Music recommendations with deep learning

Carl Rynegardh

June 2019

Abstract

In this thesis we apply deep reinforcement learning to the problem of recommending music. A content-based approach is taken, and features from music is extracted with a pretrained deep learning music-tagger. For training, user-interactions are simulated.

Acknowledgement

A special thanks to Jan Eric Larsson and Fredrik Edman at Lund University for all the conversations and excellent feedback. Without this feedback, th thesis would have been hard to finish.

Contents

1 Introduction

1.1 Motivation

Music is all around us and an important aspect in the life of many people. The last 20 years the music industry have changed massively. Twenty years ago consumers usually bought CDs in a store, but with time downloading became more common, and finally, music streaming applications such as Spotify became popular. Today thousands or even millions of songs can be accessed with a few clicks on a computer. With so much music to choose from, users benefit from receiving recommendations for music they may like.

An example of an intelligent service for recommendations is the dicover weekly functionality of Spotify. Every week a new set of songs is published tailored to the specific user. This functionality helps the user explore new songs, often from artists that are unknown to the user. Other recommendation functionalities could be recommending music based on mood or activity. Many of these systems are based on machine learning, and requiring the training of parameters to produce accurate recommendations.

Traditional recommendation systems often focused on what other users listened to, and less on the content. If nobody previously listened to a song, that soung would not be recommended to users at all. For music recommendations, the content is the music itself. Modern approaches can use both content and user data to make recommendations. When the focus is on the content instead of what other users listen to, artists that are unknown but produce quality music benefits since they have a higher chance of being recommended.

In this thesis we assume that no previous user data exist, which is the case for many companies in the beginning. If no user data exists, assumptions of user behaviour could be made together with using the content of the music to make recommendations.

Following in this Section we present the aim, an overview of recommendations and how they apply to music.

In Section 2 information in music and how to extract the information is firstly described. Secondly, relevant literature from audio music similarity retrieval and playlist generation is presented. In the end, we draw conclusions which the solution presented is based upon.

In Section 3, the problem aimed to solve is narrowed down and an overview to the proposed solution is presented.

Theory of reinforcement learning is explained the beginning of Section 4. Following the theory, details on the training environment, implementation and experiments is presented.

In Section 5 features are extracted with deep learning. Finally, in Section 6, the recommendation system is put together, and evaluated on a dataset.

1.2 Aim

The thesis aims to improve recommendations for music based on content, when no previous user data is available. A system is designed and implemented that can make music recommendations based on content and by simulating user interactions.

1.3 Background - Recommending music

Recommender systems have existed for a long time, and people are interacting with them every day. There are many ways to make recommendations, and below three conventional approaches of making recommendations are described.

Collaborative filtering

Collaborative filtering is probably the most popular type of recommendation system. The idea is to recommend songs to a user which similar users preferred in some way. Collaborative filtering has, however, two well-known issues. One is the cold start problem, which means that if there is no or very little user data, accurate recommendations is not possible. The second issue is popularity bias, which means that popular items have a very high chance of being recommended and new unknown items have a slim chance of being recommended.

Content-based filtering

In content-based recommendation systems, songs are recommended based on the content. In music, this means recommending items based on the audio content, often by recommending similar songs. First, audio has to preprocessed and then features extracted from the preprocessed audio. An issue with this approach, is that music needs to be accurately represented in a vector space. If the music is not accurately represented, then songs that sound dissimlar could be seen as similar by the system.

What information in music is, and how to extract features is described in Section 2.

Hybrid-based filtering

Hybrid systems can use several sources of information for recommendations. It could be a combination of collaborative filtering and content for example. A simple example would be a recommendation system that now and then recommends an unknown song to a user based on the content of the song. It could be a song that is similar in content to other songs the user previously rated high. If the user likes the recommended song, similar users could receive the same recommendation. While hybrid-based approaches can experience the same issues as the collaborative based and content based approaches, hybrid-based approaches can also overcome them by combining the approaches efficiently.

Music recommendations in this thesis

In music we want to find similar songs to users prefered songs. Thus, it is possible to divide the problem into two parts. The first part would be finding songs the user enjoys, and the second part recommending similar songs.

With no previous information about the user, the first thing is to find out which preference of music the user has, which for instance could be done by exploring different songs. This can be done by playlist generation(PG) where the goal is to create a playlist of songs a user enjoys. A part of creating a playlist, is to find songs the user enjoys.

At a specific point in time the user tend to listen to similar music depending on mood, activity, time of day, etc. Users also have different kind of taste. Some enjoy Jazz, while others like rock and so on. Recommending songs that are similar based on the content one listens to, should therefore, in theory produce good recommendations.

Recommending similar songs basically means, recommending songs that have been calculated to have a small distance between so called features of the song. This puts us in the field of Audio Music Similarity Retrieval(AMSR) since we are retrieving songs similar to a query song and its respective features, Section 2.4.

2 Background

2.1 Information in music

Domain knowledge is often important in machine learning. Without it, it is hard to know what is informative in the data and what is not. In this section we will go through what information in music is, how it can be extracted and how it relates to machine learning. First in 2.1.1 we go through aspects of music, and in the following subsections describe how features can be extracted from the audio signal.

2.1.1 Elements, aspects, of music

A common way to describe sound[1] is to divide it into the six categories pitch, duration, loudness, timbre, texture and spatial location. Other important aspects of music are tempo, melody, harmony, rhythm and beat. For simplicity all of these are mentioned as different aspects of music in the rest of thesis.

Some aspects such as pitch are dependent on the frequency of the notes played, while others such as rhythm are temporal aspects.

Pitch The perceptual property of sounds that allows the ordering on a frequency-related scale. Pitch is basically the frequency of sound that is perceived as the height of a tone. A higher frequency corresponds to a higher tone.[2]

[1] `https://en.wikipedia.org/wiki/Elements_of_music`
[2] `https://en.wikipedia.org/wiki/Pitch_(music)`

Loudness The attribute of auditory sensation in terms of which sounds can be ordered on a scale extending from quiet to loud. [3]

Timbre Timbre distinguishes the sound between different types of sound production, e.g choir voices and different musical instruments. Given two instruments playing at the same pitch and loudness, timbre is what makes a particular musical sound have a different sound from another. [4]

Duration The amount of time a tone, pitch, is played.[5].

Texture How the tempo, melodic and harmonic materials are combined in a composition.[6]

Melody A linear succession of musical tones that the listener perceives as a single entity. A combination of pitch and rhythm.[7]

Harmony How the composition of individual sounds, or superpositions of sounds, is analysed by hearing e.g simultaneously occurring frequencies, pitches or chords.

Rhythm The placements of sounds in time, or an ordered alternation of contrasting elements.[8]

Beat The basic unit of time in music. Oftened defined as the rhythm of a person listening to music would tap their toes.[9].

Tempo The speed or pace of a given piece and is usually measured in beats per minute[10]

These aspects exist in every song, but are not of equal importance. When dancing beat becomes a very important aspect while others less. For human perception of similarity in music, timbre is known to be the most important aspect, in most cases. Likely because of its ability distinguish instruments playing at the same pitch.

2.2 Frame level features

Frame level features are low-level features and can be considered to be "basic", or simple features. Different frame level features exist, which captures different aspects of music.

[3]American National Standards Institute, "American national psychoacoustical terminology" S3.20, 1973, American Standards Association.
[4]`https://en.wikipedia.org/wiki/Timbre`
[5]`https://en.wikipedia.org/wiki/Duration_(music)`
[6]`https://en.wikipedia.org/wiki/Texture_(music)`
[7]`https://en.wikipedia.org/wiki/Melody`
[8]`https://www.britannica.com/art/rhythm-music`
[9]`https://en.wikipedia.org/wiki/Beat_(music)`
[10]`https://en.wikipedia.org/wiki/Tempo`

Mel frequency cepstral coeffiecents (MFCC) is known to represent the timbre aspect of music. Another example of frame level features is the Chroma features, but since Chroma features is known to represent pitch it is of less importance in this thesis. As mentioned previously, timbre is an aspect when comparing human perception of similarity between music.

When computing the MFCC features other representations are created that also often are used for various applications. Therefore, we divide the creation of MFCC into three parts. First, the computation of the spectrogram, which is presented in Section 2.2.1. Secondly, the spectrogram is transformed to represent human perception of sound. This representation is called Mel-spectrogram and is presented in Section 2.2.2 At last, dimensionality reduction is applied to the Mel-spectrogram yielding MFCC features as show in Section 2.2.3.

2.2.1 Spectrogram

As mentioned, the first thing necessary is to transform the raw audio signal from the time domain to the frequency domain. For digital signals, this is computed with the Discrete Fourier Transform(DFT) and generally calculated with the Fast Fourier Transform(FFT). A sliding window is applied, with a defined hop length. For every position the sliding window takes, a window function is applied. Generally, the Hanning window is chosen. DFT is then calculated on the output of the window function. The mathematical formula for the Hanning window[18], is,

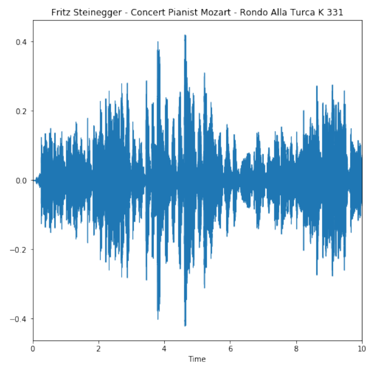
$$w(n) = 0.5(1 - cos(\frac{2\pi(n-1)}{N})), 0 \leq n < N$$

where N is the length of the array, of the digital audio signal.

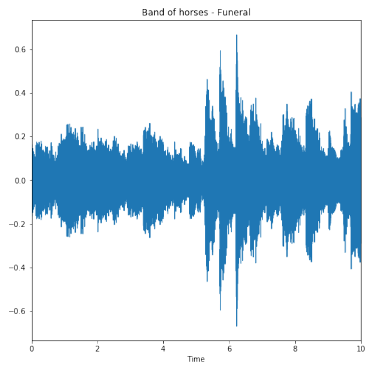
Figure 2 shows the corresponding signals from Figure 1 in the frequency-plane. In Figure 2 the x-axis is time while the y-axis contains the frequency.

The color corresponds to the energy or amplitude for the frequency. As is visible most of the energy is distributed in the lower frequencies. By comparing the spectrogram to the waveform of the audio, Figure 1, it becomes clear that more information is present in the spectrogram.

In contrast to Figure 1, Figure 2 also contains clear information concerning the pitch, and the strength the notes that are played. The pitch is visualized by the frequency, y-axis, and the strength is represented by the colors, which are mapped to values on the right side of the Figure 2, a and b.

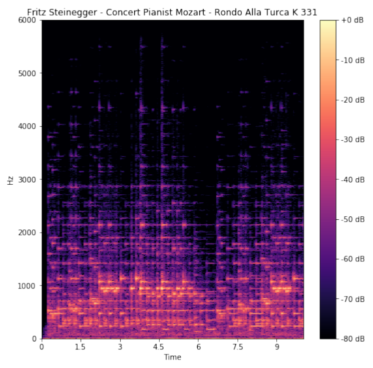


(a) Parts of the song, seconds 0:10. The song is a classical piece with clear tones of piano playing the melody with accompanying background.

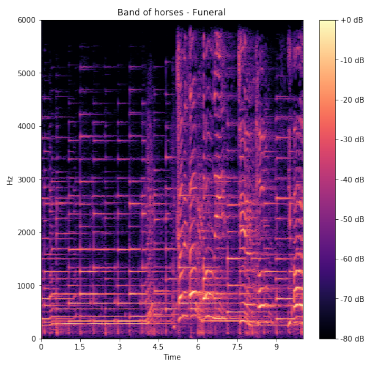


(b) Parts of the song, seconds 15:25. The song is labeled as Alternative & Punk. The first five seconds contains intro where a guitar plays the melody. Between seconds 5 and 6 vocals start as can be seen as the amplitude of the wave increases.

Figure 1: Audio waveforms of types of music.



(a)



(b)

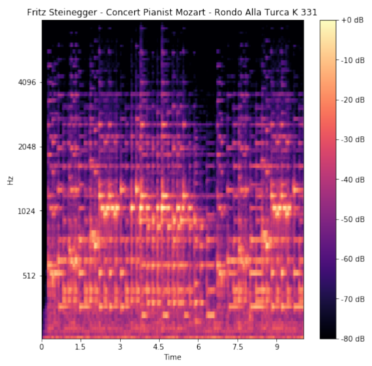
Figure 2: Linear-frequency power spectrum extracted from the audio in Figure 1.

2.2.2 Mel-spectrogram

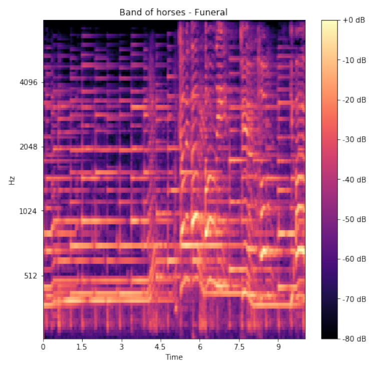
Human perception of sound is not linear, but logarithmic. As frequency increases humans it becomes harder to hear the difference between different tones. Different auditory scales exists in order to model this with examples being the Mel-Scale, Bark-Scale, ERB-Scale and the Cent-Scale. All of these scales have their own mathematical formula, but share the same concept, they try to model human perception in different ways. The most commonly used is the Mel-Scale and this is probably because it is used to calculate MFCC features.

The output of transforming the spectrogram with the Mel-Scale is called a Mel-spectrogram. The formula for transforming frequencies to Mel-Scale is, $f_{mel} = 2595log_{10}(\frac{f_{Hz}}{700} + 1)$.

After applying the Mel-Scale to the spectrogram frequencies are grouped, and often around 128 groups are used. The output is called the Mel-spectrogram and is visualized in Figure 3. The smallest unit of time on the temporal axis is called a frame, and that is also why these kinds of features are called frame-level features.



(a)



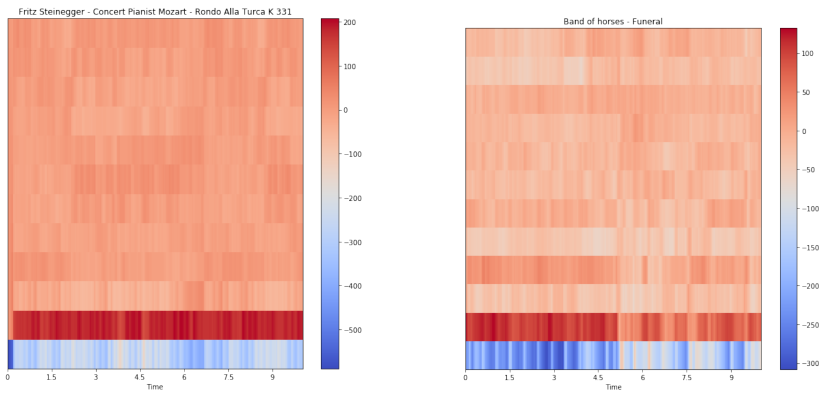
(b)

Figure 3: Mel-spectrogram with logarithmic amplitude extracted from the audio in Figure 1.

2.2.3 Mel Frequency Cepstral Coeffiecents

The Mel-spectrogram contains much data. If a song would have 10 frames per second, 128 bins and a duration of 3 minutes, there would be $10 * 128 * 3 * 60 = 230400$ floats of data. The frames are heavily correlated, leading to redundant information, and not a very compressed representation. To cope with the large amount of data dimensionality reduction is used, and for the creation of MFCC features the discrete cosine transform(DCT) is applied to the Mel-spectrogram

The reason for applying DCT is that it approximates the principal component analysis(PCA) for music signals which optimally decorrelates the data[15]. The DCT is computationally more efficient than the PCA transform.



(a) (b)

Figure 4: MFCC features extracted from the audio in Figure 1.

2.3 Higher level features

While frame level features are in some cases used as they are, an issue is the sheer amount of frames. Also, each frame only represents a small unit of time and, therefore, aspects that can only be perceived with a larger time scale are either missed or requires several frames to be represented.

Features that are created from several frame level features are here defined as higher level features since they represent a higher abstraction of music.

Evaluation of features is often a part of Audio Music Similarity Retrieval(AMSR) which is further disscused in Section 2.4, where features are used to find songs that are similar to one another. For accurate recommendations based on content, the feature space needs to represent the human perception of sound as closely as possible.

In this subsection, we present two types of high level features. The first presented is handcrafted features, which are created by researchers trying to represent some aspect of music, and the second is features extracted with deep learning techniques.

2.3.1 Handcrafted features

Handcrafted features have been specifically created to represent some aspect of music.

Examples of handcrafted features could be correlations to find reoccurring rhythm patterns, or variance to capture changes in the song. In Figure 5 onsets, which are the beginnings of musical notes, extracted from a spectrogram are visualized. From the onsets and their strength, beat per minute can be extracted. Many more handcrafted features exist and depending on the objective some might be more interesting than others. For more in-depth information, we recommend reading Seyerlehner's dissertation paper, [18]. Music is packed with information in the frequency domain and the temporal domain. Some features might be possible to find with a short time scale. Other aspects such as structure could require a much larger time scale to represent accurately.

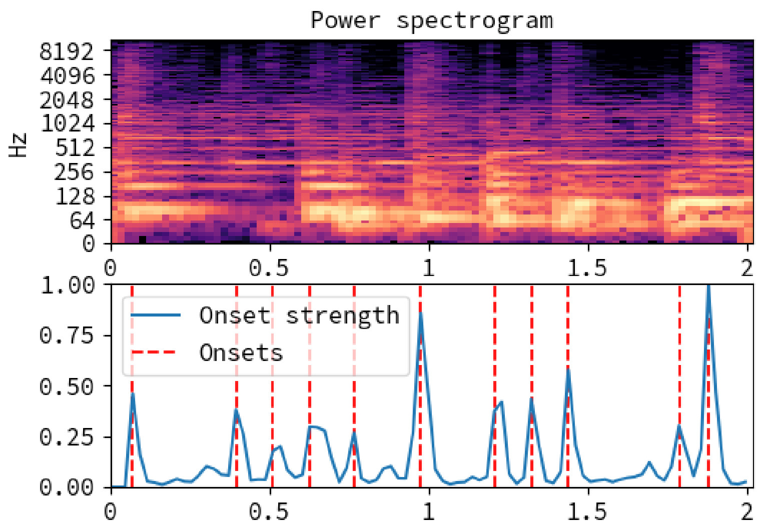


Figure 5: In the upper image the spectrogram of an audio signal is shown. Below are extracted onsets together with the strength of the onset.[11]

2.3.2 Deep Learning for music features

Deep learning methods have improved a lot over the last few years, and in many fields, they are state of the art. In many image classification problems, models based on convolutional neural networks(CNN's) are state of the art. In natural language understanding, models based on recurrent neural networks(RNN), e.g long short-term memory(LSTM) or gated reccurent unit(GRU) cells, are state of the art models for many tasks due to their ability to represent history. Since music can be represented as 2D images, techniques from image recognition field are of interest. To keep in mind though is that images are in a spatial space, while in music is in the time-frequency space when using the spectrogram as features. Given the progress in image recognition and natural language understanding, it makes sense that the techniques from these fields could also be

[11]https://librosa.github.io/librosa/_images/librosa-onset-onset_detect-1.png
[11]http://lantana.tenet.res.in/music/mridangam

applied to music.

Already in 2011, neural network(NN) and deep neural network(DNN) approaches was becoming more and more popular for MIR. Since the content of music is complex to work with even though patterns exist, it makes it an attractive target for deep learning. While the handcrafted features mentioned tries to capture different aspects of music, the signal and the information content is so complicated that it can be hard to represent accurately. As the DNN architecture is hierarchical, the use of deep learning techniques seems to be the perfect fit for many MIR applications. While higher layers would be able to find more local patterns, deeper layers should be able to find higher level patterns.

It is clear that DNN is a good fit for this type of data. For feature extraction, we look at DNN from two perspectives:

Unsupervised learning: The idea is to compress the data and then recreate it. Training is conducted by comparing the input with the output, which ideally should be the same. The reduced dimensions, embedding, from within the network, can be extracted and can be used as features.

Supervised learning: Using features such as MFCC, or the Mel-spectrogram, a deep learning model can be trained to classify objectives such as genre or tag classification. Features from different layers in the DNN can be extracted and used as features.

For both of these cases, finding labels is not an issue. In the case of genre classification, music is often already classified, and several datasets exist. In the other case of unsupervised learning, we are merely trying to recreate the original features and therefore, do not need a label.

For the supervised learning approach lots of literature exists[4-5][14-16][25]. The features used often differ and also the datasets. The most common feature used, and especially lately, is the Mel-spectrogram, but features such as the spectrogram occur as well.

These models could be considered solving the problem of genre classification and often scores above 90% accuracy in genre classification. These models also score well on tag classification.

There is one paper, that even if it is not only a content-based approach that is worth mentioning. In [20], latent factors(embeddings) of music are created by collaborative filtering. Using audio, these latent factors are predicted. In Figure 6 an embedding space is visualized by predicting latent factors from audio and then using t-SNE for dimensionality reduction. Some genres are possible to discern, and this is only with 2 dimensions. With 3 or 4 dimensions it should be possible to find better clusters for the genres. The data used is the MSD dataset together with an extension containing user data in the form of how many times they listened to specific songs. The authors of [20] were able to attain 29 second clips of 99% of the songs in the dataset from a seperate website.

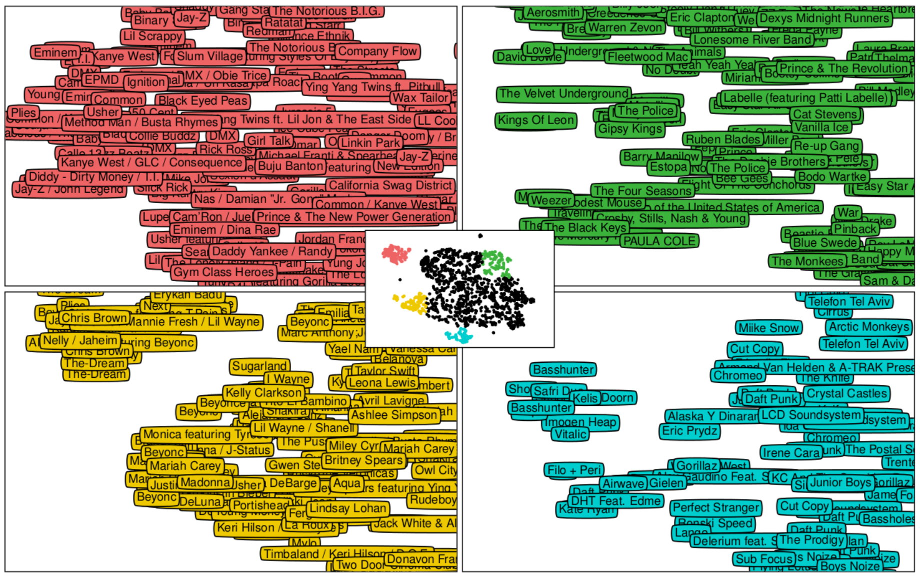


Figure 6: Image of embedding space from the paper [20]. In the image red is hiphop, rock is green, pop is yellow and electronic music is blue. Other genres are not possible to discern.

The unsupervised learning approach is to our finding less popular, and the only literature found on the subject is Schlüter's work from 2011[15].

Something worth mentioning is that while handcrafted features often are constructed with a musical aspect in mind, the unsupervised and supervised approaches described here optimizes for a classification objective. It is unclear how well the optimization and features constructed at different layers in DNN represents human perception. It could well be that there is more to music than what humans perceive, as discussed in [13].

2.4 Audio Music Similarity Retrieval

The information contained in the audio content needs to be represented adequately to make suitable recommendations. In the previous Section 2.3 we described two types of high level features, handcrafted features, and features extracted with deep learning. These features represent a more compressed representation of music and music from a higher abstraction than frame level features. Therefore, high level features are commonly used for retrieval purposes such as when retrieving similar songs.

After features are extracted, another problem exists. It is far from trivial to evaluate how good they describe the music, at least from human perception. In the Audio Music Similarity competition, a part of MIREX[12], test-users rank

[12]http://www.music-ir.org/mirex/wiki/MIREX_HOME

songs in order of similarity to a query song. The ordering of the test users is compared to the ranking of contesting algorithms. Unfortunaly, none of this data is publicly available, and gathering this kind of data is time-consuming.

The similarity in music is also complicated because people have a different perception of similarity. It could be cultural, depend on what type of music being consumed or other reasons[7]. To add to the problems in the field, music is often not publicly available. Since most music is not publicly available, the number of datasets available is limited. Some researchers also evaluate their models with music they are not allowed to redistribute.

Some publicly available datasets exist but are often limited from a few hundred to a few thousand songs together with the genre. The kinds of music in the datasets also varies. Examples of datasets are *GTZAN*, *ISMIR 2004*, *1517-Artists* and *Ballroom*. *MagnaTagATune*[10] contains tags instead of genre labels gathered by real users. Examples of tags can be guitar, rock, slow or fast. The *MillionSongDataset*(MSD)[1] is another well known dataset containing around one million songs. The audio is not available for the dataset, but features such as beat and features similar to MFCC exist for each song. For the MSD dataset, there also exist other data that extends the dataset for a subset of the songs, or all of them. An example of an extension is the *Taste Profile Subset* containing users and counts of the songs they listened to. Another contains tags from *Last.fm*.

Because of the mentioned problems, evaluation is often done against the genre. If the objective is to find similar songs, the genre of query song is compared to the genre of the songs with the highest similarity. Sometimes genre prediction accuracy is also used, by training a model to predict the genre of songs. If the accuracy is high, features are believed to be suitable for similarity tasks. The motivation for using genre is that it resembles a coarse human perception of similarity.

2.4.1 Playlist Generation

In AMSR a query song is required to make recommendations. Since a query song is required, either the user has to provide it, or a song suitable for querying needs to be found.

Another important subfield, for this thesis, is playlist generation(PG). In general playlist generation is about recommending a list of songs, the consumer enjoys. The focus could be everything from finding good candidates, to focusing on transitions between the songs in the playlist. PG can be based on information gathered from users, from the content or a mix. In this thesis, we are not interested in the sequencing but more the recommendation part and therefore, we focus on finding the candidate songs. In recent literature, the problem is often solved by applying reinforcement learning.

In [11] an interesting algorithm is described. The algorithm retrieves both candidate songs and how the songs should be ordered. It is modeled as a reinforcement learning problem. For candidates songs to recommend, it learns what features of music the user prefers and adjusts weights to increase the probability of songs with similar features to be chosen as candidates. From the MSD dataset, 34 features are used and each of the features is digitized into 10 values. In feature space, each song becomes a binary sparse vector with 1's at the entries the song populates and 0's everywhere else. A weight matrix for each user exists of the same size, which gives an indication of what features the user enjoys.

For sequencing, a reward function is maximized by tree search. Depending on user preferences, a subset of songs are selected. From the subset, songs are randomly chosen and sequenced, creating a trajectory. Several of these trajectories are gathered, and the selected trajectory, recommended to the user is the trajectory with the highest calculated reward.

In [3], the author uses Q-Learning, a type of reinforcement learning(RL), for recommending music based on the emotion of the song. Songs are categorized into four different classes. To tune the parameters of the model, they simulate users having three possible actions: skip, repeat, and rate. It is a rather simple problem compared to many others, but simulating users for parameter setting is exciting and potentially useful.

In [2], the authors have data from a music streaming application available. They create a graph connecting users to songs and songs to playlists. From the graph, they create embeddings. In a second paper, [19], the embeddings are utilized together with deep reinforcement learning for PG. The model they use is based on an attention-RNN language model where every action represents a song.

2.5 Conclusions

At the beginning of this section, we mentioned the aspects of music. Many exist, but the single most crucial aspect is timbre when comparing similar songs. To represent music by features, we described how the frame level MFCC features are created by converting the audio to the spectrogram, and then to Mel-spectrogram, and lastly transformed with DCT. How high level features, handcrafted and extracted by deep learning, can be created was after that briefly presented. Lastly, the subfields AMSR and PG were described, and relevant literature presented.

Information in music is complex, and when using traditional techniques such as handcrafted features, domain knowledge is essential. If the objective would be to make a recommendation on a specific type of music, that is distinguished by e,g the rhythm, then perhaps a handcrafted feature that is constructed to represent rhythm would be the right choice. In this thesis, we want to recommend music more generally, and therefore, several handcrafted features could be required. Also, many of the handcrafted features would need to be implemented.

Extracting features with deep learning, on the other hand, is relatively simple. The network creates features of their own, through optimization. The question then is if these features can resemble the human perception of sound[13]. By looking at Figure 6, it is evident that even in 2D, one can discern some genres. To be able to discern genre from each other with just two dimensions is powerful. In higher dimensions, it should be even easier to discern them. If a similar embedding space could be extracted from a deep learning genre or tag classifier that could potentially be appropriate features. It is worth an evaluation.

From PG, a few ideas are interesting. The idea of simulating users as in [3] is interesting if you cannot interact with users or have a dataset containing interactions. Finding what features of music a user prefers, as in [11], would be easy to simulate and is an exciting way of looking at the problem. If features preferred are found, then it should be possible to recommend songs that have similar features, the retrieval problem. Although recommendations can never become better than the features describing the music if the recommendations are made on content.

The last few years deep reinforcement learning(DRL) have gained widespread attention because its performance in a wide set of RL problems such as Atari games, but recommendation problems have also been explored. In [19] a DRL approach is used, but unfortunatly they are using music features extracted from a user-song-playlist graph. Although, they mention that also using content could be beneficial. In the paper, [19] every song is an action. If the song database is large, this clearly becomes an issue. With the approach from [11] songs with the same combination of features can be mapped to the same bin representing the combination of features. The bin representing a combination of features could then be sampled then. If no song exist in the bin, a nearest neighbour approach could be taken, and another bin could be sampled from.

Ideas from the three papers, [3], [11] and [19] could be combined with an approach without user data, by simulating users with individual music preferences. To optimize the recommendation and find the combination of features the user prefers, the DRL provides an exciting approach.

3 Problem definition and overview of solution

As described in Section 1 the problem of recommending music can be divided into two problems. The first problem is finding a song that the user enjoys, and the second problem is to retrieve similar songs.

In this thesis, the first problem is narrowed down to finding a preferred set of features the user enjoys in music, which will be done by user-interactions. The system stores no previous interactions and has to explore combinations of features, to find the users preferred settings. Once the preferred setting is found, recommendations from that setting can be made. An assumption is made that features represent music perfectly. Users are simulated and their feedback follows a hardcoded set of rules. Recommendations are optimized with a DRL approach. In Section 4 relevant theory of RL, and DRL is presented and then the environment that agents interact with. Finally, models are trained and evaluated that solves for the first problem.

The second problem is to retrieve similar songs. By doing so, the feature space has to represent the human perception of similarity accurately. Therefore the second problem equals to extracting and evaluating these features. Features are extracted from a deep learning music-tagger.

In Section 5 features for a dataset is extracted from a different layer of the deep learning model. Features are compared qualitatively.

Lastly, the two parts are put together, forming the music recommendation system, that is quantitatively and qualitatively evaluated in Section 6.

4 Reinforcement learning

In this section we look at first part of the problem, which is to find prefered setting of features the user enjoys. First in this section reinforcement learning, and deep reinforcement learning theory is presented. Secondly, the environment used for training model is explained. Implementation details are then presented, and finally experiments are conducted and presented.

4.1 Theory

Reinforcement learning can be explained as an add-on to supervised learning. In supervised learning, and in the case of classification, the input is transformed into a probability for each of the different classes. When training, labels exist and it is possible to say directly what is right and what is wrong. In reinforcement learning classes are defined as actions instead. What action is good or bad is often unknown at the given time. Instead, it will be found out in the future if the action taken led to what is called a reward, r. For a self-driving car staying on the road for a longer time would typically produce a high reward, while crashing into a ditch would result in a low reward(hopefully). It can take several actions before success or failure is known.

Every time an action is taken, an observation or state, s, is returned. In many cases, observations and state differ, but in the context of this thesis, they become the same. The observation, or state, is used as input for the action taken at the next step. A model thus chooses actions depending on the observations.

There are different types of reinforcement learning algorithms such as policy gradients, value-based, and actor-critic. In literature, policy gradients and notably actor-critic algorithm have shown promising results and is, therefore, the focus.

The type of algorithms used in this thesis is actor-critic, but actor-critic are often grouped into a policy gradient type of algorithm. Since policy gradient algorithms give an excellent introduction to actor-critic algorithms, policy gradient is first explained in Section 4.1.1 and in Section 4.1.2 we extend to actor-critic algorithms and the main algorithm used in this thesis.

4.1.1 Policy gradients

The objective function for policy gradients can be defined as,

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[\sum_t (r(s_t, a_t)]$$

where $r(s_t, a_t)$ is the reward for taking action a_t in state s_t at time t, and τ is a trajectory sampled by running the model π_θ with parameters θ. The objective function, $\theta^* = argmax_\theta E_{\tau \sim \pi_\theta(\tau)}[\sum_t r(s_t, a_t)]$, is to be maximized on θ.

The reward given is dependent on the state and the action together. The model samples actions depending on the state, $\pi_\theta(a_t|s_t)$. The objective function can be approximated $J(\theta)$ by running the agent in the environment,

$$J(\theta) \approx \sum_i \sum_t r(s_{i,t}, a_{i,t})$$

where i stands for trajectory i.

In words, we want to maximize the expected total reward for trajectories by changing the θ parameter. In simple policy gradient, this equals to changing the model and as can be understood by the name policy gradient, this is done by taking steps in the gradient direction. The model can be called many things and often it is referred to as actor, agent, policy model or simply model but in the terms of policy gradient, these are all the same. A simple policy gradient algorithm can be defined as the following:

1. Run the policy to generate a trajectory.
2. Estimate the rewards/returns.
3. Compute the gradient.
4. Alter the policy by taking a step in the gradient direction.
5. Loop over step 1-4.

A known issue with policy gradient methods is the high variance of the gradient. The variance makes convergence hard and may require many trajectories. Different ways to cope with this have been suggested, and in [16] the authors list a few versions of the gradient which all takes the form,

$$\nabla_\theta J(\theta) = E[\sum_t \Psi_t \nabla_\theta log \pi_\theta(a_t|s_t)] \qquad (1)$$

where Ψ could be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: The total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r'_t$: Reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: Baselined version of 2.

In practice, the baseline $b(s_t)$ is often implemented as just being the average of rewards. The main difference between 1 and 2, is that in 2, rewards in the future, following t, is summed up(casualty). This is sometimes called, "reward-to-go" or Q-function, which is described further in Subsection 4.1.2. Rewards in the past should not affect the rewards in the future, and by summing over fewer, the variance is reduced.

Two other ways of reducing the variance are by using discount-factors and n-step returns:

1. $\sum_{t'=t}^{\infty} \gamma^{t'-t} r'_t$: Discount factors. $0 < \gamma \leq 1$. Having rewards earlier, rather than later is better.

2. $\sum_{t'=t}^{t+n} r'_t$: N-step returns. We only look n-steps into the future.

Discount factors and n-step returns can be combined. The idea behind these two techniques is that further ahead, more actions will have been taken. In some cases, different actions are taken from the same state, and this causes trajectories starting from the same state to diverge. With more actions taken, trajectories will likely diverge more and thus increasing the variance of the returned rewards. By using techniques such as discount factors and n-step returns, it is possible to control the variance to some extent. Decreasing the variance can be useful if the variance is too high for the model to learn efficiently. Variance is often a problem in reinforcement learning.

4.1.2 Actor-critic

Policy gradients are closely related to actor-critic methods, and in some literature, they are listed under the same type of reinforcement learning.

The main difference is that instead of just using a model for the policy, a value function is learned to approximate future rewards. In actor-critic two models exists, actor and critic. The model taking actions in policy-gradient is often referred to as the actor, and sometimes also the agent. The value function that approximates future rewards is often referred to as the critic.

In Figure 7 more notations can be found. Depending on the literature, notations differ and here we follow the notations from the Berkley deep reinforcement learning course of 2017[13] closely.

In Subsection 4.1.1, equation 1, a few options for replacing Ψ were listed. Another option of replacing Ψ us by the advantage function $A(s_t, a_t)$.

From Figure 7, the only thing needed to approximate the advantage function is the value function. In actor-critic methods, the critic approximates the value function. Using the advantage function, together with the critic, greatly reduces the variance of the gradient. Comparisons can be made with the baseline $b(s_t)$ from policy gradient.

[13]`http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/`

Notation	***Definition***
T	The total amount of steps in the trajectory. For a trajectory with inifinite amount of steps $T = \infty$.
$Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_t', a_t')\vert s_t, a_t]$	Q-function. The total reward from taking a_t in s_t.
$V^\pi(s_t) = E_{a_t \sim \pi_\theta(a_t\vert s_t)}[Q^\pi(s_t, a_t)]$	Value function. Total reward from s_t.
$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \approx r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t)$	Advantage function. How much better is action a_t.

Figure 7: Notations and definitions for actor-critic.

The pseudocode for a simple actor-critic algorithm can be outlined as follows:

1. Sample a trajectory by running the actor.
2. Fit the value function to the sampled reward sums.
3. Evaluate the advantage function.
4. Compute the gradient.
5. Take a step in the gradient direction.
6. Loop over 1-5.

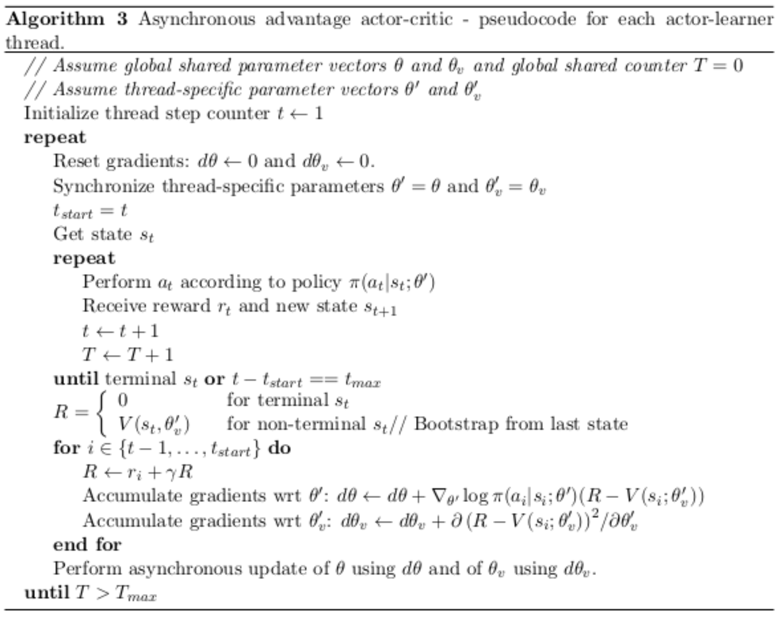
4.1.3 A2C

The paper [12] outlines, among others, an asynchronous actor-critic algorithm, called A3C. In A3C processes are spawned and run in parallel. Each process generates trajectories and in a hogwild fashion[14], update the model. The parameters for the model is thus shared across the processes.

The difference between A2C and A3C is that A2C is synchronous. In A2C many processes also run in parallel, but trajectories from the process are synchronized and put in a batch together on a single process. On this single process, the model updates, allowing for fast batch updates using GPU.

OpenAI has an implementation for A2C implemented in tensorflow, and Ilya Kostrikov [9] one implemented in pytorch.

We have found no good pseudocode for the A2C algorithm, and therefore the pseudocode for A3C is presented in Figure 8, although the algorithm used in the thesis is A2C.



Algorithm 3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```
// Assume global shared parameter vectors $\theta$ and $\theta_v$ and global shared counter $T = 0$
// Assume thread-specific parameter vectors $\theta'$ and $\theta'_v$
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state $s_t$
    repeat
        Perform $a_t$ according to policy $\pi(a_t|s_t;\theta')$
        Receive reward $r_t$ and new state $s_{t+1}$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal $s_t$ or $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ do
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i;\theta'_v))^2 / \partial\theta'_v$
    end for
    Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
until $T > T_{max}$
```

Figure 8: Pseudocode for the A3C algorithm taken from the paper[12].

4.2 Environments

An environment is what an agent is interacting with. When an environment is reset, it returns a starting observation. The observation is used as input to the agent, which returns an action used as input to the environment. The environment takes a step with the action and returns a new observation together with a reward. Keep in mind that state and observations are the same things in this thesis.

For our environment, we have set a maximum number of steps that can be taken, 21. The number was greedily chosen, with the motivation that it should be possible to find the preferred setting within 21 steps. The amount of steps needed, does, of course, depend on the amonut of combination of features, or as we could call it, the number of positions available, but should be enough in the setting presented. After 21 steps have been taken, the environment is reset again, and the agents start from the beginning again. When the environment is reset a new preferred setting is chosen. Examples of what an environment can look like can be found on OpenAI website[14]. For simplicity, we try to use the same vocabulary as OpenAI.

The observation space contains a reward function, user and feedback function, and observation representation.

[14]`https://gym.openai.com/envs/`

4.2.1 Action space

In this thesis, the action space is the positions, or settings, the agent can take and is discrete. The action space is simple, but with many features or dimensions, which is the same thing, the size of the action space becomes enormous.

As the number of dimensions and with the number of levels per dimension. The size of the action space becomes n_levels^d with d being the number of dimensions and n_levels the number of levels per dimension.

4.2.2 Observation space

With inspiration from [11] we consider the goal to find a target. A target is the preferred feature setting or combination that a user prefers. This idea is based on the assumption that music can be represented by a combination of features values. Songs with similar features should sound similar. For every feature, the user prefers a specific level, valued 0 to 4.

The goal of the agent is to explore the positions and find the target as quickly as possible. Depending on the feedback function, the target can be one position or several positions that are close to each other.

Following is defined how users are simulated, how feedback is produced, and how the rewards are computed.

User and feedback function

As mentioned above, the goal is to find the target, defined by a combination of feature levels. Each simulated user has their own uniformly randomized target, and an example of a target is visualized in Figure 9. When exploring the actor selects an action corresponding to a position, which is a combination of feature levels just as the target. The simulated user evaluates the combination of feature levels.

The simulated user returns two feedback values. One if the current position has a lower distance, manhattan(absolute) distance, to the target than the previous position. This feedback is referred to as closer feedback. The second value is the rating feedback that can take 5 values, $\{0, 1, 2, 3, 4\}$. The higher the rating, the closer the position is to the target, and if rating 4 is returned the target has been found. In this thesis, every simulated user is deterministic, meaning they will always follow a set of rules for producing feedback that is non-random.

For the rating, we decided to give a percentage of all settings to be of a specific rating. For each target, distances to all positions become calculated. A rating is given accordingly to a distance interval that is calculated with percentiles. Two settings for the percentiles was experimented with named TopTarget and WiderTopTarget. For TopTarget the percentiles used was $[0, 0.01, 15, 30, 50, 100]$ and for WiderToptarget, $[0, 5, 15, 30, 50, 100]$.

If the distance from a position to the target falls in between the distances corresponding to the percentiles of indices 0 and 1, then the rating produced is 4. If the distance falls in between the distances corresponding to 1 and 2, a rating of 3 is produced and so on. The difference between the two settings is minor. In TopTarget only the target will have rating 4, and for WiderTopTarget 5% of the positions will have rating 4.

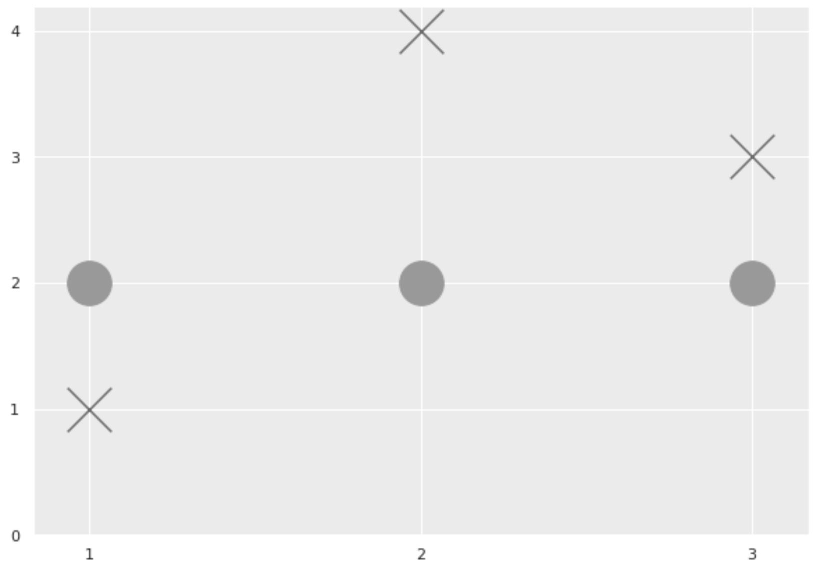


Figure 9: The x-axis contains features, and the y-axis contains feature levels. Each cross stands for the feature level the simulated user find ideal, for the specific feature. Together the crosses become the target. The circles are the start position. Every time the observation restarts the start position becomes the position.

Reward functions

Two different reward functions were implemented and tested. Both the reward functions compute the reward based on the rating provided by the user. The reward functions were named Bullseye and Rating. For Bullseye rewards, only a rating of 4 produces a reward which was arbitrarily set to 10. For Rating rewards, the rating of the user is squared and returned as the reward. The reasons for squaring the reward was only to increase the intervals between the ratings, to make the effort of exploring and finding a higher rating more rewarding.

At the beginning of training, finding the target could be hard, resulting in the agent very sparsely receive the non-zero reward in the case of Bullseye. In DRL it is essential that the agent sees a mix of good and bad trajectories. The implication could be that the agent would require more training for converging and finding the patterns to find the rewards. With the Rating reward, non-zero rewards are given much more often, and being closer to the target is rewarded.

Observation representation

The observations become the input to the model. Therefore, it needs to contain enough information in a format the model can reason about. Two different representation of observations were tested, GridObs and BoardObs.

GridObs contains the current position together with the feedback provided by the user. The size of this representing becomes the number of features plus the number feedback values, $d + 2$ with 2 being the number of feedback values. Unfortunately, this representation does not contain any information about the history which is necessary to reason about exploration.

BoardObs contains more information than GridObs. The observations are an array of the same size as the action space, n_levels^d. Each entry in the array resembles a position and is initiated with -1 for all entries. After the agent visits a position, the entry corresponding to that position is replaced with the rating from the user feedback. While this representation contains information about the history, it does so at the expense of not being a compressed representation, which can be an issue because of the exponential increase of the size with the number of features.

4.3 Implementation

For implementation Ilya Kostrikov's repository[9] was forked.

The repository contains a well implemented A2C algorithm and other DRL algorithms such as ACKTR[21] and PPO[17]. In the repository OpenAI's base class for environments were extended by the environments described in 4.2. The repository has been evaluated in many environments from OpenAI, Mujoco, and others. Reproducing results is not trivial in DRL, and there is often lots of variance in success between training runs[8]. In order to get deterministic behavior, there are many seeds to define. For example, when the model is initiated in Pytorch weights are by default, randomized with some distribution. There are also other non-deterministic parts of the algorithm, such as the sampling of actions, which exist for exploration purposes. In the experiments, the targets were also randomized. If the training is not robust, a poorly sampled trajectory can cause a decrease in performance.

There is also the matter of where to store the data, CPU or GPU, how and when to transfer data between the devices. The storing is done very efficiently in the repository of Ilya Kostrikov by keeping the model at all time on the GPU. Environments are run in parallel on the CPU and the returned values are transferred to GPU for updating the model. The repository uses many functionalities such as the multiprocessing from the OpenAI baseline repository[5].

Some changes were made though, but no changes altered the DRL algorithms. Changes implemented included saving, logging, arguments, and our environments. A script for evaluation of models was also implemented.

4.3.1 Model

While observations from BoardObs contains information of history, GridObs does not. In some way, historical positions need to be represented for the agent to reason efficiently, at least for the actor. Because the reward function computes the rewards directly from the rating provided by the user, the critic should be able to make rather good predictions without any historical information.

One way to represent history could be by concatenating previous observations with the current observation into a 1D tensor and feed it into a neural network. A second option would be to sequentially input observations into an RNN. In the repository, GRU cells are used. However, RNN's such as GRU or LSTM are known to be hard to train. Without altering the code, RNN was the only possible option of the two. The output of the RNN is fed into a three-layered neural network(linear layers). The output of the neural network are logits with the number of actions as the size, and softmax is applied to the logits to transform them to probabilities. In training, actions are sampled from the probabilities outputted by the softmax function, in comparisons to when evaluating when the action of the highest probability is taken.

Hyperparameters for the model

In DL, many hyperparameters are interacting. We use the default hyperparameters from the repository in most cases. Some experiments with the hyperparameters was done, but as they hardly had any effect on results, we do not report them.

For every sequence that is fed into the GRU, a hidden state is outputted. For the next input of the sequence, the hidden state from the previous sequence is also used as input to the GRU cell. Sometimes the hidden state is also called the context vector and contains a representation of the history. In many cases, the hidden state from the last is fed into linear layers for classification purposes. Hyperparameters for the GRU are hidden size and num_layers. The hidden size controls the dimensionality of the hidden state and thus the number of parameters. Num_layers is the amount of stacked layers as in a neural network or other machine learning techniques. The output from the first GRU layer goes into the second layer and so on. The output of the GRU, hidden state, goes into a neural network which outputs logits for each of the actions available.

For num_layers the default value of 3 was used and for the hidden size the default value of 64.

The RMSProp optimizer was used to optimize the model, with hyperparameters β and epsilon set to 0.99 and 10^{-5} respectively. The learning rate was by default $7 * 10^{-4}$.

4.3.2 Algorithmic hyperparameters

The algorithm also has some hyperparameters that affect the results.

The reward discount factor, γ, was set to a value of 0.99.

The loss function of A2C contains a value-loss coefficient that controls how fast the critic learns in comparison to the actor. The value-loss coefficient received a value of 0.5, which was the default value from the repository.

An entropy coefficient controls how much the entropy of actions should contribute to the policy loss function of the A2C algorithm. In the A3C paper, this is motivated as having a positive impact of not getting stuck in sub-optimal policies(local minima). The default value of the repository, 0.01, was used.

The last hyperparameter is the number of processes that run in parallel. It also controls the batch size of the updates made by the GPU. Usually, this is set to the number of cores on the CPU, and in our case, it was set to 16 since the machine used had 16 cores.

4.4 Experiments

There are many combinations to try and therefore have to be both greedy and smart in experimenting and choosing hyperparameters. There is a considerable risk of changing hyperparameters that are being bottlenecked by other hyperparameters. Another problem is the instability with DRL algorithms, as shown in [8]. To get reliable results, the same experiments would ideally run several times. Unfortunately, training can be time-consuming. Every combination or configuration was only run once here.

For deciding what parameters to use, or at least understand how they affect learning, we did experiments in 2D, with 5 levels per dimension. With 2D and 5 levels per dimensions, gave $5^2 = 25$ positions in total. Much experimentation was made by trial and error by changing hyperparameters, environment functions, and interpreting the results. In most cases, performance stayed the same, and we decide only to report those experiments which we deem interesting.

First experiments are made in 2D and the results discussed. Following experiments are made in 3D and 4D with using one of the promising configurations found from 2D. Since more dimensions means larger action space, more possible observations and more positions it is a harder problem.

4.4.1 Experiments in 2D

The reward function is an essential aspect the observation space. After all, the agent will be optimized to maximize the reward. If it is hard for the agent to find the target, it could lead to sub-optimal policies or high variance, which could cause problems when training. For example, if a rating of 3 is received, then 9 is given as the reward, in case of the rating reward. To increase the reward, the agent needs to explore, which could lead to receiving a lower reward while searching for a position with rating 4. This could lead to a suboptimal-policy where it is safer to stay in the same position. The risk might not be worth the potential reward.

The robustness of the training is another critical aspect. With more actions, the variance will increase. Large action spaces is a known issue with DRL.

With more features and levels, the information in music can be more accurately described.

In the following subsections, we alter some hyperparameters and test different observation representations, reward- and feedback functions. The plots visualized of the training shows the average total reward for the last 10 sampled trajectories. As will be seen in the plots, many configurations have a high variance in the average total reward. Ideally, the variance is low, and an example of how it ideally could look can be visualized by smoothing one of the plots heavily as shown in Figure 10. At the beginning of training, the variance is higher. This is likely due to the distribution used for sampling an action being more uniform, higher entropy. As training progresses the probability of sampling, the optimal action should increase while the probability for sampling bad actions decrease. In other words, there is more exploration in the beginning.

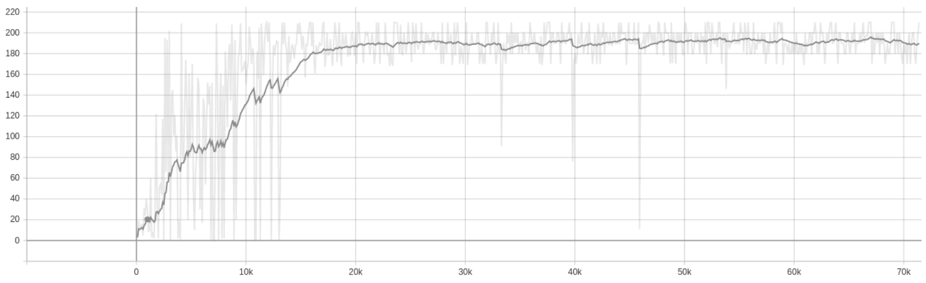


Figure 10: A smoothed version of a training plot. The stronger colored line is the smoothed version and the weaker the real training.

If not else mentioned, default hyperparameters are used.

TopTarget in comparison to WiderTopTarget

The first experimentations were with the reward function. It was believed that the reward function would have the most significant impact on performance.

In Figure 11 and 12 training is visualized from the two reward functions. The environment used was GridObs. GridObs with TopTarget has surprisingly less variance than WiderTopTarget. It was believed to be the other way around since finding one position should be harder than finding one of several.

As can be seen from both Figure 11 and 12 is that trajectories with low rewards tend to follow each other. Examples when this is visible is around iteration 190k in Figure 11, and around iteration 170k in Figure 12. We interpret this as if a bad trajectory is sampled, then there is an increased risk of the optimizer to take a large step in a bad direction.

One of the reasons the WiderTopTarget's performance is lower could be because of non-consistency of the rating levels. When using percentiles with a low amount of positions available, the number of positions with a rating will differ in the target position. Since the reward is only non-zero at rating equals 4, this could be one of the many reasons.

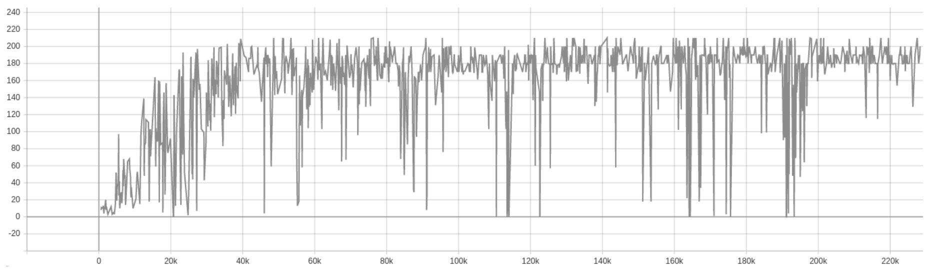


Figure 11: GridObs and TopTarget reward. After around 40k iterations the model is trained.

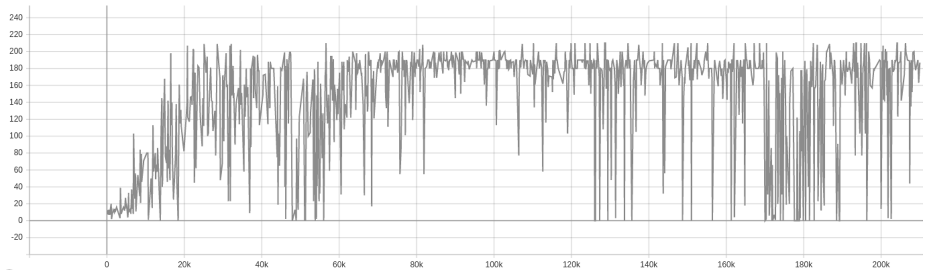


Figure 12: GridObs and WiderTopTarget reward. While the model learns, the variance is relatively high when comparing to Figure 11. After around 80k iterations the model is trained.

N-step returns

As explained in the theory, Section 4.1, the n in n-step returns controls after how many steps the model updates. To reduce variance we would like n to be small, but large enough for the actor to find the taret. If the target is not found within the n steps, then rewards with only 0 will be returned, in the case of bullseye reward, which could cause problem with learning. To start with, n was set to 5. While it should be possible for the actor to find the target within 5 steps it gives very little room for a bad action being sampled. Theory tells us that increasing n should increase the variance because trajectories divert the more in the future we look. Before we started altering this hyperparameter, we, therefore, tried many other things such as increasing the hidden size of the RNN, changed the number of layers without any success. Nothing seemed to stabilize the learning and decrease variance. In Figure 13 we increased n to 10, and in Figure 14, to 20. The results are clear. Increasing n reduces the variances.

A reason for the increased performance when increasing n could be that when n is low, the target is not often found, and as n increases the chance of finding the target increases, which could lead to a healthy mix of rewards.

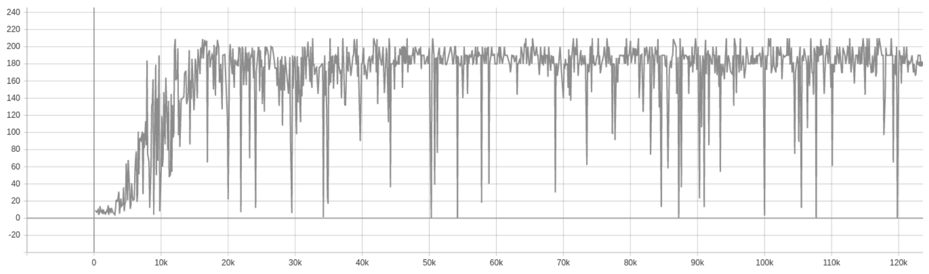


Figure 13: GridObs, WiderTopTarget and $n = 10$. After around 20k iterations the model is trained.

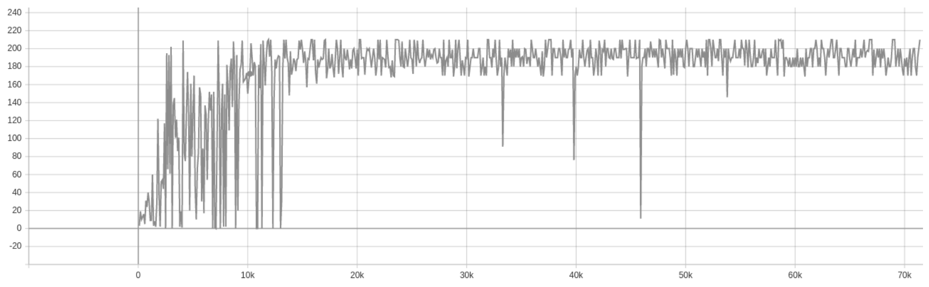


Figure 14: GridObs, WiderTopTarget and $n = 20$. After around 14k iterations the model is trained.

Learning rate

From experience, a too high learning rate can cause high variance in the learning. Therefore, the learning rate was lowered to $7 * 10^{-5}$ from the default value of $7 * 10^{-4}$. The results are plotted in Figure 15. Lowering the learning did not help as can be seen by the high variance in the average rewards, but with more iterations, it is possible that it would converge.

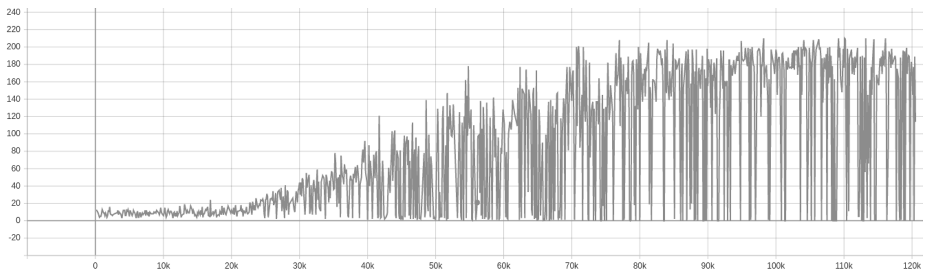


Figure 15: Learning rate $lr = 7 * 10^{-5}$ and $n = 10$. After 120k iterations the variance is still high.

BoardObs

If observations are not informative, no agent would be able to reason about the environment. The upper limit on what is possible to model is, in most cases information content. GridObs is dependent on the RNN cells to represent history. As previously mentioned, training can be hard. BoardObs does require RNN to represent history. This allows for doing experiments without the RNN and with only the linear layers. Since the ACKTR algorithm in the repository does not have an RNN implementation, it could be used with BoardObs.

The ACKTR algorithm is known to have less variance than the A2C algorithm. Less iteration is thus needed. Unfortunately, the tradeoff is computational cost. The tradeoff is, in many cases, beneficial when limited data is available.

In Figure 16 we show training for BoardObs with RNN and A2C. It does learn, and performs well. In Figure 17 we show training for BoardObs without RNN and with the ACKTR algorithm. Performance is good and ACKTR learns faster than the A2C algorithm.

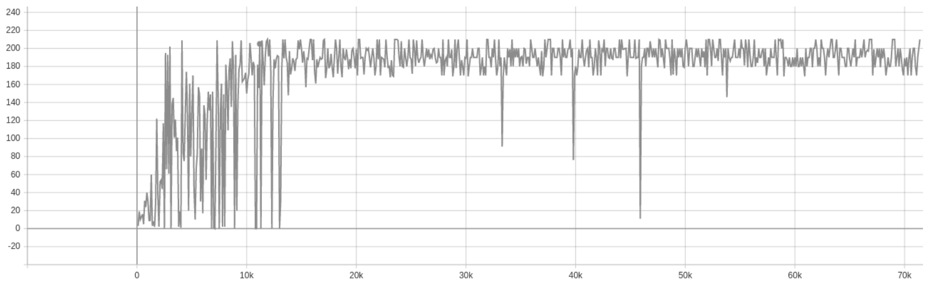


Figure 16: BoardObs, $n = 20$, A2C and RNN. After around 14k iterations the model is trained.

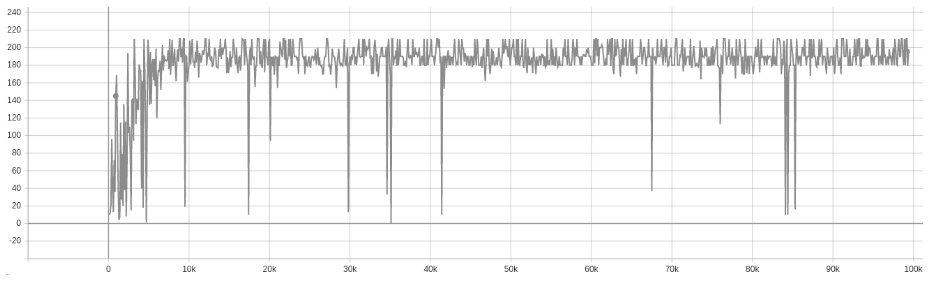


Figure 17: BoardObs, $n = 20$, ACKTR and without RNN. After around 10k iterations the model is trained. ACKTR is known to require less iterations than A2C for training.

Rating rewards

Before increasing n in n-step returns the rating reward was tested. Unfortunately, it was apparent that it got stuck in a sub-optimal policy. After increasing n, experiments with the rating rewards were conducted again.

Figure 18 show the training with using GridObs and rating reward. The average rewards cannot be compared to TopTarget or WiderTopTarget since the magnitude of rewards is different.

In Figure 21 the training configuration is the same as in Figure 18, except with BoardObs instead of GridObs. They perform similar in this case. BoardObs seems slightly more robust.

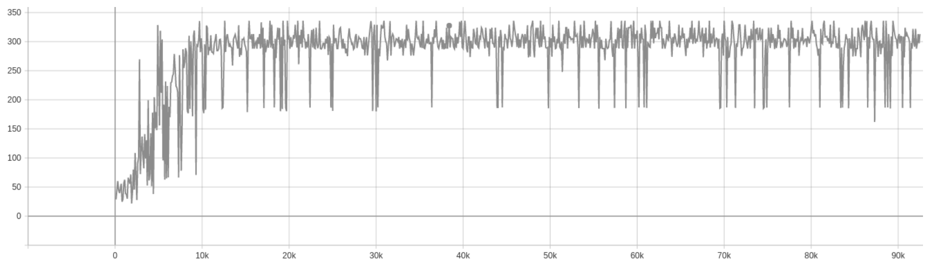


Figure 18: GridObs, $n = 20$, rating reward and WiderTopTarget. After around 11k iterations the model is trained. Some spikes with low average reward are present.

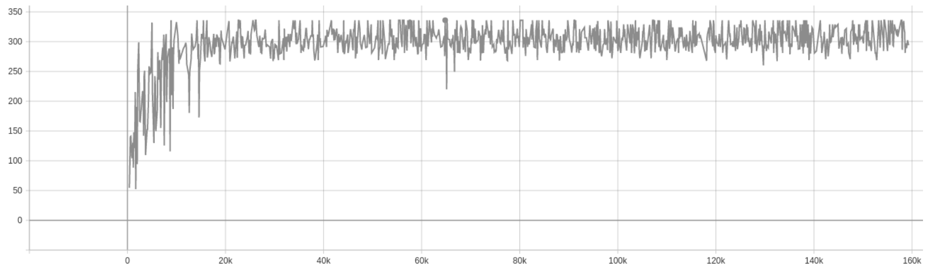


Figure 19: BoardObs, $n = 20$, rating reward and WiderTopTarget. After around 20k iterations the model is trained. This training does not any large spikes with low average reward.

The ACKTR algorithm was also tested and the results can be seen in Figure 20. The results show worse performance than the A2C variants using RNN's. For comparisons the A2C algorithm was run without using RNN, Figure 21. It seems as if the RNN has a stabilizing effect on the learning. An interesting thing to note is that in the previous experiments with the other reward functions, the difference in performance was not found.

One reason for explaining the results could be that the rating function could be more complex to optimize. More parameters could thus be beneficial. Another reason could be that the context vector built up by the RNN efficiently represents history.

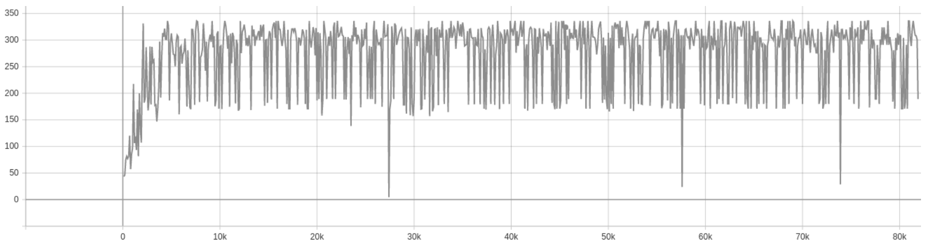


Figure 20: ACKTR, BoardObs, $n = 20$, rating reward, WiderTopTarget and no RNN. In this case the variance is high with few signs of decreasing after around 5k iterations.

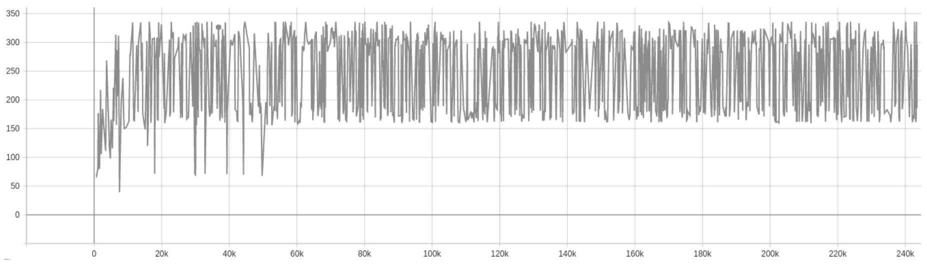


Figure 21: A2C, BoardObs, $n = 20$, rating reward, WiderTopTarget and no RNN. As in Figure 20 the variance is high with few signs of decreasing after 20k iterations.

Metrics

Visualizing the total average reward over some trajectories shows much of the information of interest, but not all. Also, in training, actions are sampled from a distribution. When the model is used, actions are deterministically selected by the action with the highest probability.

Metrics also allow for benchmarking. For benchmarking several models from the same training, 5k iterations apart were loaded. For each model, total rewards were calculated for every possible target position. From these numbers, the standard deviation and average of all total rewards were calculated. Ideally, the average reward should be high with a low standard deviation. As could be expected, a higher standard deviation leads in most cases to a lower average reward. The metrics for BullsEye rating can be found in Figure 22 and for Rating reward in Figure **??**. The results for the lowered learning rate is not shown.

Reward function	Observation representation	N-step returns	Recurrent	Algorithm	Std	Avg reward
TopTarget	GridObs	5	True	A2C	48	168
WiderTopTarget	GridObs	5	True	A2C	66	155
WiderTopTarget	GridObs	10	True	A2C	41	178
WiderTopTarget	GridObs	20	True	A2C	8	188
WiderTopTarget	BoardObs	20	True	A2C	11	192
WiderTopTarget	BoardObs	20	True	ACKTR	23	189

Figure 22: Metrics for TopTarget and WiderTopTarget reward functions

Reward function	Observation representation	N-step returns	Recurrent	Algorithm	Std	Avg reward
Rating	GridObs	20	True	A2C	33	300
Rating	BoardObs	20	True	A2C	19	305
Rating	BoardObs	20	False	A2C	70	253
Rating	BoardObs	20	False	ACKTR	59	280

Figure 23: Metrics for rating reward function

BoardObs have a slightly higher average reward, but the standard deviation is larger.

Discussion

Experiments in 2D were done in order to identify essential hyperparameters and to see how different observation representations, and reward functions impact.

The most critical hyperparameter that was found was n in n-step returns. It is possible that increasing it above 20 would be even more beneficial

Representing observations as either GridObs or BoardObs did not make much of a difference. GridObs show a lower standard deviation, while BoardObs show a slightly higher average total reward. As the dimension increase, the observation space of BoardObs exponentially increases, which is a huge flaw.

According to the experiments, the Bullseye reward functions make learning easier than the Rating reward function. As discussed in 4.2.2 the reverse was expected.

Amongst others, the experiments show how fragile the training is. Without good hyperparameter selection, learning is not always possible as was the case when n in n-step return was to low.

4.4.2 Experiments in higher dimensions

The goal of experimenting in 2D was to identify configurations of hyperparameters and functions from the environment that work together. Following the reasoning from Subsection 4.4.1, the Bullseye reward function is used. As the dimensions increase, finding a single target becomes harder for better generalization WiderTopTarget is used.

With more dimensions, music can be more accurately represented. Increasing 2D to 3D means increasing the possible positions from 25 to 125. In 4D, 625 positions are available.

In Figure 24 and 25 training with 3D respectively 4D is visualized. As can be seen, the variance increases slightly in 3D, and in 4D, the variance increase even more while still producing competitive results. The number of iterations needed for training also increases. Since the number of positions is larger than in 2D, it is natural that it takes a longer time to find the target, and therefore the reward drops slightly in comparisons to 2D.

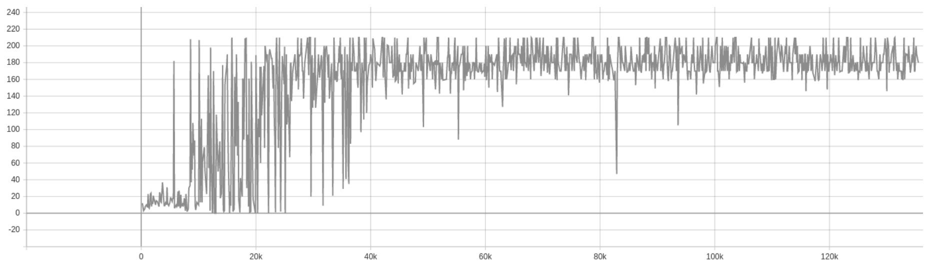


Figure 24: GridEnv, 3D, $n = 20$. After around 40k iterations the model is trained. The variance is slightly higher than the training for the same configuration in 2D, Figure 14. Also, the average reward is lower which is expected since more steps are required to find the target.

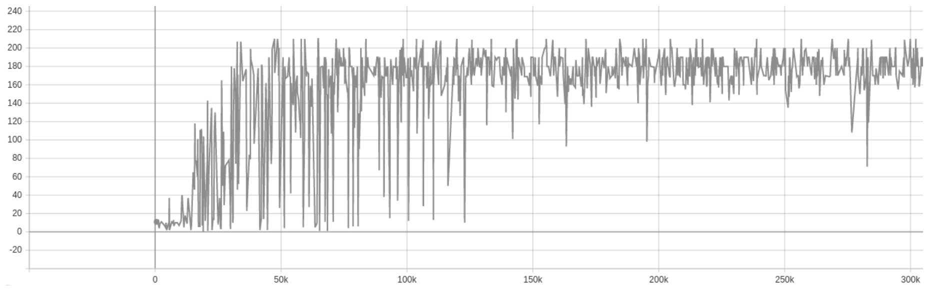


Figure 25: GridEnv, 4D, $n = 20$. After around 130k iterations the model is trained. In comparison to the same configuration trained in 3D, Figure 24, the variance is slightly higher. The average reward is also a little bit lower which is expected.

5 Feature extraction

In Section 4.4, a deep reinforcement learning systems for automatically recommending music was experimented with. For the system to recommend music, the positions taken by the model needs to be mapped against features extracted from songs in a database.

From the experiments, it is clear that a large action-space causes issues when recommending, training time, and variance increase. One way of coping with this issue could be to implement techniques similar to what is proposed in [6]. In the paper, a continuous action space is used, and with the nearest neighbor approach, the continuous action is transformed into a discrete action. Due to the limitations of this thesis, there is not enough time to implement these techniques, and thus 4 features have to be enough to represent the music.

Following the conclusions in Section 2.5, we identified that the best solution would be to extract features from a genre or tag classifier. Conveniently, pretrained genre and tag classifiers are available online. An example is the tag classifier used in the paper [4]. The pretrained classifier can be found at github[15] that takes Mel-spectrogram as input and outputs tag predictions in an interval. The tagger is built on CNN's, four convolutional layers with a dense layer in the end for predictions, that takes Mel-spectrograms as input. Features from all these different layers can be extracted to 50 dimensions. Layers closer to the input are considered extracting lower level features. Features that resemble the input more closely. As the data goes through the layers, it is transformed more and more. Layers, in the end, are then said to resemble features with a higher abstraction level.

The argument for using a tag classifier instead of a genre classifier was that several tags can exist for one song, but only one genre exists per song. Therefore, we argue that the feature representation from a tag classifier ought to be more detailed when comparing songs. The 50 tags predicted by the CNN are the following: rock, pop, alternative, indie, electronic, female vocalists, dance, 00s, alternative rock, jazz, beautiful, metal, chillout, male vocalists, classic rock, soul, indie rock, mellow, electronica, 80s, folk, 90s, chill, instrumental, punk, oldies, blues, hard rock, ambient, acoustic, experimental, female vocalist, guitar, hiphop, 70s, party, country, easy listening, sexy, catchy, funk, electro, heavy metal, progressive rock, 60s, rnb, indie pop, sad, house, happy.

5.1 Implementation

The dataset chosen was GTZAN, which contains the 10 genres blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, and rock. Each genre contains 100 songs, and each song contains 30 seconds of audio. The pretrained music-tagger was downloaded and tags were extracted. The documentation of the repository did not provide enough information on the versioning of the frameworks, e.g tensorflow, used. The default versions acquired by installing the packages did not work, and different older versions had to be tested until suitable versions were found.

Features from the second, third and last layer was extracted, each containing 50 features. Since the 50 features is to many to work with the DRL recommendation system, PCA was applied and the dimensionality reduced to 4D.

[15]`https://github.com/keunwoochoi/music-auto_tagging-keras/tree/master/compact_cnn`

5.2 Visualization of principal components in 2D and 3D

In Figure 26, 27, 28 plots of the 2 first PCs are visualized of the feature extracted from the last, third and second layer. There is no clear clustering of the genres, but some structure can be found. The embedding space is also different. This can be noticed by e.g considering the position of classical pieces in comparison to the other genres. In 3D it is possible to distinguish the different genres a bit better, but some genres are still clustered together.

Looking at the nearest neighbors of songs indicates how the music will be recommended. If the nearest neighbors to a song with features treated as the target are similar, then recommendations should make sense. If not, there is no way of making good recommendations with content. In many cases the nearest neighbor is another song from the same genre, but in other cases a song from another genre is. As can be seen in Figure 26, classical music inhibits the central parts. Our perception is that classical music is very different from all other music genres in the dataset. Therefore, it would have made sense that classical music cluster together far away from other genres. Perhaps closer to jazz or blues than other genres.

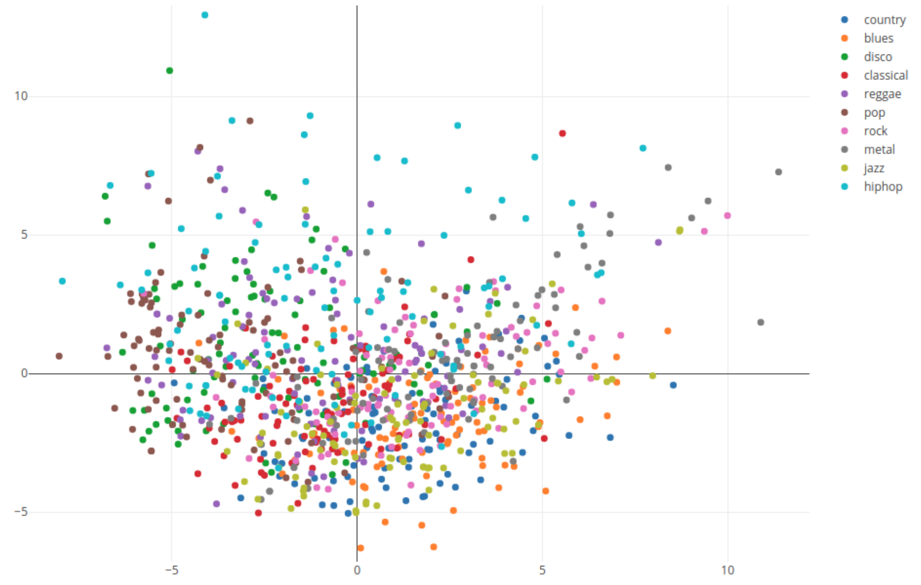


Figure 26: The 2 first PCs after PCA on the predicted tags. Features extracted from the last layer.

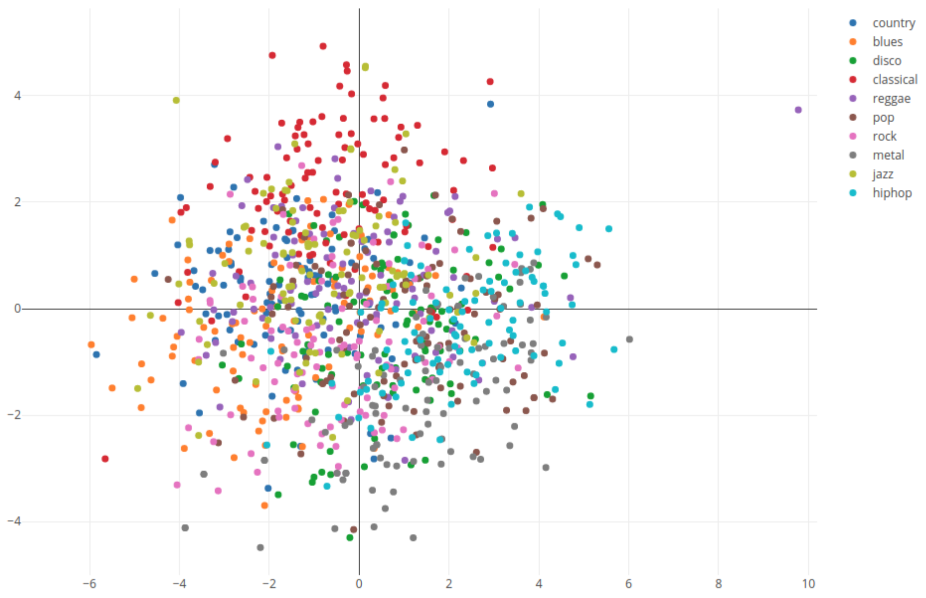


Figure 27: The 2 first PCs after PCA on the predicted tags. Features extracted from the third layer.

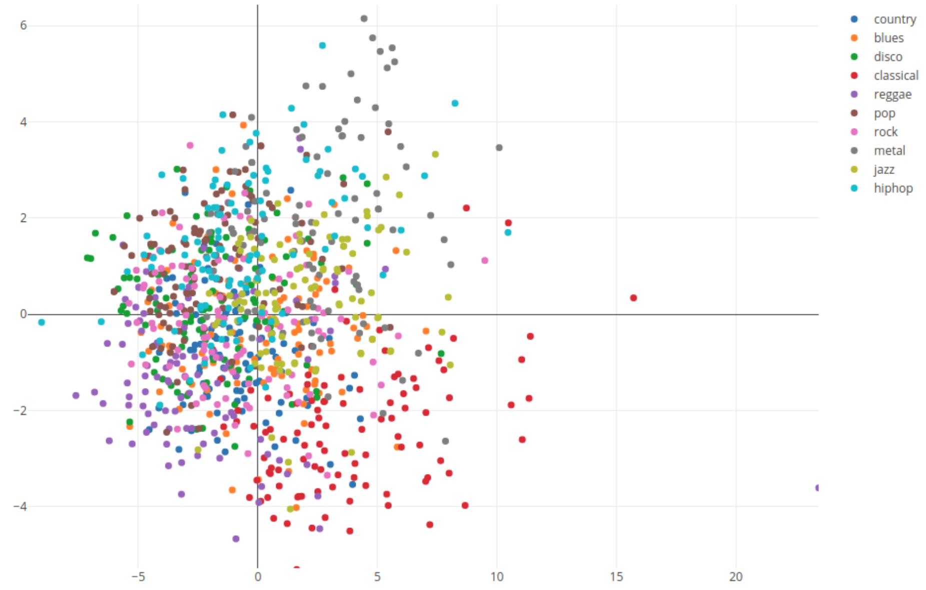


Figure 28: The 2 first PCs after PCA on the predicted tags. Features extracted from the second layer.

5.3 Evaluation

In order to define what features are the best, we have to decide what is important.

The embedding space needs to resemble the human perception of similarity as close as possible. Panning around in 3D gives some information, but with songs scattered other large spaces, and with many genres comparisons are hard. To help average positions, or centers, for all genres, were calculated. From these positions the distance to all the other genre centers was calculated, resulting in a so-called distance matrix. For visualization of the distance matrices, heatmaps are used. The visualizations provide information as on average, what the distance between the genres are. The lower the distance, the higher similarity between the genres. From the heatmaps, we would like to compare if we agree with the nearest neighbors genres and if the heatmaps resemble human perception.

In Figure 29, 30 and 31 the heatmaps of the distance matrices are visualized. Even if the representations are much more compact than a 3D plot of songs, it is still hard to make decisions on what is a better embedding than another. It is also a question that needs to be verified by human perception.

Looking at the heatmap for the last layer in Figure 29, classical music has a relatively low distance to many other genres. We would say that this resembles human perception poorly. On the other hand, the genres with the lowest distance to jazz are blues and country, and metal has a large distance to most genres except for rock which makes sense.

In the heatmap from the third layer in Figure 30, classical music have a large distance to the other genres, and in the second layer, the distance of classical music is even larger. Looking at the heatmaps, we believe the second and third layer seem to resemble human perception better than the tags extracted from the fourth layer. If one layer would be picked, the third layer looks better from the heatmap point of view. However, distinguishing which one is the better is to some extent a guess when deciding between layer 2 and 3.

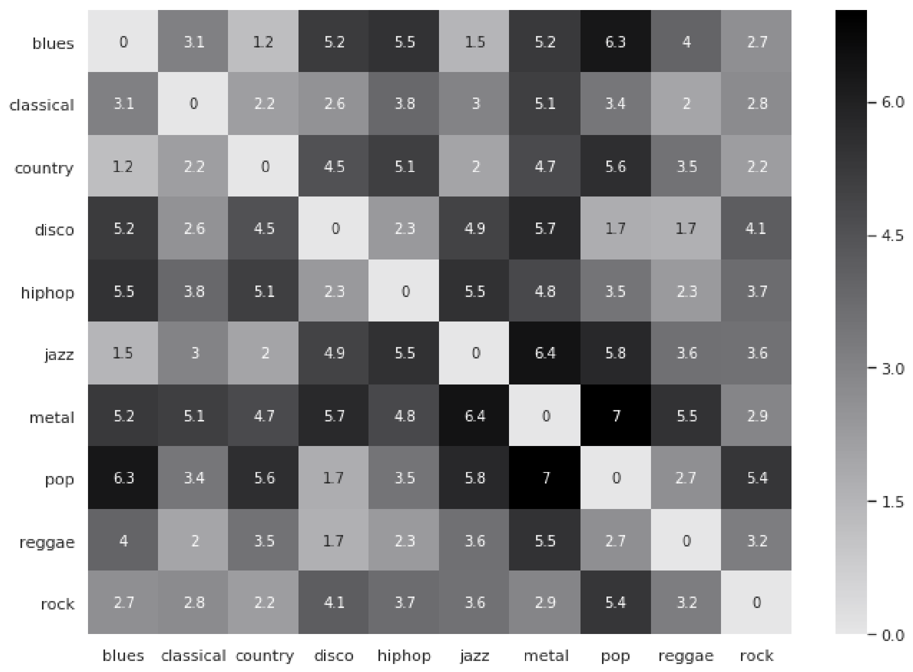


Figure 29: Heatmap of the distance matrix between the genres. Features extracted from the last, fourth, layer.

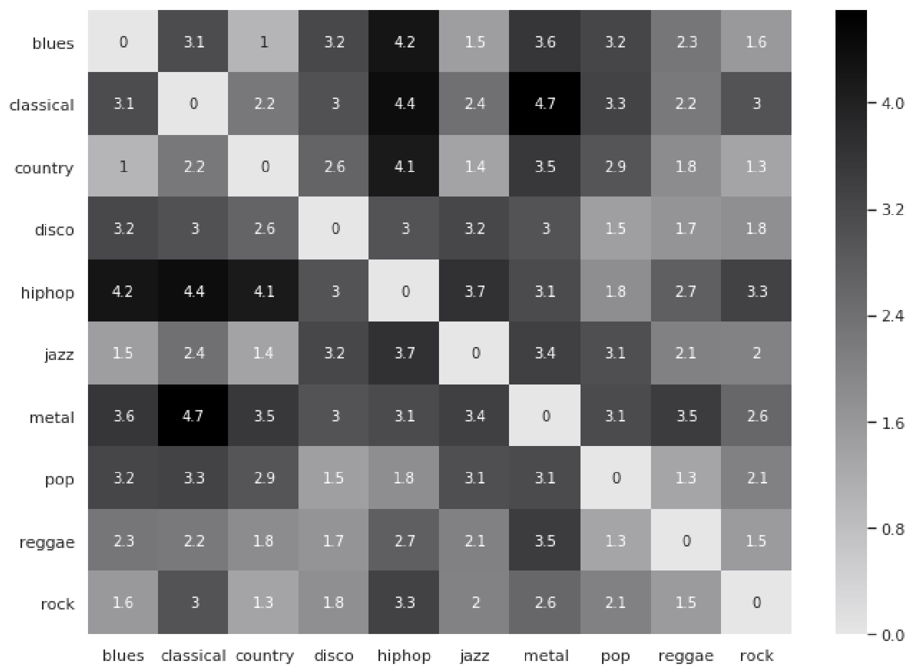


Figure 30: Heatmap of the distance matrix between the genres. Features extracted from the third layer.

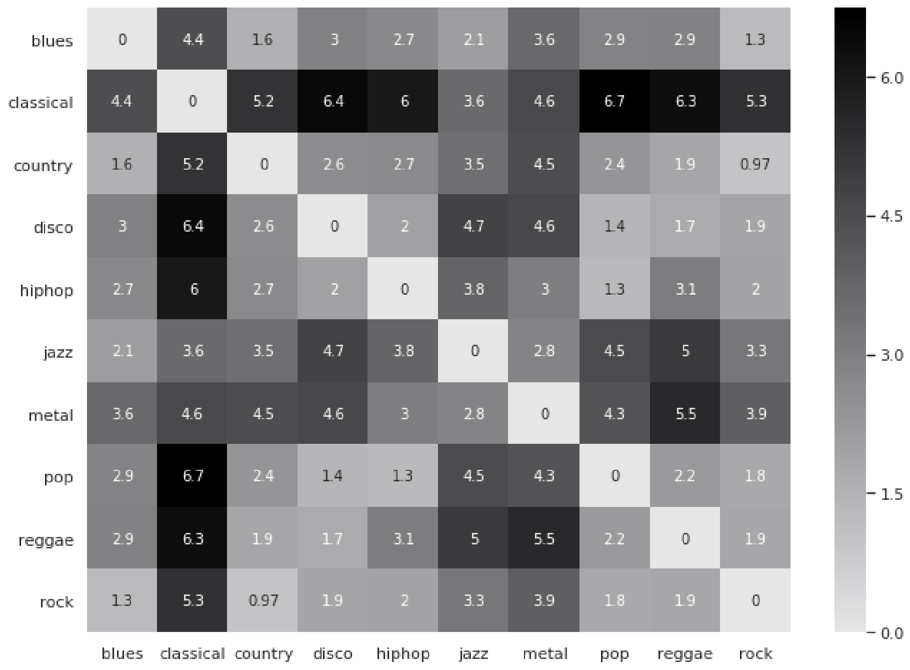


Figure 31: Heatmap of the distance matrix between the genres. Features extracted from the second layer.

6 System evaluation

For evaluation, we choose a qualitative approach mixed with quantitative metrics. For every genre, 10 random songs were sampled, without replacement, and used as the target. The complete system used from feature extraction and how it is combined with the DRL system is visualized in Figure 32.

A real user(we) listened to the target song. A song sampled from the starting position is then sampled and played for the real user. The starting position is always the same, and is the center coordinate($[2, 2, 2, 2]$). As mentioned before, when viewing the 3D plot, songs from the same genre are spread out over a large space, and often colliding with clusters from other genres. The results will, therefore, be very different depending on which song that is being sampled as the target song. Therefore the more songs sampled per genre, the better the evaluation should be. Unfortunately, this is a time-consuming task, and just using ten songs per genre took roughly 4 hours. Only the first 8 seconds of the clip is played. After every song is played, the real user rates the new song and gives feedback if it was better or not than the previous song. The feedback is stored and not fed into the DRL network. Since the network is explicitly trained on simulated users that always acts deterministically it cannot work with real user input.

The real user's feedback is then compared to the feedback fed into the system provided by an ideal user. In table 1 the mean absolute distance between the rating feedback, provided by the real user, and the rating provided by the ideal user, grouped by genre, is shown. From Section 4.4, we know that the DRL solution does find the target within a reasonable amount of steps. The evaluation, therefore, mostly evaluates how the ideal user compares to a real user, and how similar the songs recommended are. After roughly three to six steps, we have noticed that the systems find the target. To make sure that the DRL system finds the target, the rewards given for the position were printed after the real user-provided feedback for the played song. If the real user rates differently than the ideal user, that is a sign that the human perception and the system do not agree on the similarity. Also, since the DRL model should have an exploration strategy that could favor some genres that could be explored at an earlier step than others. As an example, some genres perhaps have more songs close to the center, start position. If a rating of three is fed back from the ideal user, the model knows that the target is close, which would make it easier to find the song. Remember that only ten percent of the positions have a rating of three. Another thing to note is that if the position chosen by the model does not map to any songs, a song from the closest distance is sampled. This could affect the human perception of similarity somewhat. In 4D there is $5^4 = 625$ position available so hopefully, this does not affect the results too much.

In Subsection 6.1 the quantitative results are shortly discussed and in Section 6.2 we discuss qualitative results in terms of how it was to interact with the system and if the recommendations make sense.

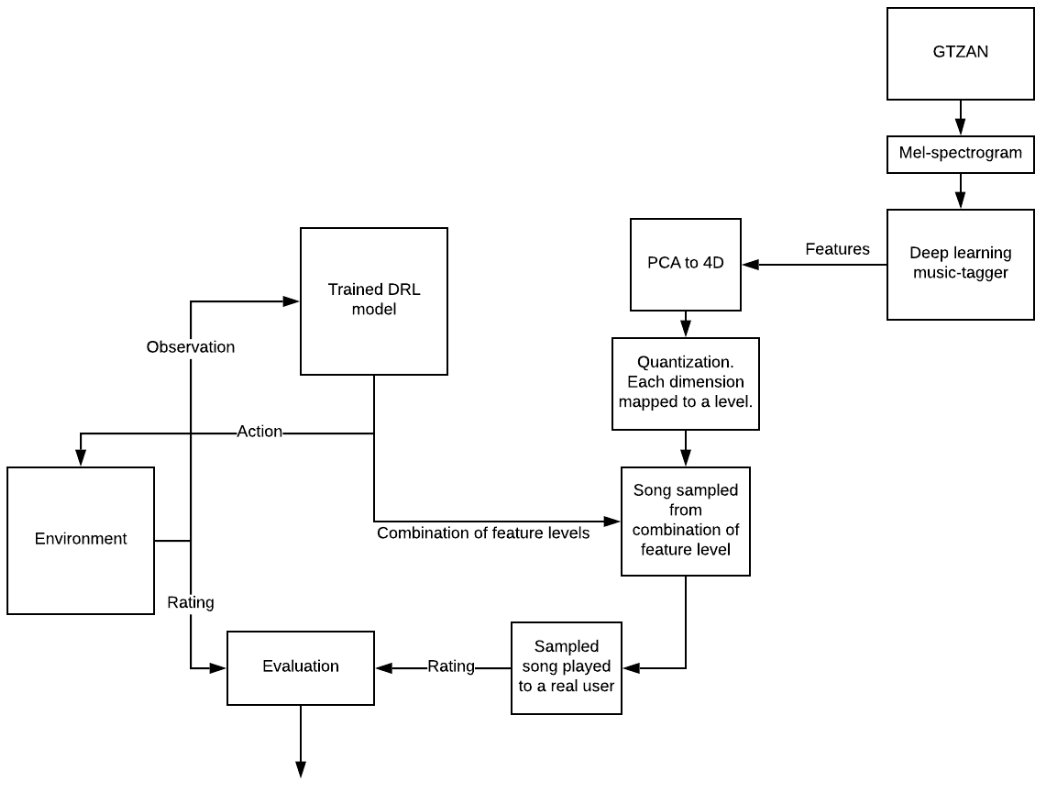

GTZAN
Mel-spectrogram
Deep learning music-tagger
Features
PCA to 4D
Quantization. Each dimension mapped to a level.
Song sampled from combination of feature level
Trained DRL model
Observation
Action
Combination of feature levels
Environment
Rating
Evaluation
Rating
Sampled song played to a real user

Figure 32: The combined system used for evaluation.

6.1 Quantitative results

The classical music was perceived as very different from the other genres. Even if the target was a classical music song, other songs were played and given rating 4. If a song was not classical music when the target was, the rating was almost always 0 except for some jazz and blues songs. This explains the large distance between the real user and the ideal user on classical music. Hiphop had the lowest distance between the real and ideal user. As with classical music, hiphop has a distinct sound. Luckily when trying to find a hiphop target, often already at the third step another hiphop song was recommended. After a hiphop song was found, songs from other genres were very seldom recommended. Target songs from other genres were often not found until the fourth to the sixth step. Looking at the 3D plot, hiphop songs are all over the space so this was a surprise. It could be that the 4th dimension adds information that distinguishes hiphop songs from others, this could potentially be visualized for confirmation.

6.2 Qualitative results

When listening to music recommended by the system, a few things were realized. Firstly, rating deterministically is hard. The comparisons become both biased by the user's preference of music and also the last few songs that have been played. The songs from the genres blues, rock and country were hard to distinguish. In some cases, it felt like the genre was arbitrarily chosen. Two of the three closest genres to rock is blues and country in the feature space used, visualized by the heatmap. Classical music stood out as being completely different from the other genres. The experience of listening to the songs recommended was severely damaged when classical music was recommended while searching for a target from another genre.

Rating songs with 0 felt wrong in many cases. In the setup, most of the positions would have a rating of 0. Perhaps 1 should be the most common followed by 2 and then 0.

Overall, the recommendations do make sense. A target song with more instruments does generate songs with more instruments in most cases. The same goes for vocals, and if a song is more or less quiet. Recommendations are not ideal, but not randomized, either. Most of this information can be reasoned from the 3D plots.

While listening, it is the recommendation often found a song and then played it over and over again, even if it by a human is not similar. Looking at the rewards, it turns out that the song receives the highest reward. One way of coping with this would be to sample songs from an area close. Some difference in the songs is likely a good idea. A reward function that rewards more exploration could perhaps be used. Some of the songs in the area should be somewhat similar to the target song.

Following this resoning, we realize that the features from the second layer could have been the better choice. After listening and comparing so many songs, some of the above points seem to fit better with the second layer than the third layer by looking at the heatmap of the distance matrices. As an example, classical music has a larger distance to all the other genres.

7 Conclusions and future work of the individual parts

7.1 Reinforcement learning

7.1.1 Conclusions

In Section 4 we defined an environment suitable for the training with a DRL algorithm and model. The idea of the environment is to find a preferred setting of features, that a user prefers. Every feature, dimension, has 5 different levels. This is done by user-interactions, and in the environment users are simulated. To simulate users, heavy assumptions on behaviour was made. The algorithm

Genre	distance
blues	1.14
classical	2.12
country	1.49
disco	1.10
hiphop	0.80
jazz	1.60
metal	1.40
pop	1.43
reggae	1.24
rock	1.35

Table 1: Difference in the mean absolute distance between the real users and the ideal users rating feedback grouped by the genre.

in [9] was forked and the environment implemented. The main algorithm evaluated was A2C, but also experiments with ACKTR was conducted, with ACKTR requiring less iterations of training for convergence with the tradeoff of computational cost. Experiments was first conducted in 2D, for faster training, to identify important hyperparameters, and try different reward functions, feedback alternatives and different ways of representing observations. The single most critical hyperparameter found was the n in n-step returns. Values of 5, 10, and 20 was tested, for n, with 10 performing better than 5, and 20 better than 10. A configuration was selected and tested on 3D, and 4D with promising performance. The required amount of iterations for convergence for the A2C algorithm was around 12k in 2D, 40k in 3D and 130K in 4D. Unfortunatly, bottlenecks of the implementation did not allow for testing higher dimensions.

We showed that depending on how reward is calculated, and how observations are represented learning training performance increase or decrease. Also, using an RNN seem to stabilize the learning somewhat.

The simulated users are simplistic, but our model is complex. Even though the model is complex, finding the right hyperparameters was not trivial, and without the hyperparameters correctly set, the model did not perform well. With simple simulated users, as in the case of this thesis, we believe a less complex approach is to be recommended. A simpler approach, perhaps even rule-based, could perform better although, simpler approaches might not be adaptable to when increasing the complexity of the simulated users.

Another thing to highlight is the choice of extending the openAI environment. By extending the openAI environment, these environments can easily be shared, and anyone could benchmark their algorithms on the environment.

7.1.2 Future work

The idea of using DRL to solve the problem was reasoned from that they could perhaps be adaptable to complex user behavior. Unfortunately, we were not able to test complex user behavior within the scope of the thesis, but we at least showed that with a deterministic simulated user providing user-interactions, learning is possible. To conclude how appropriate these types of algorithms are for this type of problem simulated-users needs to act more complex, and not always deterministic. Examples of how to incorporate more complexity could be that features are of unequal importance, users rate differently, or include random feedback. If the simulated users act more complex, training would require more iterations and it is likely that the variance would increase. Even if experiments in 5D could have been made, large actions spaces is a known problem for actor-critic algorithms. The approach from [6] could be applied to train a DRL system with more dimensions. We believe however that the most important thing would be to increase the complexity of the simulated user.

7.2 Feature extraction

7.2.1 Conclusion

In Section 5 the GTZAN dataset was selected, and for extraction of features a deep learning music-tagger used. Features from three different layers were extracted, and with PCA the dimensionality was reduced to 4D in order to fit with the DRL system. A qualitative evaluation was done, which showed that the resulting embedding space was different depending on what layer features was extracted from. Out of the three extracted layers, the two earlier layers provided an embedding space that was argued to be better than the features extracted from the last layers. Although none of the features were perfect. In many cases, nearest neighbor songs were from another genre. In some cases that makes sense, but in other cases, it does not. For example, if the nearest neighbor song of rock would be a classical piece. The features, therefore, bottlenecks the recommendation system, since two songs that have low distance between the features could be perceived as dissimilar as to the human perception of sound.

7.2.2 Future work

Music is complex, and even representing it accurately in 4D would have been astonishing. With more PCs, the embedding space could perhaps be more accurately represented. Simple quantitative metrics, such as measuring if the nearest neighbors belong to the same genre, nearest neighbor accuracy, could be used to compare if more dimensions would be helpful. Comparisons with a genre classifier could also be of interest. A proper benchmarking dataset of music, where users have rated similarities between songs would be of great value. With such a dataset, any features could be extracted and be compared to see if they agreed on the users' perception of similarity.

7.3 The recommendation system

The features and the DRL system was implemented into a system and recommendation qualitatively compared. As expected, the features bottlenecked the performance. Some genres were more easily found than others, indicating that the DRL has an exploration strategy which explores some areas before others. To improve, approximate matching could be done in 4D. Once a target song is found, the k closest songs in 4D could be retrieved and then an exact matching done in the 50 original dimensions.

8 Conclusions and future work

In this thesis, we divided up the problem of recommending music into two problems.

In the first problem, which was to sequentially recommend music to find a combination of feature levels the user enjoys, we used deep reinforcement learning together with relatively simple simulated users. The results show that the model learns, and we tested the system up to 4D where 625 actions are possible. With users simulated as simple as now, our model is much more complex than what should be needed.

The second problem, to retrieve similar music, finding a set of features to explain songs in 4D became the main problem. When the two parts, DRL and feature, were combined, it became apparent that the features used did not explain the songs well enough, when transformed by PCA to 4D. Songs that in the feature space was considered similar was not similar to human perception.

If unlimited time and computer resources were available, the next steps would include increasing the complexity of the simulated users to make them closer to how a real user behaves. Also, increasing the dimensionality from 4D to 5D and above would help to represent the songs better. Finally, to cope with the few dimensions required by the DRL system approximate matching could be done in this lower dimensional space. Exact matching of the nearest neighbours in the lower dimensional space could be done with the original 50 features.

References

[1] Thierry Bertin-mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *In Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR*, 2011.

[2] Chih-Ming Chen, Chun-Yao Yang, Chih-Chun Hsia, Yian Chen, and Ming-Feng Tsai. Music playlist recommendation via preference embedding. In *RecSys Posters*, 2016.

[3] Chung-Yi Chi, Richard Tzong-Han Tsai, Jeng-You Lai, and Jane Hsu. A reinforcement learning approach to emotion-based automatic playlist generation. pages 60 – 65, 12 2010.

[4] Keunwoo Choi, George Fazekas, and Mark B. Sandler. Automatic tagging using deep convolutional neural networks. *CoRR*, abs/1606.00298, 2016.

[5] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[6] Gabriel Dulac-Arnold, Richard Evans, Peter Sunehag, and Ben Coppin. Reinforcement learning in large discrete action spaces. *CoRR*, abs/1512.07679, 2015.

[7] Jeffrey Ens, Bernhard E. Riecke, and Philippe Pasquier. The significance of the low complexity dimension in music similarity judgements. In *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017, Suzhou, China, October 23-27, 2017*, pages 31–38, 2017.

[8] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560, 2017.

[9] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

[10] Edith Law, Kris West, Michael Mandel, Mert Bay, and J. Stephen Downie. Evaluation of algorithms using games : The case of music tagging. In *In Proc. wISMIR 2009*, 1201.

[11] Elad Liebman and Peter Stone. DJ-MC: A reinforcement-learning agent for music playlist recommendation. *CoRR*, abs/1401.1880, 2014.

[12] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu.

Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[13] Jordi Pons, Thomas Lidy, and Xavier Serra. Experimenting with musically motivated convolutional neural networks. In *14th International Workshop on Content-based Multimedia Indexing (CBMI 2016)*, Bucharest, Romania, 15/06/2016 2016. IEEE, IEEE. https://github.com/jordipons/CBMI2016.

[14] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc., 2011.

[15] Jan Schlüter. Unsupervised Audio Feature Extraction for Music Similarity Estimation. Master's thesis, Technische Universität München, Munich, Germany, October 2011.

[16] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[18] Klaus Seyerlehner. *Content-Based Music Recommender Systems: Beyond simple Frame-Level Audio Similarity.* PhD thesis, 2010.

[19] Shun-Yao Shih and Heng-Yu Chi. Automatic, personalized, and flexible playlist generation using reinforcement learning. *CoRR*, abs/1809.04214, 2018.

[20] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2643–2651. Curran Associates, Inc., 2013.

[21] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger B. Grosse, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *CoRR*, abs/1708.05144, 2017.