Semi-Supervised RNN-GAN for Audio Chord Estimation

Charles Zhijie Chen
Department of Electrical Engineering
Stanford University
zcchen@stanford.edu

Elizabeth Chu
Google
elizachu@stanford.edu

Corey McNeish
Lawrence Livermore National Laboratory
cmcneish@stanford.edu

Abstract

In this study, we present a semi-supervised learning chord recognition system based on GAN and RNN. Unlike image recognition or NLP, the amount of publicly available labeled audio chord datasets is scanty. To address this, we explore the possibility of semi-supervised learning by implementing a RNN-GAN structure. The hypothesis is that the performance of chord recognition can be further improved by introducing unlabeled data produced by the generator.

1 Introduction

Audio chord estimation is the determination of the current chord being played for a given time slice of aural data. As an example, a chunk of audio could contain a C Major chord for time when it contains frequencies at 523, 659, and 784 Hertz simultaneously.

The applications for chord estimation are limited, but nonetheless interesting. Possible applications include music sentiment analysis, automatic music accompaniment, and assistance in teaching ear training and music theory.

Unfortunately, the presence of given combinations of frequencies over a certain threshold in a snippet of audio is not sufficient to accurately estimate chords. A number of reasons are responsible for this. Firstly, musical instruments produce more than just sinusoids of a given frequency. Tones produced by musical instruments also contain harmonics; other frequencies that cause instruments to sound distinct from one another. Secondly, chord tuning is rarely exact. Differences in temperature and performer fatigue can cause group swings in pitch over tens of Hertz. One other notable reason is that, to better align higher harmonics, musical groups intentionally play some notes in a chord slightly out of tune.

Technical reasons make building learning frameworks for chord estimation difficult as well. For one, very little annotated data is publicly available. The largest dataset we were able to find contains just 740 unique songs [1]. Getting labeling assistance from places like Amazon's Mechanical Turk would be difficult due to the lack of general public knowledge on what various chords sound like. In addition, it is useful to train on complete songs instead of just snippets of music, as the distribution of chords in a given song is less random than the distribution over all songs. However songs become extremely long sequences when binned into chunks small enough to be useful, which heavily impacts training time.

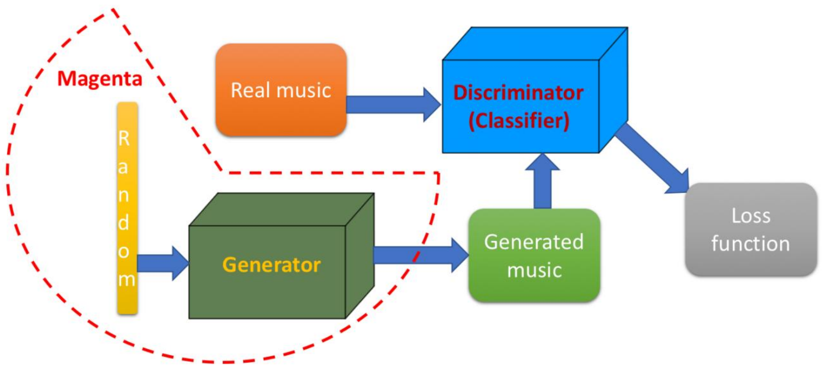

Figure 1: Overall GAN structure of the system.

Figure 1 shows the overall structure of our system. Our input and output data representations are presented in a later section. The discriminator and the generator are both RNNs. The generator produces synthesized music pieces, which is fed into the discriminator. The discriminator performs both the recognition of music chords and the identification of whether the music piece is synthesized or from the training distribution. For our model loss, we used the combined entropy of both the generator and the discriminator tasks. For the scope of this study, we did not train the generator, rather we incorporated the Magenta music generation project[1], a Google Brain development. Correspondingly, we deleted the generator loss from our total loss function, giving a loss of the summed chord recognition and real/fake cross entropies.

2 Related work

[2] introduces the use of Recurrent Neural Networks for chord recognition. In this paper, Boulanger et al. use chromagrams and preexisting chord information to train a state-of-the-art model. This paper achieves a notable improvement in accuracy, with basically no downsides. Minor issues include having to train the network in stages, to build intermediate representations of aural data, and a convoluted dynamic-programming beam search implementation for sequence transformation. Unfortunately, their dataset is not available.

In [3], a GAN was proposed as a way to improve the performance of standard classifiers. Regarding a standard classifier that classifies a data point into one of K classes, Salimans et al. suggested that supervised training can be converted to semi-supervised training by adding unlabeled samples from the GAN generator to the dataset and labeling them with a new "generated class" label $y = K + 1$. The optimization was thus modified to minimize the sum of $L_{\texttt{supervised}}$, supervised loss given by cross entropy of labeled chords, and $L_{\texttt{unsupervised}}$, unsupervised loss being the cross entropy of the last dimension of real/generated label. The mechanism was not fully understood, but our intuition is that by identifying real/generated data points, the discriminator better captures distribution of the real dataset, from which other dimensions (chord labels) may benefit.

3 Dataset and Features

Our dataset includes chromagrams (see Figure 2) of 740 unique songs from the McGill Billboard dataset. These chromagrams are a sequence of 24-dimensional vectors, containing tuning-adjusted and normalized spectra corresponding to bass and treble registers of the standard 12-tone chromatic scale. The dataset also includes labels per time-slice in the standard MIREX format: <start time> <tab> <end time> <tab> <chord>, with floating-point start and end times, and chord in the format <Note>:<tonality>. An example, trimmed to 4 decimal places: 1.8015 3.5296 A:min. There is an additional label, N, which indicates no chord is present for the duration. We convert the chord label representation to an array of chord labels at the sampling rate of the chromagram, for example [N, N, N, A:min, ..., N]. This is then one-hot encoded into 25 categories.

[1]https://magenta.tensorflow.org/

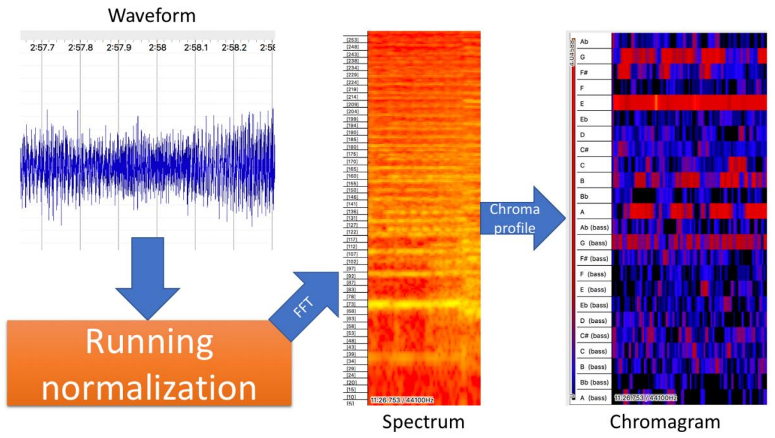

Figure 2: Conversion of raw audio samples into a chromagram

4 Methods

4.1 Preprocessing

Each of the 890 songs in McGill Billboard dataset was sliced into frames of increment 46ms (corresponding to 2048 points under 44.1kHz sampling rate). The chroma vector at each frame is 24-dimensional. Therefore, the chromagram $X^{(i)}$ of song i is a $n^{(i)}$-by-25 matrix, where $n^{(i)} \times 46ms$ is length of the song. We also fetched the ground truth label for every frame from .lab discriptive format and formed a $n^{(i)}$-dimensional vector $y^{(i)}$. Labels are given in integer indices, while the two-way mapping between chord names and their indices are also given. Since the length of different songs varies in a wide range (min=2458, max=15122, mean=4656), it is not efficient to zero-pad all songs to uniform length. Thus, we created a python list to store all chromagrams and chord labels.

4.2 Discriminator

The discriminator D reconstructs chords from chromagrams by outputting a K-dimensional vector of logits $\{l_1, ..., l_K\}$ with a final softmax layer to represent chord prediction. An extra class $K+1$ is appended at the end to represent the prediction of whether the input data was real or generated music.

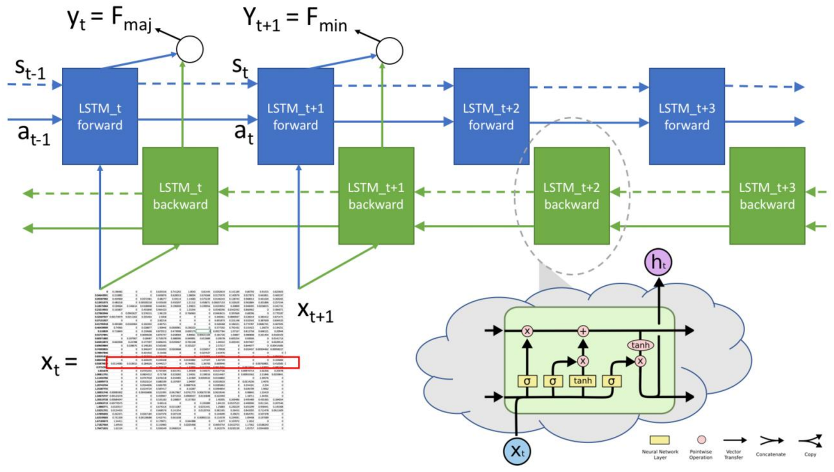

Figure 3: Single layer bidirectional LSTM. We stack two layers for the discriminator.

As the chord of a moment is defined by both preceding, current, and successive notes, we used bidirectional layers to capture the forward and backward temporal dependencies between notes. Thus, D is a RNN consisting of 2 stacked bidirectional LSTM layers (figure 3).

We use cross entropy loss that takes into account both supervised and unsupervised loss, where supervised loss comes from erroneously classifying labeled data in classes $1..K$, and unsupervised loss from classifying generated data as real.

$$L_{\text{unsupervised}} = -\mathbb{E}_{\boldsymbol{x},y\sim p_{\text{data}}(\boldsymbol{x})} \log[1 - p_{\text{model}}(y = K+1|\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim G} \log[p_{\text{model}}(y = K+1|\boldsymbol{x})] \tag{1}$$

$$L_{\text{supervised}} = -\mathbb{E}_{\boldsymbol{x},y\sim p_{\text{data}}(\boldsymbol{x})} \log p_{\text{model}}(y|\boldsymbol{x}, y < K+1) \tag{2}$$

$$L_{\text{discriminator}} = L_{\text{supervised}} + L_{\text{unsupervised}} \tag{3}$$

4.3 Generator Pipeline

To train both the generator and the discriminator is beyond the scale of this study. Thus, we utilized a well trained music generator model Magenta developed by Google Brain[2]. However, Magenta outputs MIDI files (containing only notes and time), and bridging between Magenta and McGill Billboard formats through code is by no means a trivial task. Our protocol was to convert MIDI files to soundtracks by Fluidsynth[3] with Soundfont[4]. Consequent soundtracks were later fed through Vamp[5] with plugins Sonic-Annotator[6] and NNLS-Chroma[7] in order to attain chromagrams produced with the same configuration as described in [4].

5 Experiments/Results/Discussion

5.1 Results

Figure 4: Training loss of the first 39 epochs. The training time of one epoch is exceedingly long due to large song sequences.

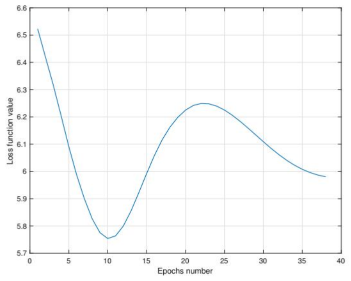

Table 1: Hyperparameters and results.

Batch size	89
Learning rate	0.005
LSTM hidden units	100
Loss value	5.76
Overlap ratio	4.9%

[2]https://magenta.tensorflow.org/

[3]http://www.fluidsynth.org

[4]http://freepats.zenvoid.org/sf2/sfapp21.pdf

[5]https://www.vamp-plugins.org/index.html

[6]https://www.vamp-plugins.org/sonic-annotator/

[7]http://www.isophonics.net/nnls-chroma

We trained on 89 labeled songs with an initial learning rate of 0.005, using the AdamOptimizer to minimize binary cross entropy. The result loss values are shown in 3. The dip in the beginning of Figure 4, we hypothesize, is due to the learning rate being too large, overshooting the local minima, resulting in an increase in loss.

We additionally experimented with GRUs and forward-only RNNs (to improve training time), however we did not see significant improvements from either modification, in training time or accuracy.

Our evaluation metric is the overlap ratio, as is the industry standard [2] and used in MIREX audio chord estimation competitions. The overlap ratio is computed as a ratio between the duration of correct chord annotations and the total duration of annotated segments. We have not reached convergence, and therefore the overlap ratio is 4.9%, compared to 80% in RNN implementations in recent results [2]. Note that the expected overlap ratio of random uniform classification for 25 chords is less than 4%, because the prior input distribution is non-uniform (there are biases towards certain more popular chords). Thus, a performance of 5% indicates that our network does learn to predict chords.

It was later realized during code clean-up that there was an error in Discriminator loss value implementation. We apply softmax across all logits when calculating loss, but the softmax should really be applied only to the first K classes that represented chord predictions. The last class, $K+1$, should be a binary value of 1 or 0. Applying softmax artificially enforces the value of $K+1$ to be less than 1, given that we want non-zero values for $\{l_1, ..., l_K\}$. This mistake was realized too late for further training, but could be a future improvement.

6 Conclusion/Future Work

Despite the implementation mistake, we see an improvement in loss value and overlap ratio. This shows that the network does learn to predict the output distributions of chords. We have confidence that given more time and computation resources to train, especially in a generative adversarial framework, we could see further improvements. In addition, we would like to explore different discriminator architectures and improvements beyond bidirectional LSTM, such as utilizing GRUs with the working model, beam search for generation, and attention models for identifying important chromagrams for chord prediction.

6.1 Discussion on Strategic Improvement

The result shows the importance of good strategic planning at the early stage of the project. We should have better assessed the difficulties and workload; tuning GANs is a very time-consuming process. Therefore, the better way would be to start with some very simple models and rough results, and add more components and features one after another. For example, we could instead start with a simple fully-connected model, then switch to a forward LSTM, then to bidirectional LSTM or perhaps a different type of RNN. This would have helped us catch our error sooner, due to the quicker training time of non-recurrent networks. For the generator part, we could output random noise at the beginning to incorporate into the system, and then figure out how to fit Magenta in the pipeline. We spent a considerable amount of time fine-tuning our individual parts, but did not fully realize the challenge of assembling the whole system. We should learn from this project, and keep Prof. Ng's iteration circle of idea->code->experiments in mind.

7 Contributions

Charles Chen	Generator, dataset preprocessing
Elizabeth Chu	Discriminator
Corey McNeish	AWS manager, evaluation metrics

Table 2: All unlisted workload (papers/poster) was shared equally.

References

[1] John Ashley Burgoyne, Jonathan Wild, and Ichiro Fujinaga. An expert ground truth set for audio chord recognition and music analysis. In Anssi Klapuri and Colby Leider, editors, *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 633–638, Miami, FL, 2011.

[2] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Audio chord recognition with recurrent neural networks. In *ISMIR*, pages 335–340. Citeseer, 2013.

[3] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[4] John Ashley Burgoyne, Jonathan Wild, and Ichiro Fujinaga. An expert ground truth set for audio chord recognition and music analysis. In *ISMIR*, volume 11, pages 633–638, 2011.