CS230

Harmonizing with Piano Genie

Craig Chan, Divya Gupta, Matthew Stein
Stanford University
{ckc1, dg524, klezmer}@stanford.edu

Abstract

As an extension of a recent Magenta[1] project called Piano Genie, we have trained a two-layer LSTM RNN which generates polyphonic musical improvisation over a chord progression based on real-time user input. This gestural interpretation, a contextual mapping from a limited 8-button input device to a full 88-key piano, makes virtuosic classical-style improvisation over chords accessible to non-musicians, akin to a performative Guitar Hero.[2]

1 Introduction

Magenta is an open source, TensorFlow-powered Python library for creative deep learning applications with a focus on music and image generation. Under Google Brain, the Magenta team regularly releases open source implementations of many useful models ranging from generative music improvisation[3] to collaborative drawing[4].

Although most people enjoy listening to music, few possess the knowledge of music theory or proficiency in an instrument to create their own. Piano Genie[5], a recent Magenta project, seeks to empower non-musicians to compose music by creating a simple interface which translates high-level musical gestures provided by the user to real piano music. A user improvises a sequence on an 8-button input device which is then decoded into realistic 88-key piano music in real time. The current implementation uses an RNN autoencoder to learn a mapping from button contour to a full size piano sequence using pitch and tempo features from MIDI piano performances [2]. This yields convincing melodic contours but lacks cohesive musical structure. We aim to extend the capabilities of Piano Genie by extracting additional musical features in the form of chord roots and qualities in order to produce performances with more musical structure.

2 Related work

Many mathematical approaches to musical composition and expression have been explored in existing literature. Music can be represented as a sequence of events. Predictive models can therefore compute conditional probabilities between musical events. In [5], hidden Markov models (HMMs) were used to generate chord progressions to accompany a melody using a chord transition probability matrix. RNNs became a popular approach to model musical sequences as they enabled incorporating long-term dependencies which could capture musical structure like A-B-A patterns or melodic motifs. Limitations of RNNs due to vanishing gradients were solved by long short-term memory (LSTM) units, as done by Eck et al. [3] to learn jazz melodies and chord progressions. RNNs have been

[1] https://magenta.tensorflow.org/
[2] Our Github repo: https://github.com/rachthree/magenta_cs230_win19
[3] https://github.com/tensorflow/magenta/tree/master/magenta/models/improv_rnn
[4] https://magic-sketchpad.glitch.me
[5] https://magenta.tensorflow.org/pianogenie

CS230: Deep Learning, Winter 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

used in modeling a wide range of musical tasks, from automatic composition, as done by Magenta's MelodyRNN and PerformanceRNN, to composition based on underlying chord progression, as in Magenta's Improv RNN, to chord accompaniment generation as done by Zhou and Meng [6].

Choi et al. utilized a character-based LSTM RNN for generating melody given a chord progression [1]. Chord root and quality were split up, so that a C minor chord is denoted as 'C' 'm' rather than "Cm". Using character-based rather than word-based RNN greatly reduced the dictionary needed to train their model, allowing two separate dictionaries combined with $n_{root} + n_{quality}$ words to be used rather than $n_{root}n_{quality}$ words.

3 Dataset and Processing

3.1 MAESTRO Dataset

The MAESTRO dataset [4] is comprised of over 172 hours of electronic piano music collected from live, competition-level performances of 17th through 20th century classical music, documented as pairs of aligned audio recordings and MIDI files. MIDI (Musical Instrument Digital Interface) is the industry standard musical communications protocol, used for the performance and recording of music on digital instruments, sheet music generation, and other applications. Pitch information is discretely represented by an integer in range [0, 127] (e.g. an 88-key piano ranging from a low A (A0) to a high C (C8) can be represented by MIDI range [21, 108].) In addition to pitch, each note's event message also includes key strike velocity (indicating volume) and a time delta ΔT indicating how many ticks[6] have passed since the last event, or how much time has passed since the last note was played.

In the Magenta framework, a fragment of music is represented as a NoteSequence protocol buffer (snippet included below). Polyphonic sequences, where multiple notes are played at the same time, are flattened to individual NoteSequence.Note messages and sorted in ascending pitch and time order.

```
message NoteSequence {
  string filename = 1;
  repeated Note notes = 2;
  repeated TextAnnotation text_annotations = 3;

  message Note {
    int32 pitch = 1;
    int32 velocity = 2;
    double start_time = 3;
    double end_time = 4;
    int32 chord_root = 5;  // Newly added
    int32 chord_quality = 6;  // Newly added
  }
}
```

The MAESTRO dataset authors propose an 80/10/10 train/validation/test split, ensuring that the same piece, even if performed by different pianists, does not appear in multiple subsets. Due to the implicit filtering out of many pieces which could not be annotated with chords, we used a reduced train set of 235 performances and test set of 50 performances, still maintaining a 80/20 train/test split. We employed two methods of data augmentation: extracting random subsequences from each performance and stretching the overall timing of each piece by a factor of 0.95 to 1.05 (close enough to the original tempo that the piece is still recognizable).

3.2 Chord Annotation Algorithm

Chord annotation of classical music is very difficult, and to properly automate it is an ongoing effort, including deep learning techniques such as Zhou and Meng's [6]. Entire music theory courses center around harmonic analysis, and the density, placement, and naming of chord labels is largely subjective as one strives to emphasize the form or cadential structures of a complex piece of music. Unlike jazz, in which "lead sheets" contain fixed chord progressions that typically repeat both through a melodic "head" and extended improvisation sections, classical sheet music rarely include such chord annotations. However, lead sheets typically only include simple, unornamented melodies, merely

[6]Ticks are the smallest unit of time in MIDI, and a beats (i.e. quarter notes) are comprised of many ticks.

the starting point for what a musician would actually play. We did not have access to a dataset of polyphonic jazz solos, so using human performances of classical piano was our best option.

Since the focus of our project was generating melody over chords and not the chordal analysis itself, we decided early on that a simple, programmatic approach would suffice. Our approach simply looks at all the notes played at a given time and determines what chord is most likely to fit those notes, using an existing Magenta utility chord_symbols_lib.pitches_to_chord_symbol(). Only major, minor, augmented, or diminished chords are considered "valid", and more complex chords or unknown chords are discarded, instead using the last seen valid chord for the notes played at that time. We introduced two tunable parameters to control accuracy: min_notes_per_chord and max_repeated_chords. The parameter min_notes_per_chord is how many notes must be played simultaneously to try to infer a chord, and we kept this at 3 or 4, the minimum number of notes generally necessary to determine both root and quality. The parameter max_repeated_chords is how many consecutive times an invalid or unknown chord can take on the last seen valid chord, and we experimented with values between 20 and 60, ultimately settling around 50, which led to 25% of the corpus being successfully annotated. While this parameter value may seem high, most pieces included at least one lengthy run of fast, non-chord notes which would otherwise cause the overall chord analysis for that piece to fail.

Whereas the previous Magenta convention was to include chord information as timestamped TextAnnotation messages, it was more efficient for us to attach this information directly to each individual note. If a note persists through multiple chord changes, it is just annotated with the first chord in which its included. Chords are represented as a root pitch (C, C#, D, Eb, E, F, F#, G, G#, A, Bb, and B, represented by [0, 11]) and a chord quality (major = 0, minor = 1, augmented = 2, and diminished = 3.) For instance, a C minor chord is represented as (0, 1).

4 Methods

Our network architecture is closely based on the baseline Piano Genie model in order to perform a fair comparison between the two models. The Piano Genie uses an autoencoder to perform an unsupervised learning task to infer 88-key melodies from 8-button input sequences, for which ground truth pairings do not exist. The autoencoder is composed of two LSTMs: a bidirectional LSTM encoder learns the mapping of 88-key piano sequences to 8-button sequences using integer quantization autoencoding (IQAE), and a unidirectional LSTM decoder learns to map the button contours back to note sequence melodies. Once trained, only the decoder is needed to convert user button inputs to melody [2].

Both the encoder and decoder are LSTM RNNs, each with 128 units. LSTM RNNs were well-suited to this task as they can capture the long-term dependencies of key and button contours. For example, repeated button presses could be inferred as repeated notes, or if the repeated button is the highest button on the device and they come after an ascending sequence, it may call for continuing the ascending sequence in piano keys. LSTM networks can utilize sequence history to make different predictions in these two cases. The size of the RNNs is intentionally kept small to allow for lower latency at inference time and the decoder is a unidirectional RNN so it can be evaluated in real time.

Below are the loss functions for the model [2]:

$$
\begin{aligned}
L &= L_{recons} + L_{margin} + L_{contour} \qquad (1)\\
L_{recons} &= -\Sigma log(P_{dec}(x|enc(x))\\
L_{margin} &= \Sigma max(|enc_s(x)| - 1, 0)^2\\
L_{contour} &= \Sigma max(1 - \Delta x \Delta enc_s(x), 0)^2
\end{aligned}
$$

Above, the encoder produces $L_{contour}$ and L_{margin}, and the decoder produces L_{recons} (reconstruction). L_{recons} is the reconstruction loss of the decoder, calculating by taking the average negative log likelihood of the model obtaining the correct note sequence given its encoding. L_{margin} discourages the encoder from producing values outside an interval in order to align with the IQAE discretization strategy. $L_{contour}$ is a musically-inspired regularization term to align the direction of key and button contours. This loss term incentivizes the sign of the note interval to match the sign of the button interval, i.e. match ascending note sequences with ascending button sequences, and vice versa.

In addition to note pitch and delta time features, our modified model also takes in chord progression features in order to hopefully produce melodies with more musical structure than the baseline model. The chord features for both encoder and decoder are expected to further constrain the outputted melody, making it more musically sensible, i.e. less random. The aforementioned chord annotation method produces chord progression information for each note, represented by the $x_{chordroot}$ and $x_{chordquality}$, which are one-hot encoded vectors. The new feature vector for our model is then:

$$x_{feature} = [x_{pitch}, x_{\Delta T}, x_{chordroot}, x_{chordquality}] \tag{2}$$

In application, the user inputs a desired chord progression and contour, and the model provides melodies with musical structure. Figure 1 shows the block diagram of our modified Piano Genie model.

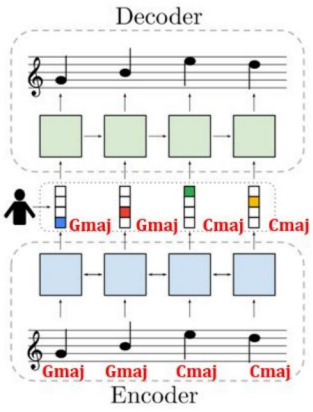

Figure 1: Modified Piano Genie Block Diagram with Chord Progressions.

5 Training and Results

5.1 Training

We maintained many of the same hyperparameter values from the baseline Piano Genie in order to do a fair evaluation of the effects of adding chord progressions to the input features. The mini-batch size was 32 examples, note sequence length was 128 notes, and an Adam optimizer was used with a learning rate of 0.0003.

The model was trained end-to-end to minimize the loss functions described above. The original baseline in the Piano Genie paper was trained with early stopping at about 166k steps with the complete MAESTRO dataset. Our chord annotation method produced a smaller subset of this data, decorated with chord annotations, which we used for both our baseline model (which ignored the chord annotations) and modified model. Since the time frame for the project was limited and the dataset was smaller, we decided to train our modified model for 126k steps. To avoid data mismatch, we needed to retrain the baseline model as well for 126k steps. We hypothesized that training both models for the same amount of steps would give a fair comparison between the two models. This way, we would directly see the effect of adding the chordal inputs.

5.2 Results

We evaluate our models using two metrics: perplexity (PPL) and contour violation ratio (CVR). Perplexity is a standard sequence model cross-entropy measurement between the original sequence and the model's prediction for that sequence. The calculation is given by $PPL = e^{L_{recons}}$, where L_{recons} is reconstruction loss of the decoder (see Equation 1). A lower perplexity score indicates better performance.

Contour violation ratio is a musically-based metric measuring the proportion of contour violations in key-to-button mapping, i.e. the fraction of time steps where the sign of the note interval does not

match the sign of the button interval. For example, if an ascending sequence of two notes is mapped to a descending sequence of two buttons, then this is considered a contour violation. A low contour violation ratio is preferred.

Dataset	Model	PPL	CVR
Train	Baseline	2.445	2.4603E-03
	Modified	2.64	3.4449E-03
Test	Baseline	3.216	1.2303E-03
	Modified	3.385	4.9213E-04

Figure 2: Performance results for baseline and modified models on train and test sets using perplexity and contour violation evaluation metrics.

Our overall hypothesis was that the modified model, which took in chord progressions as additional input features, would produce melodies with more musical structure than the naive implementation. Therefore, we expected that the perplexity of the modified model would be lower, as the decoder could produce better reconstructions of the original melody given chord structure, leading to a lower reconstruction loss. We anticipated that contour violation ratio would remain relatively the same as chord information should have little effect on the direction of button contours.

Our results were quite different from these expectations. On our test set, perplexity went up slightly from 3.216 on the baseline model to 3.385 on the modified model, a roughly 5% increase. The contour violation ratio decreased from 1.2303E-3 to 4.9213E-4, a difference of less than one one-thousandth. We compared the differences in perplexity and contour violation ratio scores of our two models to the scores achieved by the different models in the original Piano Genie paper to help frame whether the differences between the baseline and modified model were statistically significant. Contrary to what we expected, there was not a significant difference in contour violation and perplexity scores between our baseline and modified models.

Our perplexity results for both the baseline and modified model were higher on the test set than they were on the training set, which could suggest a variance problem and overfitting. On the other hand, contour violation results were actually slightly better on the test set than train set. We attempted to mitigate overfitting via several techniques, such as reducing architectural complexity by using a small two-layer LSTM RNN network and using data augmentation techniques, such as tempo augmentation and random subsequence crops.

6 Conclusion

At the start of this project, we hypothesized that adding chords to the Piano Genie model would decrease perplexity and have neutral effect on contour violation. However, our results indicate that there was not a significant difference in performance between the baseline and modified model, although contour violation slightly decreased and perplexity slightly increased.

Future work to improve our results includes the following approaches. More sophisticated ways to annotate chords that can result in a complete and sensible dataset would help gain more confidence in the results of the modified model. Some possible methods are to use hand annotation, an inverted ImprovRNN, other machine learning and deep learning models, or signal processing techniques. It is quite possible that since there are additional input features to learn from, the modified model would have to be trained longer than the original Piano Genie to see the full effects of implementing chord progressions. Additional data augmentation can be done by transposing each each piece into every key, scaling rhythmic durations by a constant, or introducing multiple granularities for beat annotations. Using a larger dataset and data augmentation could help reduce possible variance issues we saw in perplexity metric results.

Lastly, we intend to extend the original Piano Genie web demo[7] to use our chordal version of the model, allow users to input a chord progression before playing or pick from a preset one, and hear and visualize both the chords and the model's generated melody in real time.

[7] `https://piano-genie.glitch.me/`

7 Contributions

Craig, Divya, and Matthew came together from a shared background in and a passion for music and a desire to create either a useful tool to aid musicians or a meaningful way for non-musicians to generatively compose music. All three equally contributed to the project, and the below highlights their contributions.

Craig was responsible for the initial dive into the Piano Genie code to investigate how it was trained, the model architecture, the encoding and decoding variations, and how it was evaluated. He made modifications to the model to accept Matthew's new chord data in the input features, updated the configuration script to reflect the original Piano Genie and modified model, and trained and tested the modified model.

Divya contributed to understanding the model architecture and evaluation. She was responsible for path-finding the training pipeline on AWS EC2 p2.xlarge instances as well as training and testing the baseline model.

Matthew facilitated the initial connection with the Magenta team and met with Ian and Kevin on the Mountain View Google campus to discuss project ideas that could fit into the scope of this course and also meaningfully contribute to ongoing Magenta projects. He was responsible for chord annotation, the data conversion pipeline, and the loading of the new chord data in the model.

8 Acknowledgements

Thank you to Ian Simon and Kevin Malta on the Google Magenta team for meeting with us multiple times, both digitally and in person, to discuss music-oriented project ideas and orient us in the field of cutting-edge musical HCI and for answering countless questions over email; to Chris Donahue, who created Piano Genie while an intern at Google last summer, for answering questions about past design choices and model implementation; and to our TA mentor Hojat Ghorbani for meeting with us regularly and helping us keep on track.

References

[1] Keunwoo Choi, George Fazekas, and Mark Sandler. Text-based lstm networks for automatic music composition. 2016. arXiv:1604.0535.

[2] Chris Donahue, Ian Simon, and Sander Dieleman. Piano genie. 2018. arXiv:1810.05246v1.

[3] Douglas Eck and J. Schmidhube. "finding temporal structure in music: blues improvisation with lstm recurrent networks. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pages 747–756, Martigny, Switzerland, 2002. IEEE.

[4] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the maestro dataset. 2018. arXiv preprint arXiv:1810.12247.

[5] Ian Simon, Dan Morris, and Sumit Basu. Mysong: automatic accompaniment generation for vocal melodies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, page 725–734, Florence, Italy, 2008. ACM.

[6] Yulou Zhou and Shuxin Meng. Automatic chord arrangement from melodies. 2018. CS230, Stanford University.