Jukebox: Challenges and Chances in Raw Audio Music Generation

Julian Karlbauer
Computational Creativity / 2021
julikar@stud.ntnu.no

Abstract

The creation of music is a highly creative process and thus is considered as a challenging task to perform by computers. First attempts in computational music generation reach back to 1957 with the Illiac Suite. Since then, much effort went into research for algorithmic generation of credible music. Recent upcoming Deep Learning approaches like Long-Short-Term-Memory (LSTM) networks could be effectively used to tackle the challenge of temporal long-term coherence in music. However, for efficiency reasons, music has mainly been modeled in symbolic audio representation such as MIDI, which does not represent high-fidelity features of raw audio, such as timbre, and thus is less expressive. To create realistic sounding music of any type or genre, first efforts in raw audio music generations have been done, facing the challenges of processing complex temporal distributed data in neural networks. In this report, an overview of the challenges and state-of-the-art in raw audio music generation is given. Further, the possibilities of OpenAI's Jukebox, one of the most powerful publicly available raw audio music generation tools, are explored by conducting experiments in terms of reproducing OpenAI's research results and custom music generation. The results are analyzed in terms of credibility, coherence and sound quality.

1 Introduction

Computational music generation has a long history and started in 1957 with the Illiac Suite[1]. Back then, algorithmic approaches have been used to model the probability of note sequencesin a score. Since then, plenty of work has been done in the field (e.g., applying Markov-Chains or Genetic Programming approaches) yielding improving results over the years. Latest with the rise of Deep Learning, the field did another leap forward. Especially the introduction of LSTMs (Hochreiter and Schmidhuber (1997)) enabled to predict increasingly long probabilistic sequences, which, for music generation can be seen as an autoregressive modelling problem, was an important development. LSTMs have since then be used to generate sequences for symbolic music representation such as MIDI or Piano Roll. Symbolic representations have the advantage of being a compact delegate of real-world raw audio music, which consists of millions of data points for an ordinary few-minute song. Symbolic music representation only requires a fraction of data points in comparison. Despite delivering good results, symbolic representations have the downside of requiring a synthesizer to produce the actual music. Thus, a generative MIDI model is not capable of generating the actual music but rather the score for a machine to play. Synthesizers further do not essentially have the capability to imitate all aspects of music created on real instruments or vocals sung by a human. Such approaches therefore are limited. To capture the entire expressiveness of instruments or the human voice, it is necessary to either improve synthesizers or to generate music directly in the raw audio domain, circumventing the synthesizer altogether. Due to the large amount of data points in raw audio this was hardly doable until the publishing of WaveNet (Oord et al. (2016)). WaveNet uses dilated causal convolutions to efficiently generate raw audio. While being optimized for speech synthesises, WaveNet was already capable of producing music to a limited degree. A couple years later, Jukebox (Dhariwal et al. (2020)) was published, building on top of WaveNet's achievements. Jukebox is, in contrast to WaveNet, designed to generate music and is the current state-of-the-art in the

[1] https://en.wikipedia.org/wiki/Illiac_Suite

field of raw audio music generation. It uses a combination of Vector Quantized Variational Autoencoders (VQ-VAEs) and autoregressive Transformers to bring raw audio music generation a step forward. Some of the remaining challenges are: establishing long-term musical coherence and a slow training and sampling rate. In this report, Jukebox is presented with its strengths and weaknesses. A rough overview of Jukebox' functionality is given along with experiments to reproduce original results. Along the way, challenges and chances of Jukebox and raw audio music generation in general are identified.

2 Background

Jukebox combines already existing architectures into a new and powerful model. The required information to understand all components of Jukebox are covered here.

2.1 Raw Audio Music Representation

Music can be represented in many different ways. For musicians to read, the most common is standard sheet music representation consisting of notes with pitch and duration information. A similar way to digitally represent music is the Musical Instrument Digital Interface (MIDI) representation. It can store information about e.g., timing, duration, pitch and velocity of notes in a piece of music. To play the music, a MIDI synthesizer is needed. The most realistic audio representation is in the form of raw audio which mimics real world audio waves by modelling them in a discrete digital form. As this representation however does not abstract music in a symbolic manner, it is memory consuming. The amount of memory needed is dependent of the sampling rate. Standard CD Audio is sampled at 44.1kHz. An audio file sampled at that rate contains 44100 data points per second of music which is by magnitudes more than symbolic representations. A three minute audio sample consists of about 8 million data points. For that, raw audio representation is capable of grasp all aspects of an original audio signal, which includes timbre, emotion, modulation, environment and more.

2.2 Autoregressive Modeling

An autoregressive model[2] can be described as a fully probabilistic model that predicts and generates a signal based on a number of its own generated predecessor signals. It can therefore generate a whole sequence of signals, such as an audio waveform, based on a given seed signal using the probabilistic model. This model however is hard to design by hand for an arbitrary problem and in the context of Deep Learning is learned and represented by neural networks.

2.3 Transformers

Transformers are neural networks optimized for sequence to sequence (S2S) operations. Until recently, the state-of-the-art for these kind of tasks was the use of Long-Short-Term-Memory (LSTMS) networks. LSTMs introduced the idea of using recurrent connections and attention mechanisms to predict or transform given sequences. Using Transformers, the idea of recurrent connections was abandoned while the attention mechanism was enforced. Broadly said, transformers are fully attention based, non-recurrent S2S models. As of their S2S nature, they are especially suitable for autoregressive modeling problems.

2.4 VQ-VAEs

Vector Quantized Variational Auto Encoders (VQ-VAEs) Razavi et al. (2019) are a special form of traditional VAEs. A VAE consists of an encoder and a decoder. The encoders compresses an input signal to a hidden latent representation while the decoder does its best to reconstruct the original signal from the compressed representation. For this to work, the encoder needs to find a way to compress the input while maintaining valuable information for the reconstruction. During training, the VAE is expected to learn important features of the given problem to be able to perform the task. VQ-VAEs work in a similar way but add a quantization step after the compression to obtain a discrete latent representation in case of a continuous input signal. The quantization is done via a codebook C_k with k entries that is

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

also learned during training. The compressed signal is split into n signals which then are mapped to the closest codebook vector.

3 Related Work

3.1 Challenges in Realistic Music Generation

An important work addressing the challenges of realistic raw audio music composition was published by Dieleman et al. (2018) of DeepMind. The authors give a detailed overview of the main challenging aspects in raw audio generation while introducing state-of-the-art concepts to tackle the issues. A more detailed description of using VQ-VAEs for autoregressive problems can be found in their work. They further give an overview of existing architectures in 2018 along with their technical approaches, results and individual challenges. It is important to note that many suggestions that were mentioned in this work where later applied in OpenAI's Jukebox model.

3.2 Generating Music in the Raw Audio Domain

Jukebox Dhariwal et al. (2020) is a recent model published by OpenAI that is capable of generating credible music in a large variety using a combination of existing techniques form previous works. It is therefore important to also mention this former work in the field.

3.2.1 Models

In 2016 WaveNet Oord et al. (2016) was published by DeepMind. WaveNet is a fully convolutional autoregressive model to generate raw audio. It was mainly designed for speech synthesis and achieved results outperforming state-of-the-art TTS systems to that time. A main contribution of WaveNet was the introduction of dilated causal convolutions on temporal distributed data. The introduction of causal convolutions forced WaveNet to not break the autoregressive requirement to only depend on previous signals rather than future ones. The use of dilation can be compared to striding in Deep Convolutional Neural Networks (DCNNs) for image recognition. It allows to extend the receptive field and thus enables WaveNet to trade between long-term coherence (important for music) and local fidelity (important for speech). Despite being designed for speech, the authors also trained WaveNet on a large music database. It turned out that WaveNet is able to produce music to some extent. However the they found that the model struggles with long-term coherence, generating music that changes in pace and style every few seconds. Some audio samples are provided on their website[3]. Despite not being capable of generating long samples of realistic sounding music, WaveNet was the first proof of concept that raw audio generation of arbitrary wave forms is possible using autoregressive neural networks and thus paved the way for future more sophisticated models. As described in Dieleman et al. (2018), further promising work has been done after WaveNet. Mentionable are WaveRNN Okamoto et al. (2019) and SampleRNN Mehri et al. (2016) that use recurrent models. More recent models like Jukebox however do not rely on recurrent connections anymore, thus they are not further discussed here. SampleRNN and WaveRNN further are also not primarily designed for music generation but speech synthesis. The most recent successor that builds upon WaveNet knowledge and solely focuses on the generation of music is OpenAI's Jukebox. It currently is the most sophisticated publicly available model for raw audio music generation.

4 Jukebox

Previously, fundamental components for the understanding of how Jukebox works to generate new music were introduced. In this section, the functioning of all the components in a single system is shown. Further, the capabilities and the challenges of the model are presented. The model description will not go into deep detail, as these can be read in the original paper. Here rather an overview of the system is given to have a foundation for further discussion.

[3] https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

4.1 Architecture

The general idea of Jukebox is, that modelling raw audio in its true form is too complex to perform by state-of-the-art architectures as can be seen in WaveNet. However, modelling in a discrete latent space that represents music might be doable. The overall task therefore is to first, learn to compress and decompress music to and from a discrete latent representation and second, learn to generate new sequences in the latent representation to decompress it to new music. For an overview see Figure 1 and Figure 2 respectively. The learning of a latent representation is done using VQ-VAEs while the sampling in the latent representation, as well as the upsampling is done by autoregressive Transformers.

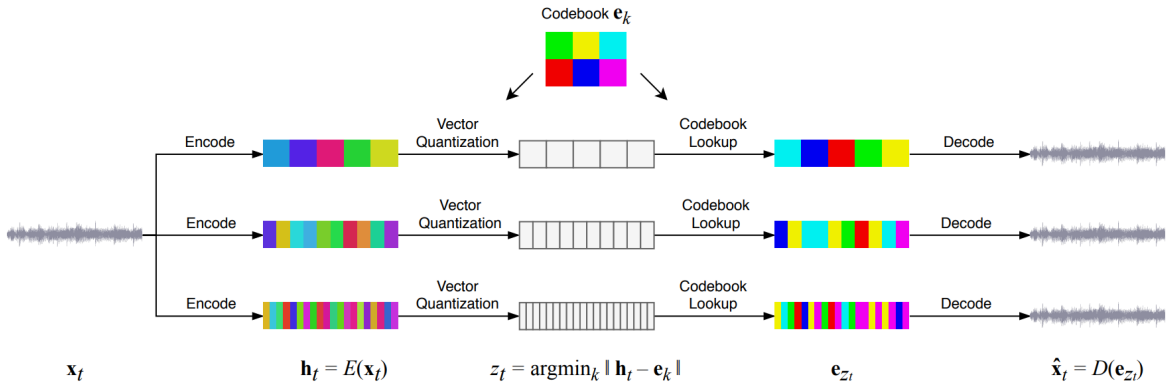

Figure 1: Architecture from OpenAI's original paper Dhariwal et al. (2020). The system is a combination of several models, here: three independent VQ-VAEs to learn discrete latent representations z_t of different granularity to be able to generate music with good global coherence and local fidelity. At each level, the input audio is segmented and encoded into latent vecotrs h_t, which are then quantized to the closest codebook vectors e_{z_t}

.

4.1.1 VQ-VAEs

Jukebox uses three independent VQ-VAEs. As described earlier, VQ-VAEs are capable of learning a discrete latent representation for a given problem. To efficiently represent local fidelity and global coherence, three separate VQ-VAEs are used with different resolution i.e., hop lengths. The topmost VQ-VAE in Figure 1 mainly learns the overall structure (repeating patterns, rhythm, intro, outro, climax) of a song while the bottom most VQ-VAE learns local structure (timbre, phonemes etc.). The training procedure is as follows: An audio file is fed into all three encoders and the data gets encoded into latent vectors h_t. Then, the encoded signal is quantized via codebook lookup, where each latent vector is mapped to the closest codebook vector. After that, the process is reversed. The quantized latent vectors are fed into the three decoders respectively which try to reconstruct the original audio input as $D(e_{z_t})$. The reconstruction loss is calculated via the standard equation (1), bringing the input signal and the reconstruction closer together.

$$\mathcal{L}_{recons} = \frac{1}{T} \sum ||x_t - D(e_{z_t})||_2^2 \tag{1}$$

The process of training robust VQ-VAEs on large music databases is computational highly expensive. OpenAI's original VQ-VAEs trained for several months on hundreds of GPUs on millions of songs.

4.1.2 Autoregressive Transformers

After the three VQ-VAEs are trained, the model is capable of generating music from a compact latent representation. To now generate new music, new latent representations need to be learned. The generation of new latent representations over a discrete space can be interpreted as an autoregressive modelling problem and thus is approachable by Transformers. The problem can be formalized as:

$$p(z) = p(z^{top}, z^{middle}, z^{bottom}) = p(z^{top})p(z^{middle}|z^{top})p(z^{bottom}|z^{middle}, z^{top}) \quad (2)$$

$p(z)$ then is a learned prior over the discrete latent space z_t. Each of the priors are represented as Transformers as they are the state-of-the-art for the given problem as stated by Jukebox' authors. $p(z^{top})$ is now the top-level prior and $p(z^{middle}|z^{top})$ and $p(z^{bottom}|z^{middle}, z^{top})$ mid-level and bottom-level upsamplers respectively, see Figure 2. For music generation a prior is learned. The top-level prior then produces a coarse latent representation on which the mid-level upsampler is then conditioned to increase the audio quality. In the last step, the bottom-level upsampler is conditioned on the output the mid-level upsampler to get the best audio quality. Finally the bottom-level upsampler output is fed into the decoder to obtain the final result.

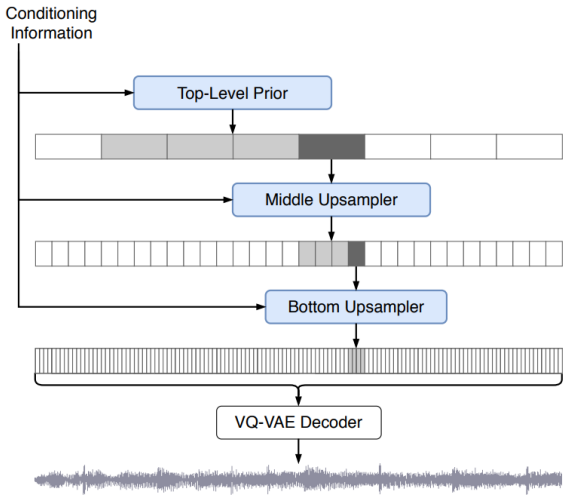

Figure 2: Three autoregressive Transformers in blue. The top level prior learns to generate new sequences in the discrete latent representation. The middle and bottom upsampler are of the same nature as the top-level prior but are conditioned on the respective previous level to upsample the input to a higher audio quality. The output of the bottom upsampler gets fed into the Decoder to obtain the final audio.

4.2 Lyrics Conditioning

One new feature that Jukebox introduces is the ability for the model to condition on lyrics and thus not only produce basic music but also to sing along. The authors recognized that, as the training data consists of complete audio files including singing, the model started to produce phonemes along the instrumental parts. These phonemes however did not build up to understandable English words. To help the model to produce real words, the authors improved the model by allowing it to also process provided song lyrics in text form. A challenge here is to map the text on the song length so that the timing is correct. Eventually, the encoder-decoder architecture is slightly modified via model-surgery to also allow lyrics conditioning. A more detailed description can be found in the original paper.

4.3 Challenges and Results

Having introduced the main concepts of Jukebox, here the main challenges and results of the model according to the authors are presented as a foundation for later discussion in the experimental part. The model is judged in several categories, namely: Coherence, Musicality, Diversity and Novelty. Here is a quick overview of the findings. Thereafter technical challenges are mentioned before presenting Jukebox audio results.

4.3.1 Model Behavior

Coherence: The model is able to produce coherent music to a certain degree. This however depends on the context length the top-level prior is trained on. In the standard configuration, music shows coherence

for at least 24 seconds which is an improvement to other models. Repeating patterns like choruses is something that the model does not grasp.

Musicality: The model generates familiar harmonies and matches lyrics in a realistic way. In genres with fast lyrics, such as rap, the model produces less good results. In general, Jukebox' created music appears less interesting than music created by humans.

Diversity: The model is able to produce diverse samples of various genres and styles. Some genres are more likely to produce good results. Hip-Hop and Rap with fast and uneven distributed lyrics tend to be more difficult for the model to sample properly.

Novelty: The model can generate novel music in the ancestral sampling mode in the style of arbitrary artists. Jukebox' struggles generating music on contradictory inputs, such as 'Frank Sinatra' in the 'Rap' genre. In such cases, the artist dominates over the genre. When inputting novel lyrics, the model sometimes struggles with pronunciation. Writing the lyrics as they are spoken helps. Jukebox is able to generate novel riffs based on a conditioned input.

4.3.2 Technical Challenges

These are already promising findings, however, there are technical challenges that limit Jukebox in performance and practical use.

Raw Audio Complexity: Without a doubt, the complexity of raw audio data is a problem as described earlier. The huge amount of data points in a typical audio file sampled at 44.1kHz is a challenge to process quickly. For a comparison: The amount of data points in a 3 minute audio file is about the same as in 3 Full-HD (1920x1080) images. The learning process thus is time consuming and dependent on high-end hardware.

Real Time Sampling: For music generation models to be helpful in every-day usage, it needs to be real-time capable or at least close to that. Due to the autoregressive nature of the sampling process, the real-time goal is not achievable with today's hardware. The process of subsequently sample the audio point-by-point is not efficient enough for an interactive application on the current state of technology. Further improvements in sequence generating models is crucial here.

4.3.3 Interactive Sample Explorer

To get an overview of generated samples by Jukebox, an interactive file explorer is provided on their website[4]. It presents hundreds of samples and gives a good overview of the capabilities of the model. The explorer can also be used to search for specific genres, artists or training methods.

5 Experiments and Results

In this section, experiments in terms of reproduction of Jukebox' results and practical use-cases is done. In a first part, Jukebox is set up using the pretrained VQ-VAEs and Prior in different environments to investigate accessibility of the model. The model is then used to generate custom music using different parameter settings to explore a variety of possibilities. To see how the model might contribute artists in daily life, a custom guitar-song is recorded and fed into the model to be continued and provide inspiration for the creator. All results are documented on Google Drive[5].

5.1 Experimental Setup

OpenAI published their entire code for Jukebox publicly on Github[6]. It is therefore possible to explore what specifically happens under the hood of the model and actively experiment with custom settings. In this section, the process of setting up and using Jukebox is described.

[4]https://jukebox.openai.com/

[5]https://drive.google.com/drive/folders/1V4cx9Yb4NfJe1UEEVhWk38wMQ1xXX27q?usp=sharing

[6]https://github.com/openai/jukebox

5.1.1 Google Colaboratory

As the model requires immense processing power, the authors also provide an Google Colaboratory Notebook[7] to interactively use the model online. Google Colaboratory provides an interactive online Python notebook, to run UNIX and Python commands in code cells while offering the possibility to connect to one of Google's GPUs to remotely access higher processing power in a free or paid subscription. As Google however started to heavily restrict free usage by introducing time ambiguous limitations, it is very hard to produce reasonable results there. In the scope of this report, it was not possible to generate results using the provided notebook.

5.1.2 Kaggle

Another free interactive online Notebook is provided by Kaggle[8]. Kaggle provides a free GPU for each user for around 40 hours per month. GPU usage is limited to 9 hours at a time. These constraints are much less strict than on Google Colaboratory. To see whether Jukebox is executable on Kaggle Notebooks as well, the code was ported from Google Colaboratory to Kaggle, requiring minor modifications to run. An accessible Kaggle version of the Google Notebook is also provided by a Kaggle user[9]. Using the ported version on Kaggle allowed to produce first results. Success however often depends on getting a powerful enough GPU assigned, as this process seems to be random or dynamically controlled. In the worst case, hours might pass until a powerful enough GPU is assigned to the notebook. Due to the named restrictions and required effort to up- and download custom data into and from a Kaggle notebook, this solution is also not suitable for extensive use.

5.1.3 NTNU Idun HPC Cluster

The final set up thus is done on the Idun HPC cluster of NTNU. The cluster provides an UNIX environment along with plenty of powerful GPUs (e.g., P100, V100) for NTNU members. During the time of experimentation, shareholder accounts allowed to consistently get access to any required GPU (V100 in this case). Getting used to the HPC environment took some time but significantly sped up the experimentation process later on, as data transfer as well as consistent GPU access was simplified. To execute Jukebox on the cluster following steps have been done:

- clone the Jukebox repository into the HPC environment.
- connect to an interactive GPU session using the $srun$ command.
- load dependencies into the session:

```
module purge
module load Anaconda3/2020.07
module load mpiP/3.5-iimpi-2020b
```

- execute the respective python script to sample music

5.2 Experiments

There are many aspects of Jukebox to do interesting experiments on. Due to time limitations however, the experiments in this report focus on some fundamental functionalities of Jukebox only. The main goal is to be able to reproduce basic Jukebox results locally to get a better overview of the functionality of the model as well as the quality of the results. A sub-goal is to apply Jukebox in a creative manner to explore how it can be beneficial for artists when creating music. For each sampling process, a single NVIDIA V100 GPU was allocated.

[7] https://colab.research.google.com/github/openai/jukebox/blob/master/jukebox/Interacting_with_Jukebox.ipynb
[8] https://www.kaggle.com/
[9] https://www.kaggle.com/rkuo2000/jukebox

5.2.1 Sampling with Jukebox

For standard sampling with Jukebox, the following hyper parameters need to be set: model, sample_-length_in_seconds, total_sample_length_in_seconds, sample rate, number of samples, hop_fraction. For "model" it can be chosen between 5b, 5b_lyrics and 1b_lyrics which are the pretrained models provided by OpenAI. 5b and 1b thereby mean 5 billion and 1 billion parameters respectively, thus indicating the model size. 5b is considered being more powerful while 1b is more lightweight both in results and computational requirements. "sample_length_in_seconds" defines how long in seconds the individual samples should be, while "total_sample_length_in_seconds" defines how many seconds of audio are generated in total. "sample rate" defines at which rate the audio is sampled and thus determines the audio quality (44100 Hz for standard CD audio). "number of samples" sets how many individual samples are generated in parallel and "hop_fraction" is a parameter to adjust the context length.

An example command to start the sampling can look like this:

```
python jukebox/sample.py --model=5b_lyrics --name=sample_5b --levels=3
--sample_length_in_seconds=30 --total_sample_length_in_seconds=180
--sr=44100 --n_samples=3 --hop_fraction=0.5,0.5,0.125
```

5.2.2 Experiment 1: Reproduce Jukebox Results with Standard Configuration

Of course it is out of scope to reproduce the entirety of OpenAI's results here. However, it is to be investigated what results are delivered using the standard out-of-the-box configuration after cloning the project. To be able to compare the different models, it is sampled from all three of them. As the sampling process is computationally demanding, the sample length is set to a value between 20 and 30 seconds. To not run out of memory, the number of generated samples needs to be lowered when running the 5b models. The parameter is set to 6 and 3 respectively. The total sample duration is consistently set to 180 seconds. In total, three runs are conducted. For the "lyrics" models, standard lyrics, standard genres and standard artists are provided in the $sample.py$ file. The model will sample in the respective style. In the standard configuration, five entrances are present: Alan Jackson – Country, Joe Bonamassa – Blues Rock, Frank Sinatra – Classic Pop, Ella Fitzgerald – Jazz, Céline Dion – Pop.

5.2.3 Experiment 2: Generating Custom Results with Custom Configuration

It gets specifically interesting, when playing around with the model to produce music on custom artists and custom styles. Therefore the model gets adapted to sample in the style of Eminem, Adele, Rihanna, Billie Eilish, The Beatles, System Of A Down, Ed Sheeran and Queen with respective genres. To do so, the $sample.py$ file is changed to now fit the new artists and genre. In another run, we change artist and genre to 'unknown' to see how the model behaves without any conditioning. Multiple runs were conducted conditioning on various artists and genres. In some cases, a second sample run has been done to extend the audio sample length to about a minute. To achieve the best possible result, the 5b_lyrics model was used.

5.2.4 Experiment 3: Prime Jukebox with Custom Music for Continuation

A main motivation of music generating models might be the support of creative work. The possibility of Jukebox to prime on input audio and then continue it in given or adapted styles can be used as an supportive tool for music production. To try this, a custom guitar instrumental has been recorded that is then fed into the model. The recording can be found on Google Drive in the 'Experiment3' folder. To prime the model, the first 15 seconds of the original guitar recording were taken. In total, three different model configurations are tested: First, the 5b_lyrics model with standard artists and genres, second: the 5b model with standard artists and genre, third: the 5b_lyrics model without provided genres and artists to pick up the style of the input.

5.3 Results

Here the results for all experiments are presented without being discussed. A detailed discussion on the results will follow in the next section. All of the over 40 generated audio files can be accessed on Google

Drive[10]. See folders Experiment1 , Experiment2 and Experiment3 respectively.

To generate a 20 second audio file in .wav format, about 3 hours computation time on the provided hardware was needed. In some cases, song duration has been extended to 40 seconds or 1 minute to see if it helps developing more interesting song structures. Computation time doubled or tripled respectively. Due to computationally and time constraints, sometimes only three instead of six audio samples were generated for a run.

5.3.1 Experiment 1: Reproduce Jukebox Results with Standard Configuration

For each the 5b_lyrics and the 1b_lyrics model, several audio files in .wav format of each 20 - 30 seconds duration with standard model configuration have been generated. A first listen indicates that the model did not generate random audio noise but structured patterns that remind of music. As expected, the model also produced vocals in the samples. The audio files for each model can be found in the 'Experiment1' folder.

5.3.2 Experiment 2: Generating Custom Results with Custom Configuration

The 5b_lyrics model was not able to condition on the artists Adele and Billie Eilish as they are apparently only covered in the 1b version. In these cases, music conditioned on 'unknown' artist and pop genre was sampled. A first listen indicates that specific artist styles might be recognized. See the files in the 'Experiment2' folder. Results in 'custom_artists_1' where conditioned on Eminem, Adele, Billie Eilish, Rihanna and the Beatles. As some artists where not represented, another run conditioned on Ed Sheeran, The Beatles, System Of A Down, Queen and Eminem is stored in 'custom_artists_2'.

5.3.3 Experiment 3: Prime Jukebox with Custom Music for Continuations

See the 'Experiment3' folder for the results. A first listen indicates, that the priming worked in so far, that the model could reconstruct the original priming input. Further, in some occasions it seems like the model adapts to the input and is able to continue it.

6 Evaluation and Discussion

6.1 Experiments

In this section, the final Jukebox results are discussed with respect to expectations and quality. Before entering the discussions it is worth mentioning that using a model that does not support lyrics, does not mean that jukebox does not add singing to the song. Depending on the provided genre and artist, vocals are added but tend to be not understandable.

6.1.1 Experiment 1

The results obtained are similar to the ones provided by OpenAI on their website. The model generated music in various styles with vocals. In the generated samples, no big differences between the 1b_lyrics, 5b_lyrics and 5b models could be recognized. On could argue that the 'lyrics' models generate better 'understandable' vocals than the normal 5b model. In any case however, the lyrics are not very clear. It appears like it is somewhat random if understandable lyrics are produced or not. Taking a look on '5b_lyrics/item_0.wav' demonstrated clear singing in the end but the words do not exist in real or are not relatable. In terms of generating understandable and clear lyrics as in the original samples from OpenAI the here performed experiments were not successful. All models however do without a doubt generate music in the style of the provided artists and genres provided in the standard Jukebox configuration.

6.1.2 Experiment 2

Here the model has been modified to sample in the style of custom artists. This is an interesting tool as it allows steer the results into preferred directions. Of course, not all genres and artists could be tested here but a small overview on how Jukebox behaves in that manner can be given. In all runs, the model was not able to generate convincing music in every aspect, however it was able to mimic specific artists styles to a certain degree. In the first run, item_0.wav and item_5.wav for example grasped characteristics

[10]https://drive.google.com/drive/folders/1V4cx9Yb4NfJe1UEEVhWk38wMQ1xXX27q?usp=sharing

of Eminem in the middle or end part. Especially the voice sounds realistic. Problems with the beat and overall structure as well as lyrics pronunciation however occur. Other samples in the first run successfully imitate aspects of pop music. The second run yielded clearer examples. Here, in all of he provided samples the style of the given artists can clearly be recognized. The worst results are the Ed Sheeran samples (item_0 and item_5), as his voice is not well represented. Characteristic guitar and rhythm however can be heard. Especially good is the Queen imitation in item_3. Also the the characteristic style of The Beatles in item_1 is recognizable. Generally spoken however, lyrics are not understandable which is also because no lyrics were provided in text form for this experiment. The music is not on a level to compete with the originals in any matter. In retrospect it would have made more sense to provide some sort of dummy lyrics, as it supports the model in generating them as can be seen in the unknown category. As here no artists, genres and lyrics are provided, the model focuses on instrumental music as it best suits the requirements. In the above case, the model probably also tried to avoid generating lyrics but failed because lyrics are such an integral part of the training set. Therefore the lyrics are blurred into the music itself.

6.1.3 Experiment 3

Generative music models might in future be used as a supportive tool in music production. The results produced here are already quite promising in terms of assisting a creator in a creative field such as music production. Remember that the original audio lasts for 15 seconds in the sample. Everything thereafter is generated by Jukebox. Most of the continuations are not really usable but a few of them capture interesting ideas, are able to continue the mood and sound of the provided input. Especially good is the result in 'prompted_5b/item_0.wav' as it captures the guitar playing and adds nice voicing while maintaining key and rhythm of the original sample. Also the transition from sample to continuation is rather smooth. This example could very well be an inspiration for song continuation. Another remarkable example is in 'prompted_5b_lyrics_unknown_artists_no_lyrics/item_3.wav' as it adds interesting variations and emotions through the singing. Other samples as well capture the guitar playing and even continue the sound of fingers sliding over the strings. Still there are plenty of samples that do not meaningfully continue the song.

6.2 Audio Quality

It is noticeable that the generated audio quality is not comparable to today's standard. The audio in general sounds flat and a bit unstructured at occasions. It would be interesting to have a quantitative measure of the audio quality generated by Jukebex vs. today's common audio quality that is mixed in studios. This comparison could help identifying Jukebox' weaknesses in terms of audio quality. A good way of visualizing audio is using Mel Spectograms, see Figure 3.

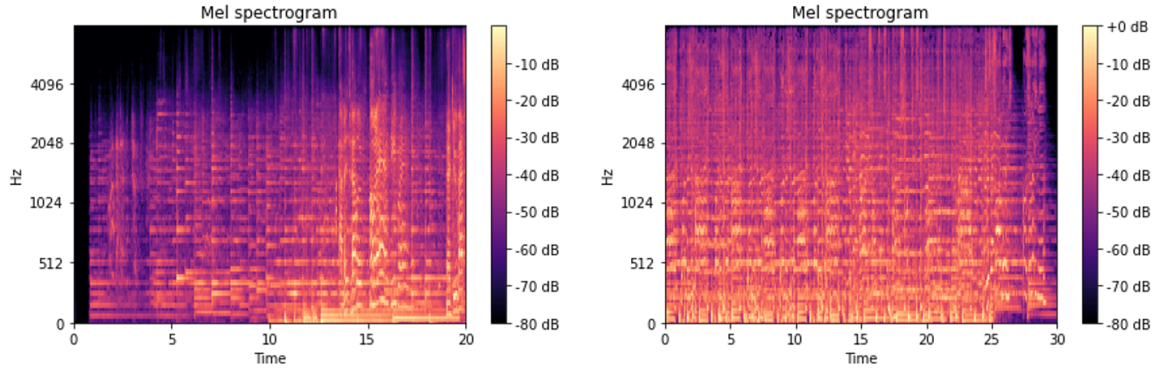

Figure 3: Mel Spectograms of two different audio files. On the left side, an audio file generated by Jukebox is visualized while on the right side a high quality audio sample downloaded form Spotify is visualized. Differences in frequency distributions are clearly visible.

It is clearly visible that frequencies in Jukebox' generated audio sample are less evenly distributed than in the high quality audio sample which might be a reason for existing audio noise. Jukebox' sample especially suffers in the high frequency area as it does not represent plenty of high tones. Also in the low frequency area, the Jukebox' sample lacks in power compared to the high quality sample. The audio reconstruction in terms of audio quality therefore is visibly not perfect and improvements need to be done to generate high end audio in the future.

7 Conclusion and Future Work

In this report challenges and chances of OpenAI's Jukebox for raw audio music generation have been explored. An overview of Jukebox' technical aspects has been given and challenges in state-of-the-art raw audio music generation have been mentioned. Especially problematic is the autoregressive nature of sequence sampling and the large amount of data points in raw audio which causes the model to be slow in both learning and sampling. Further, experiments in raw audio music generation have been conducted to explore Jukebox' capabilities. In a first experiment similar results to the original publication could be reproduced as a proof of concept. In a second experiment, Jukebox was used to successfully imitate the style of various chosen artists. In a third experiment it was investigated whether Jukebox can be used in an interactive manner to help in a creative music creation process. While not being perfect, plenty of the generated results already are satisfying and there is much potential for improved systems in the future. For future work it would be interesting to take single components of Jukebox and train them on custom data. Instead of conditioning on a specific artist or genre, music of the specific field can be taken to create custom training set. Training a custom Prior on a custom data set might then yield better results for any specific use case.

References

Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

Sander Dieleman, Aäron van den Oord, and Karen Simonyan. The challenge of realistic music generation: modelling raw audio at scale. *arXiv preprint arXiv:1806.10474*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

Takuma Okamoto, Tomoki Toda, Yoshinori Shiga, and Hisashi Kawai. Real-time neural text-to-speech with sequence-to-sequence acoustic model and waveglow or single gaussian wavernn vocoders. In *INTERSPEECH*, pages 1308–1312, 2019.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *Advances in neural information processing systems*, pages 14866–14876, 2019.