DeepMixing: Music Audio Mixing using Deep Reinforcement Learning

DS856 Neural Networks and Reinforcement Learning

Aditya T*	Ahmad Shayaan*	Bharath N*
IMT2015521	IMT2014004	IMT2015527

{T.Aditya, Ahmad.Shayaan, Bharath.N}@iiitb.org

IIIT-Bangalore

December 7, 2018

Contents

*All authors contributed equally to this work.

1 Problem Statement

1.1 What is Audio Mixing?

In music recording and production, audio mixing is the process of combining multi-track recordings of various instruments into a single output song. The various individual tracks generally represent various different instruments recorded separately, sitting one on top of each other. These tracks are 'blended' together by using various processes such as equalization, compression, panning, reverb, delay etc. which are either applied to whole or parts of the track. Mixing techniques can vary widely based on the skill-level and the intention of the mixer and the output result can be drastically different for different music genres.

Audio mixing can be split into different categories: [1]

1. Processes that affects levels
2. Processes that affect frequency response
3. Processes that affect time
4. Processes that affect space

1.2 Goal of this Project

The goal of this project is to learn a network which can produce good mixes, given the raw tracks and the expected final mixed track. But, here in this project we are specifically targeting processes that affect levels and one of the very first things an audio engineer would do - local + global volume adjustments for each of the tracks.

2 Dataset

2.1 Requirements

The dataset we are using to tackle this problem is required to consist of the following:

1. Should have raw audio files of every individual instrument recorded.
2. There can be multiple raw tracks for each instrument
3. The processed mixed/blended track
4. All raw tracks should be of the same length as the mixed track

In this project, we have chosen to work only with Rock based music genres - as the kinds of instruments used are specific and limited in nature, making it a good test bench to understand how the network is learning.

2.2 Source

The data was downloaded from the audio mixing practice library provided for the book 'Mixing Secrets for the Small Studio' by Mike Senior[2]

It contains a list of multi-track projects which can be freely downloaded for mixing practice purposes. All these projects are presented as ZIP archives containing uncompressed WAV files (24-bit or 16-bit resolution and 44.1kHz sample rate). For maximum mix-down flexibility, the contributors have made every effort to provide audio 'raw', in other words without additional effects or processing (beyond treatments printed during tracking/editing) [2]. The Alternative Rock and Rock sections of this list were used for learning this network.

NOTE: All downloads from this site [2] are provided free of charge for educational purposes only. This project is purely for educational/academic purposes.

3 Data Pre-Processing

The following are steps are to be done for pre-processing the data:

3.1 Curate Statistics

1. Find the various types of instruments present in the dataset, maximum number of sub-tracks for each instrument and the maximum track length
2. Create a consistent ordering for all the instruments across songs (Refer Appendix 1)

3.2 Pre-Processing Steps

The following pre-processing steps were applied on all the songs:

1. While reading every track, sample them at 44.1kHz - Get the NumPy array for all the tracks
2. Create dummy tracks for each instrument, based on the maximum number of tracks present across songs for that instrument
3. Pad all the tracks to longest track/song in the dataset
4. Chunk every track in such a way that each chunk is of a certain dimension. Since all track sizes are the same, the number of chunks in every data should be the same

For reading the audio files and doing the various pre-processing steps as mentioned above, we used librosa library for python. [3]

For every song in every epoch, yield a tuple of (combined raw track, mixed song), where the dimensions of the combined raw track is (number of tracks $\times$ number of chunks $\times$ chunk size) i.e. (C $\times$ T $\times$ chunk size) and the dimensions of the mixed song is (number of chunks

× chunk size) i.e. (T × chunk size). The number of chunks into which it a track is split is also the number of time steps when progressing through the song.

4 Architecture

The architecture consists of a Policy network that learns a policy to sample the scaling factors at every time step (scaling factors here refer to the factor by which you want to increase/decrease the volume/levels of individual tracks at each time step). These scaling factors are then used to scale the individual track at each time step and mix them giving the mixed track.

The architecture also consists of a reward engine, that takes the MFCC features of the ground truth as well as the MFCC features of the mixed tracks. The reward engine gives a positive feedback if the MFCC features of the mixed tracks and the MFCC features of the ground truth are close to each other and the scaling is in such a way that there isn't any drastic increase in volume between 2 consecutive time steps.

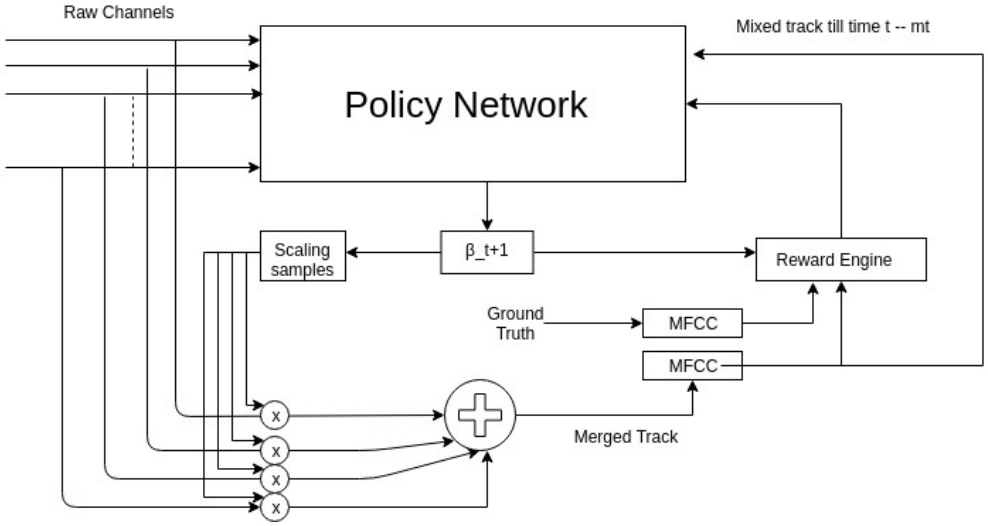

Figure 1: Architecture of the network

4.1 Policy Network

The policy network consists of C (number of raw tracks of the instruments) Bi-LSTMs and 1 Uni-LSTM. The Bi-LSTMs are given chunks of the raw tracks till time step t as input. Each chunk is given as a NumPy array of size 500 (embedding dimension/chunk size). The Uni-LSTM is fed in with the Mixed MFCC features of the raw tracks at time step t.

The Bi-LSTMs are used to generate a hidden representation (size k) of the input chunks till time step t and the Uni-LSTM is used to create a hidden representation (size l) of the mixed song based on MFCC features at time step t.

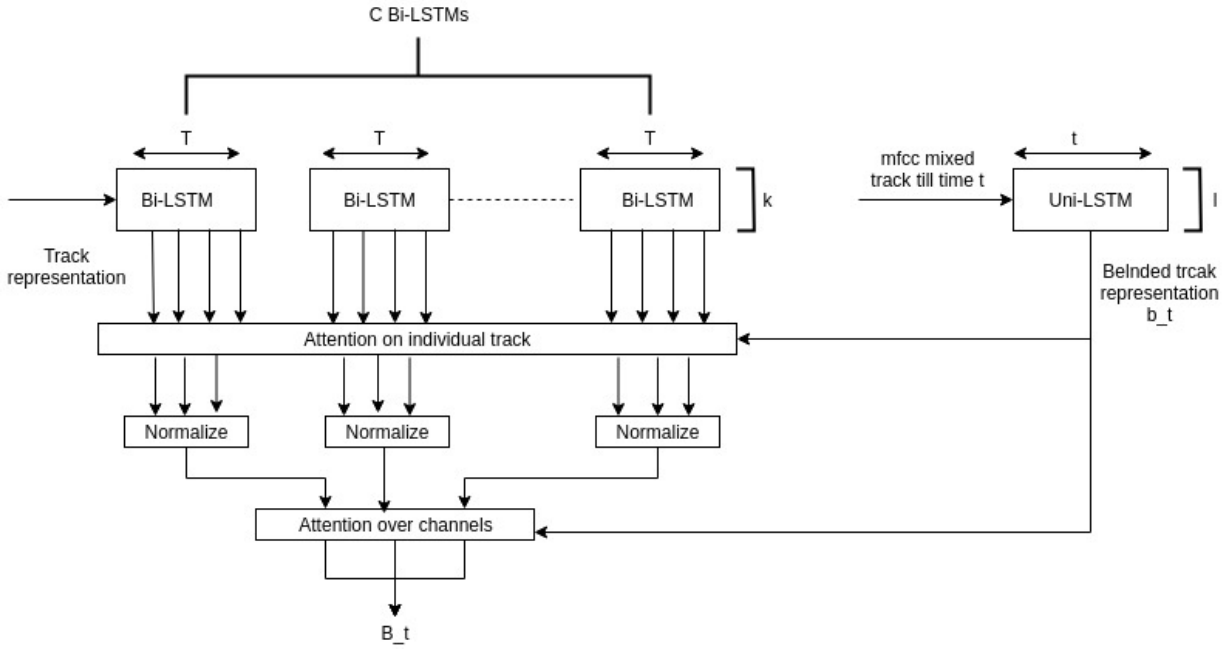

Figure 2: Policy Network

The hidden representation of the tracks along with the representation of the mixed songs is used to find the attention over individual track positions as well as the attention over the different tracks. The idea here is that, we should be able to find how loud which sections of a track should be and what are the relative loudness levels between the tracks. The attention is applied across all the hidden representations across every time step so that we are able to identify the loudest sections of the song and relatively adjust volumes for the rest accordingly. The mechanism to find the attention over the individual track position as well as the attention over the raw tracks is described in the next section.

The output of the attention over the different tracks is used as parameters for a Dirichlet distribution. The policy network then takes a sample from the Dirichlet to get the scaling factors. The scaling factors are then multiplied with the raw tracks and then these scaled tracks are then mixed together (summed up) to get a mixed song as output. The network then computes the MFCC features of the ground truth and the MFCC features of the mixed tracks and feds them to the reward engine along with the scaling factor distributions from both the current and next time step.

The reward engine calculates the reward, which is then fed back to the policy network and is used to learn the optimal policy to learn sample the scaling factors form the Dirichlet.

4.2 Attention Mechanism

The networks calculate two types of attention: First the attention over the individual track position and second the attention across the various instrument tracks themselves. The attention over the individual tracks helps the network to find which portion of the track is

important. Whereas the attention across the instrument tracks tells the network which track is to be focused on more at that time step.

The attention over the individual track positions are calculated by using the hidden representation of the raw tracks given out by the Bi-LSTMs, the representation of the MFCC features of the mixed tracks generated by the Uni-LSTM, and the parameter matrices $(B_1, H_1 \cdots, H_c)$.

The intuition behind these parameter matrices $(H_1 \cdots H_c)$ is to learn an embedding look-up matrix of sorts for the various input raw instrument tracks which in-turn will learn which parts of the song are generally more important than the others.

The Parameter matrices $(H_1 \cdots H_c)$ are each of size (parameter matrix dimension $\times$ hidden dimension size of the Bi-LSTM) i.e. $(r \times k)$ and the parameter matrix B_1 is of size (parameter matrix dimension $\times$ hidden dimension size of Uni-LSTM) i.e. $(r \times l)$. The size of the hidden representation form the Bi-LSTMs is (output of Bi-LSTM $\times$ 1) i.e. $(k \times 1)$. The hidden representation for the i^{th} track at the s^{th} time step is multiplied i^{th} parameter matrix, which are then multiplied with the transpose of the multiplication of the parameter matrix B_1 and the blended representation form the Uni-LSTM. The result is calculated for all the track and then normalized to get the context vector. The equations for the calculation of the context vector are given below.

For i ranging from 1 to C (number of tracks) and and s ranging from 1 to T (number of chunks):

$$\lambda_{is}^{t+1} = (B_1 \cdot b_t)^T (H_i \cdot h_{is})$$

$$\alpha_{is}^{t+1} = \frac{exp(\lambda_{is}^{t+1})}{\Sigma_{s=1}^{T} exp(\lambda_{is}^{t+1})}$$

For the i_{th} track, the attention weights looks as follows:

$$\alpha_{is}^{t+1} = (\alpha_{i1}^{t+1}, \alpha_{i2}^{t+1}, \cdots, \alpha_{iT}^{t+1})$$

The context vector for the i_{th} track is calculated by multiplying the above attention weights with the corresponding hidden representation of the track.

$$\text{context vector } CV_i^{t+1} = \Sigma_{s=1}^{T} \alpha_{is}^{t+1} \odot h_{is}$$

The attention across the instrument tracks is calculated by multiplying the context vector for the i_{th} track CV_i and the parameter matrix (F_i) for the corresponding channel. The intuition behind F_i is similar to that for H_i. The dimension for the context vector is $(1 \times$ output Bi-LSTM) i.e. $(1 \times k)$ and the dimension for the parameter matrices are (parameter

matrix dimension $\times$ hidden dimension size of Bi-LSTM) i.e. $(r \times k)$. The product of the two is then multiplied with the transpose of the product of the parameter matrix B_2 and the representation of the mixed track that is generated by the Uni-LSTM.

$$\gamma_i^{t+1} = (B_2 \cdot b_t)^T (F_i \cdot CV_i^{t+1})$$

$$\beta_i^{t+1} = \frac{exp(\gamma_i^{t+1})}{\Sigma_{j=1}^{C} exp(\gamma_i^{t+1})}$$

The attention weights β^{t+1} for the attention across the instrument tracks are used as the parameters for the Dirichlet distribution. The Dirichlet distribution is used to sample the scaling factors for time step $t+1$ which are then multiplied with the individual instrument tracks at that corresponding time step.

4.3 Reward Engine

After applying the scaling factors to the individual tracks and mixing them, the network then proceeds to calculate the reward signal for the current mixing. The reward signal is generated using the MFCC features of the mixed track, ground truth song and the Dirichlet parameters for the t and $(t+1)^{th}$ time step. The expression for the reward function is given as follows.

At time step $t+1$, at state b_t and action β^{t+1}:

$$\text{reward} = exp(-||M_t - m_t||_2^2 - \rho \cdot KL(\beta^t || \beta^{t+1}))$$

The reward function takes the difference between the MFCC of the mixed tracks and the MFCC features of the ground truth and finds the norm of the difference, which indicates how far is the mixed track from the ground truth. The function also finds the KL divergence between the Dirichlet parameters of the current time step and the next time step. The KL divergence in the reward function will insure that the network dose not bring drastic changes when scaling the songs.

The calculated reward is then propagated back through the network to learn the policy for sampling the scaling factors form the distribution and also learn all the parameter matrices.

5 Conclusion

Due to the massive size of the data set - where each song is about 8GB, training the network is still in progress and thus results aren't ready yet. The report would be updated with the results along with the insights gained from this architecture once it is completed.

6 Appendix

6.1 Appendix 1: Ordering of various instruments:

The following list consists of the ordering used for every song after downloading it from Mixing Secrets Practice Library [2]. This is essential, since the network has to know what scaling factors / filters should be applied for what specific instrument and sub instrument. The maximum number of raw tracks for each sub instrument is taken and those whose track counts are lesser than this number are padded with empty tracks to maintain uniformity.

The various instrument sections and instruments are as follows:

1. Drums
    - Kick
    - Rim
    - Hi-Hat
    - Overheads
    - Snare
    - Cymbal
    - Bass
    - Drums
    - Percussion
    - Tom
    - Room
    - Ambience
    - Extra
2. Strings
    - Electric
    - Ukulele
    - Bass
    - Acoustic
    - Extra
3. Keys
    - Organ
    - Piano
    - Harmonica

- Synth
- Extra

4. Vocals
    - Lead
    - Backing
    - Extra
5. Woodwind
6. Flute
7. Overall
8. Extra (overall mix tracks if any)

6.2 Appendix 2: Variables and Model Parameters

1. Number of epochs = 20
2. Chunk Size for a track = Input for the Bi-LSTM = 500
3. MFCC dimension = 13
4. Hidden representation dimension for a single Bi-LSTM = 100 (k)
5. Hidden representation dimension for the Uni-LSTM = 5 (l)
6. Number of raw tracks for each song = 72 (C)
7. Number of chunks (number of time steps) in each song = 2640 (T)
8. Parameter matrix (H and B and F) dimensions = 100 (r)
9. Rho for the reward function = 0.01 (ρ)

Bibliography

[1] Wikipedia. *Audio mixing (recorded music)*. URL: https://en.wikipedia.org/wiki/Audio_mixing_(recorded_music).

[2] Mike Senior. *Mixing Secrets For The Small Studio*. URL: http://cambridge-mt.com/ms-mtk.htm.

[3] *LibROSA is a python package for music and audio analysis.* URL: https://librosa.github.io/librosa/.