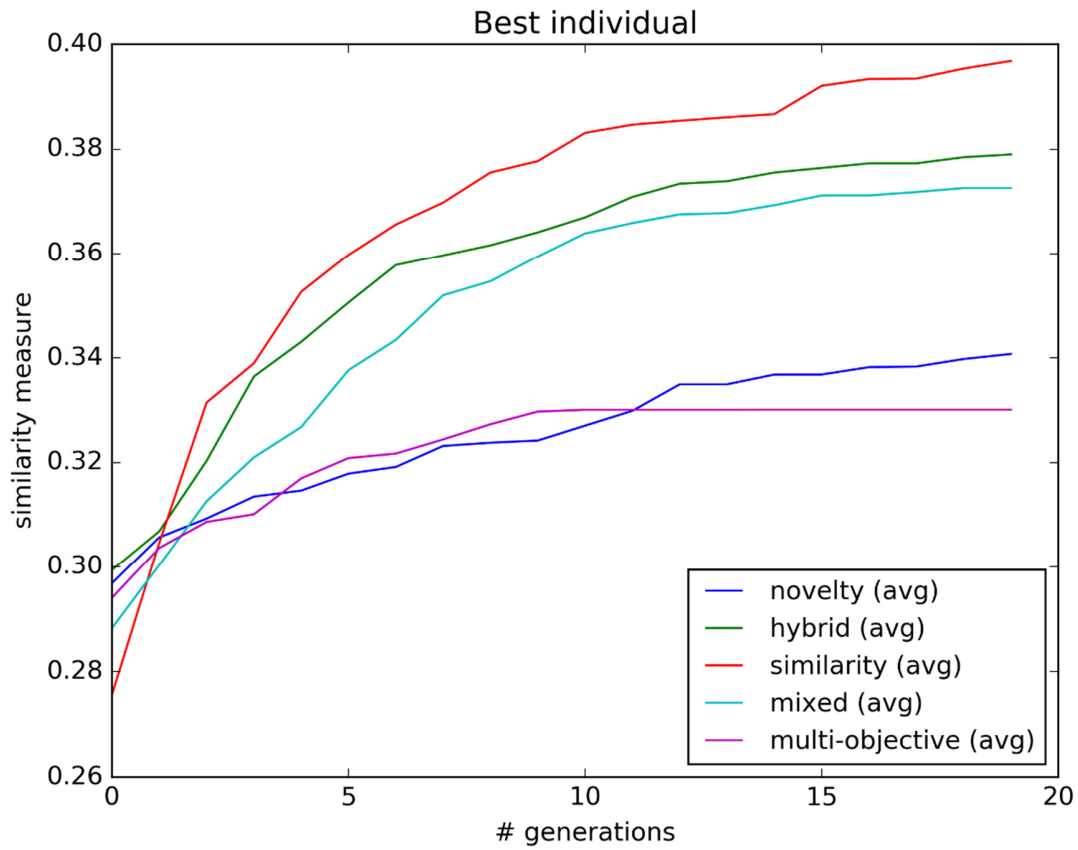


Figure 4.4: Best individual on average over 20 runs, with 20 generations for each run. The legend shows which fitness function each line correspond to.

The y-axis of figure 4.4 is the score returned by LOCAL_SIMILARITY. That's probably the reason why the local similarity fitness function gets the best scores on this plot. It doesn't necessarily mean that local similarity is the best fitness function. One problem with local similarity is the lack of diversity in populations. Almost all individuals in the final population have similar characteristics. Multi-objective and hybrid don't have this problem. In fact, the individuals of the final population have different characteristics in runs with multi-objective or hybrid fitness. This gives the user more choice. Generally, the author finds that hybrid is better than multi-objective, because hybrid yields better tradeoffs. One problem with multi-objective is that for example one individual has good-sounding snare drum hits, but the bass drum hits sound bad or are too weak.

Novelty search does find good solutions, but it does not necessarily find the best solutions. However, if we define "best" as "novel", then novelty search is the best fitness function. It can be used for finding novel cross-adaptive audio effects. It can also be used to explore the entire search space. However, it

requires a human for evaluating the musical quality of the results. This is also the case for the mixed fitness function, which yields varying results due to its noisy, dynamic fitness landscape.

4.3.4 Evaluation of output sound

The best output sound, as evaluated by the author, is produced by one of the runs with hybrid fitness. The sound resembles the drum loop, and one can easily hear the difference between the bass drum and the snare drum. The spectral centroid analysis of the sound verifies this, as it goes up and down according to which drum is hit. See figure 4.5.

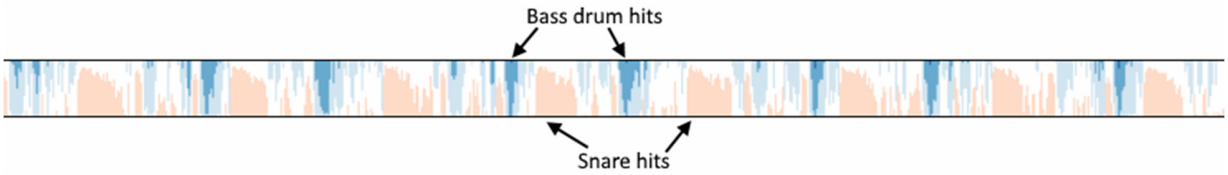


Figure 4.5: Spectral centroid of the best individual in experiment 2

4.4 Experiment 3

This experiment is mainly about dealing with high-dimensional data. 68 different audio features are used for neural input and similarity calculation. Also, the low-pass filter effect (without distortion) is used, because distortion is not needed for transforming noise into a sine sweep. The low-pass filter effect has three parameters. A fully connected neural network with no hidden nodes would have 68 * 3 = 204 weights. To deal with the situation effectively, some techniques will be applied:

- Use FS-NEAT (Whiteson, Stone, Stanley, & Miikkulainen, 2005). This NEAT variation starts with just a few connections and gradually adds/removes connections. NEAT will be compared with FS-NEAT.
- Give more weight to important features (such as mfcc_0 and spectral_centroid) and less weight to other features. These weights are used in the similarity calculations. All the weights are included in appendix D.

4.4.1 Configuration

As this experiment is more about finding the correct connections than finding the correct weights for the connections, the probabilities for adding and removing links need to be tweaked. To find a good value, a few settings have been tried in runs with FS-NEAT. Based on the results seen in table 4.5, 0.3 seems to be the best value, so that is what will be used.

Add/remove link probability	Average fitness of best individual after 20 generations
0.03	0.398
0.1	0.392
0.2	0.418
0.3	**0.421**
0.4	0.414

0.5	0.406

Table 4.5: Results from trying different values for add/remove link probability. Fitness values are the average of 10 runs.

Parameter	**Value**
Fitness function	Local similarity, as defined in experiment 1
Target sound	Logarithmic sine sweep from 40 Hz to 15 kHz
Input sound	White noise
Effect	Resonant low-pass filter
Audio features	68 features. See appendix D for a complete list.
Population size	24
Add/remove link probability	0.3

Table 4.6: Configuration for experiment 3

4.4.2 Evaluation of configurations

Figure 4.6 shows that FS-NEAT performs better over time, and does not get stuck as much as NEAT does. It also shows that the best run with FS-NEAT is significantly better than the best run with NEAT.

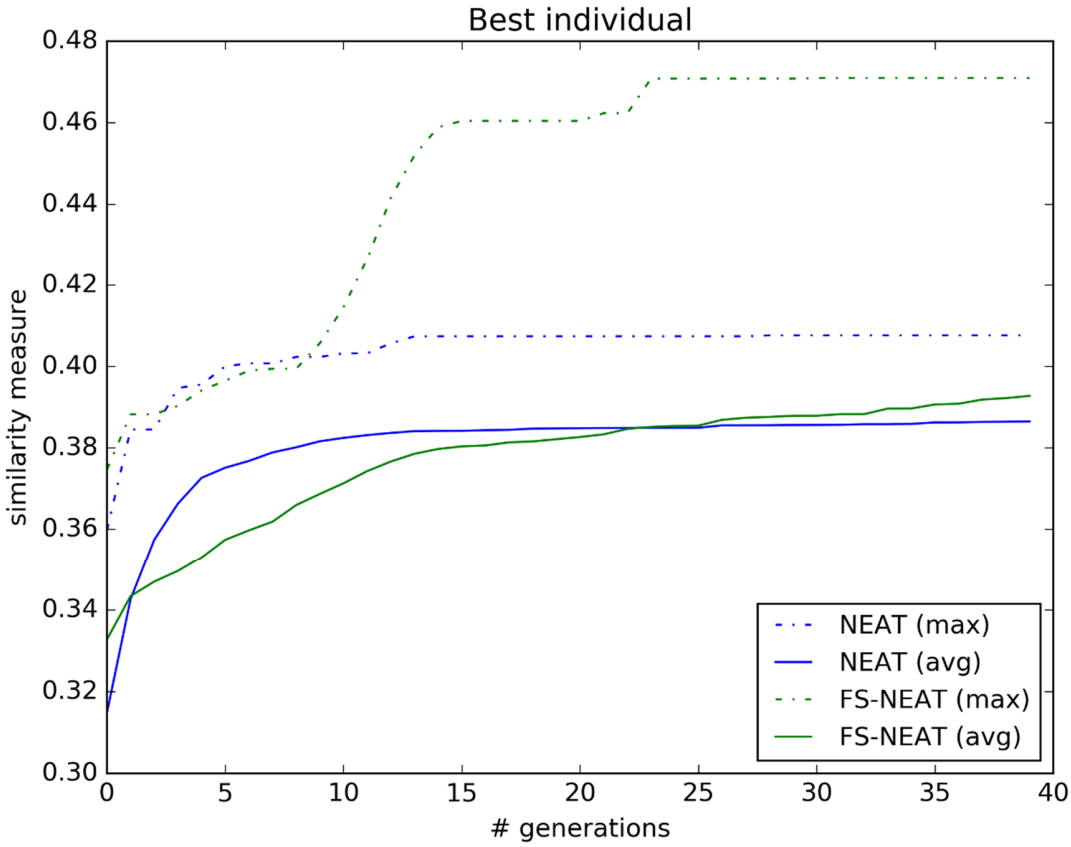


Figure 4.6: Similarity measure for NEAT vs. FS-NEAT experiments. Average of 20 runs.

4.4.3 Evaluation of output sound

Because the fitness function doesn't quite capture how feasible a solution is, the output sounds are evaluated qualitatively. A solution must satisfy all the following constraints to be deemed feasible:

- Clear tone (high filter resonance)
- No pops or glitchy noises
- Smooth frequency sweep from low to high
- Amplitude must not be too low

The author listened to the sound with the highest fitness value from each run. 2 of the 20 runs with FS-NEAT yielded at least one feasible solution, while none of the runs with regular NEAT yielded any feasible solutions. The main problem with classic NEAT is that it tries to pass signals from almost all 68 features to the audio effect parameters. This is an issue because it results in a very noisy parameter control, so the output sounds become glitchy and not very musically interesting. Therefore, one can safely conclude that FS-NEAT performs better than NEAT in this experiment.

5 Conclusion

It has been shown that using neuroevolution for finding useful mappings in cross-adaptive audio effects is feasible. Several fitness functions have been developed and compared. Based on qualitative evaluations, the hybrid variant, that is a combination of local euclidean distance and NSGA-II-inspired multi-objective optimization, has been found to yield the best results. Furthermore, in experiments with high-dimensional spaces, FS-NEAT has been proven to do better than NEAT, because FS-NEAT chooses only a few useful connections rather than a fully connected neural network. A comprehensive toolkit has been developed during the course of the project. The toolkit includes an interactive visualization tool that makes it easier to evaluate results and understand the neuroevolution process. The toolkit has lots of configuration options, enabling a flexible platform for experimentation. It is open source, and is expected to live on and be used in future research within the field of cross-adaptive audio effects.

5.1 Future work

As stated in the introduction, current research at the Music Technology department at Norwegian University of Science and Technology aims at exploring radically new modes of musical interaction in live music performance. This project is a good start, but there is still a lot to be explored. For example, it would be interesting to try other effects than distortion and resonant low-pass filter. One related idea is to enable the neuroevolution process to be more creative by letting it choose from a pool of audio effects. Further, it should be able to add multiple audio effects and decide the order in which they are applied.

While an evolved neural network may perform well on the input sound and target sound it was trained on, it is desirable to be able to try the same neural network on other sounds to see how well it generalizes. In particular, it would be interesting to try it in live music performances. This requires some architectural changes, as audio needs to be analyzed in real time, preferably with low latency.

Picbreeder (Secretan et al., 2008) and Soundbreeder (Ye & Shen, 2014) had success with HyperNEAT, which is one variant of NEAT that has not been tried in this project. HyperNEAT might be able to produce better results than NEAT in this project, so that's worth exploring.

While five different fitness functions have already been developed and compared, the author is fairly certain that further research on this could yield improvements. For example, a (recurrent) neural network that is trained to classify a few classes of sounds could be used for measuring fitness.

The author imagines that methods developed in this project could be used for mastering audio and also for novel crossfading in DJ mixing software. However, that would require smart methods for dealing with long sounds (several minutes). This project has only dealt with short sounds (a few seconds) so far. When dealing with longer sounds the author sees two challenges: 1) Computational time and 2) A long sound might have several very different parts, and the evolved neural network might have trouble dealing well with all of them. One possible solution to these challenges is to chop the long sound into a few short audio segments that represent the different parts of the sound well and then run the program on each audio segment. When applying the resulting neural networks on new sounds, the program can automatically fade between the evolved artificial neural networks based on similarity with the various audio segments they were trained on.

Acknowledgements

I express gratitude towards my supervisors Øyvind Brandtsegg and Gunnar Tufte for guidance and valuable feedback during this specialization project. I would also like to the Department of Computer and Information Science, NTNU for providing me with a Linux Virtual Machine to run experiments on. Thanks to students at the office and my girlfriend for supporting me and listening to my ramblings about sounds and genetic algorithms. Thanks to Sigve S. Farstad for valuable feedback. Lastly, I would like to thank contributors to the various open source software and libraries that have been used throughout the project.

References

Bäck, T. (1996). *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms*. New York: Oxford Univ. Press.

Bogdanov, D., Wack, N., Gómez, E., Gulati, S., Herrera, P., Mayor, O., . . . Serra, X. (2013). ESSENTIA: An Audio Analysis Library for Music Information Retrieval. *International Society for Music Information Retrieval Conference (ISMIR '13),* 493-498.

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

Brossier, P. (2003). Aubio, a library for audio labelling. Retrieved June 13, 2016, from http://aubio.org/

Bullock, J. (2007). LibXtract: A lightweight library for audio feature extraction. *Proceedings of the International Computer Music Conference*.

Cartwright, M., & Pardo, B. (2014). SynthAssist: Querying an Audio Synthesizer by Vocal Imitation. *Proceedings of the ACM International Conference on Multimedia - MM '14*. doi:10.1145/2647868.2654880

Caudill, M. (1987). Neural networks primer, part I. *AI Expert, 2*(12), 46-52.

Chervenski, P., & Ryan, S. (n.d.). MultiNEAT neuroevolution library. Retrieved June 13, 2016, from http://www.multineat.com/

Csound: Sound and music computing system. (n.d.). Retrieved June 13, 2016, from http://csound.github.io/

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation IEEE Trans. Evol. Computat., 6*(2), 182-197. doi:10.1109/4235.996017

Eldhuset, A. W. (2015). Experiments in Using Genetic Algorithms to Find Parameters for Adaptive Audio Effects. Unpublished specialization project, Norwegian University of Science and Technology

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Massachusetts: Addison Wesley.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks, 4*(2), 251-257. doi:10.1016/0893-6080(91)90009-t

Lecun, Y., Bottou, L., Orr, G. B., & Müller, K. -. (1998). Efficient BackProp. *Lecture Notes in Computer Science Neural Networks: Tricks of the Trade,* 9-50. doi:10.1007/3-540-49430-8_2

Lehman, J., & Stanley, K. O. (2008). Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. *Proceedings of the Eleventh International Conference on Artificial Life (ALIFE)*. Retrieved from http://eplex.cs.ucf.edu/papers/lehman_alife08.pdf

Lehman, J., & Stanley, K. O. (15, December 5). Novelty Search Users Page. Retrieved June 15, 2016, from http://eplex.cs.ucf.edu/noveltysearch/userspage/

Logan, B. (2000). Mel Frequency Cepstral Coefficients for Music Modeling. Retrieved June 16, 2016, from http://musicweb.ucsd.edu/~sdubnov/CATbox/Reader/logan00mel.pdf

Mermelstein, P. (1976). Distance measures for speech recognition, psychological and instrumental. *Pattern Recognition and Artificial Intelligence,* 374-388.

Moritz, A. (2003). Stockhausen Hymnen Introduction. Retrieved June 13, 2016, from http://home.earthlink.net/~almoritz/hymnenintro.htm

Reiss, J. D. (2011). Intelligent systems for mixing multichannel audio. *2011 17th International Conference on Digital Signal Processing (DSP)*. doi:10.1109/icdsp.2011.6004988

Sarle, W. (2014, March 27). Comp.ai.neural-nets FAQ, Part 2 of 7: LearningSection - Should I normalize/standardize/rescale the data? Retrieved June 13, 2016, from http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-16.html

Secretan, J., Beato, N., D'Ambrosio, D., Rodriguez, A., Campbell, A., & Stanley, K. (2008). Picbreeder: Evolving Pictures Collaboratively Online. *Proceeding of the Twenty-sixth Annual CHI Conference on Human Factors in Computing Systems - CHI '08*. doi:10.1145/1357054.1357328

Stanley, K. O., & Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation, 10*(2), 99-127. doi:10.1162/106365602320169811

Stanley, K. (n.d.). NEAT Software Catalog. Retrieved June 13, 2016, from http://eplex.cs.ucf.edu/neat_software/#NEAT

Veldhuizen, D. A., & Lamont, G. B. (2000). Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art. *Evolutionary Computation, 8*(2), 125-147. doi:10.1162/106365600568158

Verfaille, V., Zolzer, U., & Arfib, D. (2006). Adaptive digital audio effects (a-DAFx): A new class of sound transformations. *IEEE Transactions on Audio, Speech and Language Processing IEEE Trans. Audio Speech Lang. Process., 14*(5), 1817-1831. doi:10.1109/tsa.2005.858531

Walsh, R. (2008). Cabbage, a new GUI framework for Csound. Retrieved June 16, 2016, from http://lac.linuxaudio.org/2008/download/papers/7.pdf

Werbos, P. J. (1982). Applications of advances in nonlinear sensitivity analysis. *System Modeling and Optimization Lecture Notes in Control and Information Sciences,* 762-770. doi:10.1007/bfb0006203

Whiteson, S., Stone, P., Stanley, K. O., Miikkulainen, R., & Kohl, N. (2005). Automatic feature selection in neuroevolution. *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation - GECCO '05*. doi:10.1145/1068009.1068210

Ye, J., & Chen, S. (2014). SoundBreeder with MultiNEAT. Retrieved June 13, 2016, from https://www.cs.swarthmore.edu/~meeden/cs81/s14/papers/AndyLucas.pdf

Appendix A: Open source toolkit

The toolkit that has been implemented in this project is open source and available at https://github.com/iver56/cross-adaptive-audio

This includes software for doing neuroevolution on sounds as well as the software for visualizing the experiments. The readme file includes:

- Detailed instructions for how to install the project and all dependencies on Windows and Ubuntu
- User manual with example commands for getting started

Appendix B: Neuroevolution application dependencies

Name	Description
Jinja2	A general purpose templating language. Useful for generating csd files that are fed into Csound.
six, futures	Makes the Python application compatible with both Python 2 and Python 3
whichcraft	A tool that can check which programs are installed on the computer. Used for starting Node.js correctly on both Ubuntu and Windows.
natsort	Natural sorting. Used for showing audio features in the correct order.
nose	A tool for running all the automated tests in the project
statistics	A package that can calculate various statistics, for example standard deviation, of data series.
numpy	A package for scientific computing. Used for computing gradients and euclidean distance.
natplotlib	A 2D plotting library. Used for creating similarity plots for this report.
MultiNEAT	A portable neuroevolution Library written in C++. It has Python bindings.
Sonic Annotator with Vamp plugin LibXtract	Sonic annotator takes in a set of audio files, runs them through the specified vamp plugin and outputs the results. LibXtract is a library for audio feature extraction.
Aubio mfcc	A tool that takes in a single audio file and calculates and outputs MFCC coefficients for each frame
Essentia music extractors	A tool that takes in a single audio file, calculates a large set of audio features and writes the resulting data to a file
Csound	A sound and music computing system. Csound can be given a piece of Csound code that describes how to process audio, and then Csound processes the audio accordingly. In this project, Csound is used for applying audio effects with effect parameters that vary over time.

Appendix C: JavaScript libraries in interactive visualization application

Name	Description
NodeJS	Used for serving the application and pushing results to the application via websockets whenever new data becomes available
AngularJS	Application framework that makes it easy to build Single-Page Applications
Angular-material	User Interface (UI) Component framework. Makes it easy to add UI elements, such as buttons and sliders, that look nice and have great usability.
Color-brewer	Various sets of colors that are useful for visualizing data
Cubism	Time series visualization in the form of horizon charts, which reduce vertical space without losing resolution. This is useful when there are many variables to visualize simultaneously in a limited vertical space.
n3-line-chart	Used for line charts and histograms. Is nicely integrated with AngularJS and features some useful interactions. Depends on D3.js
Debounce, limit	Used for throttling the refresh rate of computationally demanding actions
Sigma	Used for visualizing neural networks. Features zooming and panning.
Wavesurfer	Works as an audio player that also visualizes the waveform of the sound that is played. The user can click on the waveform to seek to that position.
jQuery	Makes it easier to do Document Object Model (DOM) manipulation

Appendix D: Experiment 3 audio features and weights

Audio feature	Weight	Analyzer
mfcc_0	0.1	Aubio MFCC
mfcc_1	0.02	Aubio MFCC
spectral_centroid	4.0	libXtract
bark_0, bark_1, …, bark_25	0.02	libXtract
tristimulus_1	0.02	libXtract
tristimulus_2	0.02	libXtract
tristimulus_3	0.02	libXtract
spectral_flux	0.02	libXtract
barkbands_crest	0.02	Essentia
barkbands_flatness_db	0.02	Essentia
barkbands_kurtosis	0.02	Essentia
barkbands_skewness	0.02	Essentia
barkbands_spread	0.02	Essentia
dissonance	0.02	Essentia
erbbands_crest	0.02	Essentia
erbbands_flatness_db	0.02	Essentia
erbbands_kurtosis	0.02	Essentia
erbbands_skewness	0.02	Essentia
erbbands_spread	0.02	Essentia

melbands_crest	0.02	Essentia
melbands_flatness_db	0.02	Essentia
melbands_kurtosis	0.02	Essentia
melbands_skewness	0.02	Essentia
melbands_spread	0.02	Essentia
pitch_salience	0.02	Essentia
silence_rate_20dB	0.02	Essentia
silence_rate_30dB	0.02	Essentia
silence_rate_60dB	0.02	Essentia
spectral_complexity	0.02	Essentia
spectral_decrease	0.02	Essentia
spectral_energy	0.02	Essentia
spectral_energyband_high	0.02	Essentia
spectral_energyband_low	0.02	Essentia
spectral_energyband_middle_high	0.02	Essentia
spectral_energyband_middle_low	0.02	Essentia
spectral_entropy	0.02	Essentia
spectral_kurtosis	0.02	Essentia
spectral_rms	0.02	Essentia
spectral_rolloff	0.02	Essentia
spectral_skewness	0.02	Essentia
spectral_spread	0.02	Essentia
spectral_strongpeak	0.02	Essentia
zerocrossingrate	0.2	Essentia