Automatic Music Transcription: Generating MIDI From Audio

Aitan Grossman (aitan@stanford.edu), Josh Grossman (jdgg@stanford.edu)

June 11, 2020

Abstract

Automatic Music Transcription (AMT) is the process of computing the symbolic musical representation of a musical audio recording. Out of the existing AMT solutions, none have reached the human-level accuracy on this task of nearly 100%. Although the problem formulation is simple, it requires a complex model to accurately detect individual notes from a noisy signal. We propose an approach that transforms this task to sequential image classification and leverages deep learning's superior ability to learn the structure within images and across sequences. Specifically, we constructed a bidirectional LSTM network with convolutional and pooling layers in an attempt to symbolically classify audio representations of classical piano music. Although we were not able to achieve passable performance, we believe our methods are basically sound.

Introduction

Music transcription is a task with great potential to benefit from current methods in deep learning; music transcription is formulaic, mundane, and crucial in professional music. An accurate AMT tool would be able to take as input a session of acoustic piano playing and quickly convert it into a MIDI file, as if the piano were a MIDI instrument. MIDI is a communications protocol used in recording and synthesizing music, capable of storing parameters such as notation, pitch, and clock signals for tempo. This MIDI file can easily be converted into a transcribed score, can easily be shared with other musicians for collaboration, and can be edited in order to change individual notes from the performance to improve a composition.

Related Work

Until recently, accurate AMT was nearly impossible. The first version of Anthemscore, a costly tool for AMT, was released in 2015. In 2018, Google's Project Magenta trained and open-sourced an automatic music transcription tool with fairly impressive accuracy using a neural network consisting of convolutional layers, fully connected layers, and bidirectional LSTMs. The tool achieves an F1 score close to 0.9 using a reasonable custom evaluation metric. The tool works well on simple music and is more prone to err on music containing highly clustered chords. The goal of our project is to use Keras build a network similar to that of Google's Project Magenta, and ideally obtain similar performance.

Data Set and Features

We used the MAESTRO dataset[1], which contains over 200 hours of classical piano performance from 2010 to 2018. The full dataset contains MIDI representations of the piano performances aligned temporally within 3ms of the recorded audio. The uncompressed audio is 200GB of 44.1–48 kHz 16-bit PCM stereo. To speed training, we used the MAESTRO data from 2018, which contained waveforms and MIDI representations for 99 pieces of piano music.

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

Methods

For the remainder of the paper, we discuss how deep learning can offer an improvement over the baseline through the use of a recurrent CNN. We provide an overview of our approach including pre-processing, neural network architecture, and evaluation.

Designing A Neural Network

Learning MIDI directly from WAV data is extremely difficult, because MIDI is optimized for ingestion by computer programs and WAV is highly unstructured—just a single piano note in WAV format would create a nearly incomprehensible sequence of ones and zeros. However, we can formulate the problem as image classification with some clever data processing. Specifically, by performing signal processing on the WAV data and using a Python library to process the MIDI data, we can produce a two-dimensional spectrogram which displays the power at different frequency buckets over time as well as a corresponding two-dimensional set of labels indicating which notes are "active" over time.

CNNs are a class of deep neural networks that are designed to perform particularly well in image classification. Convolutional layers apply filters to an input to detect the presence of specific features or patterns present in the original image, while pooling layers reduce the number of parameters in the network. By formatting our spectrograms and labels as training pairs with fixed dimension, we can perform image classification using a CNN. This approach has been employed by other developers tackling the challenge of AMT.

Rather than treat the transcription process as a series of independent image classification problems, we chose to use a recurrent CNN based on the insight that musical information is highly contextual. In the presence of a noisy or complicated audio signal, an experienced human listener will base their perception of a note on information from the few seconds before and after the note in question is played. As such, we designed a model which utilizes bidirectional long short-term memory (LSTM) cells to incorporate previous and future hidden activations in the learning process.

Data Processing

We designed a data pipeline to process the MAESTRO WAV and MIDI data in an efficient and flexible manner. We used the `librosa` library to convert WAV files to spectrograms using the constant Q-transform (CQT) (Figure 1), and we used the `pretty_midi` library to convert MIDI files to pianorolls (Figure 2). We leveraged the full processing power of our virtual machine's 4 CPU cores using Python's built-in `multiprocessing` library. This allowed us to aysynchronously convert all of the 2018 MAESTRO data within an hour. The `numpy` array representations of the spectrograms and pianorolls were efficiently stored as `hdf5` files for further processing. We note that processed data used approximately 8 times as much disk space as the raw data.

After processing the data into `.h5` files, there are several additional steps before we can feed the data into our model. The 2018 data occupy roughly 200 GB (compared to the 64 GB of RAM available to our p2.xlarge AWS instance), so they cannot be fed directly into our model. To enable training on all of the data, we load one piece at a time into memory using a custom generator function, and for each piece, we process it into batches of shape x : (32, 360, 288, 5, 1) and y : (32, 360, 88). This means each batch contains 32 sequences, each sequence contains 360 consecutive slices, and each slice has dimensions $288 \times 5 \times 1$ for the spectrogram and 88 for the 88 keys of a piano (the 1 specifies the number of "color" channels for our first convolutional layer). The model optimizer then requests these batches one at a time as needed.

Training

Once the data generation process was defined, we trained a recurrent CNN according to the model outlined in Figure 3. We used kernel sizes of (20, 2) and pool sizes of (4, 2), LSTM hidden layer sizes of 500 and 200, a recurrent dropout probability of 0.75 on the first LSTM layer, and a sigmoid activation function to generate the $\hat{y}$ for each time step. For training, we used a fixed sequence length of 360 input pairs representing approximately 5 seconds of music, which we sampled every 720 slices in order to speed up training over the large dataset. We used a fixed batch size of 32 sequences per batch, and with a 0.5/0.25/0.25 train/dev/test split, we were able to train on 135 batches per epoch for a total of 4320 5-second sequences of music.

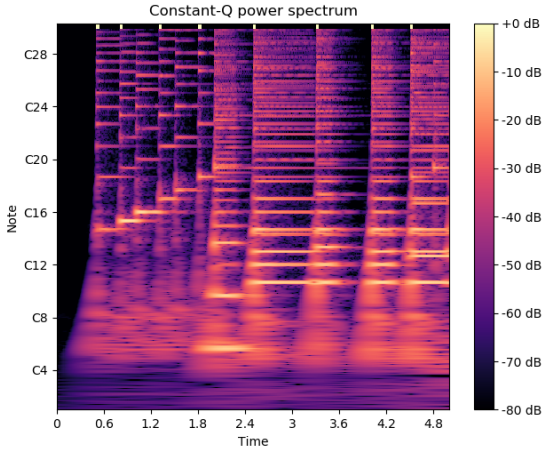

Figure 1: Visualization of a spectrogram using Constant-Q Transform (CQT)

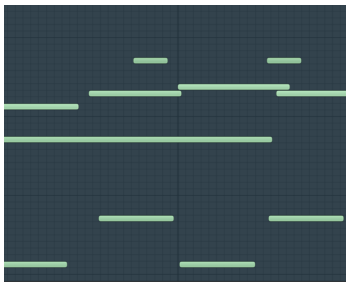

Figure 2: Visualization of pianoroll

We used an Adam optimizer and binary crossentropy loss, and experimented with a variety of different hyperparameters, described in Discussion.

Results

Accuracy Evaluation

To quantify the accuracy of the model on our test data, we prepared the test data in a similar fashion to our train data, the only differences being (1) that we used all slices, not just those following note onsets, and (2) that we prepared pianoroll for evaluation of the CNN output rather than for training of the CNN. With the aggregated output for a piece, we then used a custom evaluation function in an attempt to quantify precision and recall in agreement with the human ear.

The first step of our evaluation function is to reduce strings of consecutive 1s to a single 1 followed by 0s, thus preserving note onsets and ignoring note duration. We justify this because notes on a piano decay quickly, which makes note duration more useful theoretically than practically, and because our CNN is trained only on note onsets.

To calculate the precision of the output transcription of a piece, our evaluation function locates note onsets in the output pianoroll and searches for the nearest note onset in the golden pianoroll on the same key up to a distance of 0.6 seconds. It then reports as the loss for that note as follows:

$$loss = \frac{\min(|t - t'|, 0.6)^{0.8}}{0.6^{0.8}},$$

where t is the time in seconds of the output note and t' is the time in seconds of the nearest ground truth note found. The total loss is the sum of $loss$ for each note in the output pianoroll. This approach encodes the assumption that a note that is reproduced a little bit early or a little bit late is an inaccurate reproduction of that note, but the output of any other note is assumed not to be an attempt at the intended note, and it probably would not sound good at all if it were!

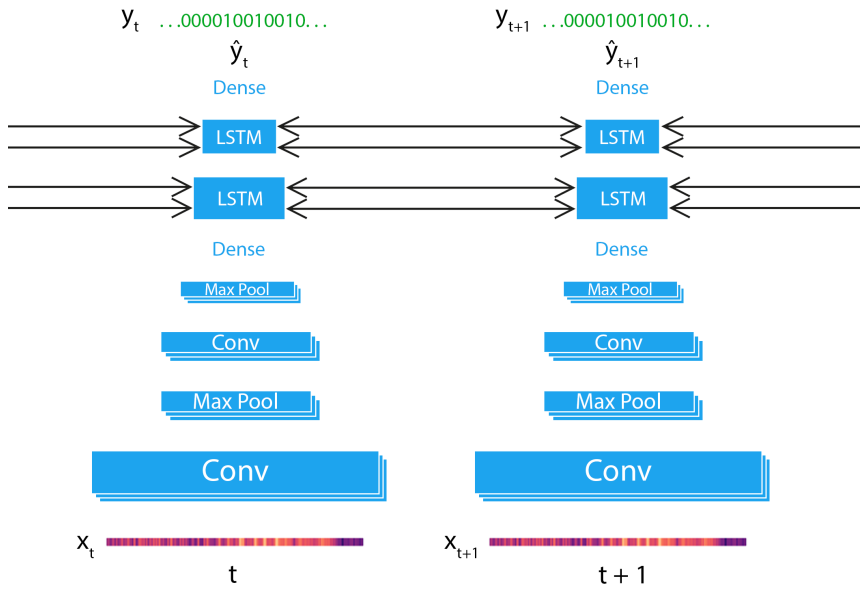

Figure 3: Diagram of our recurrent CNN model with bidirectional LSTM

To calculate recall, the output and golden pianorolls are swapped and the same function is used. We use the F1 score as our final measurement of accuracy:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}.$$

We also implemented a separate evaluation function which takes into account note duration as well as time of onset, where precision and recall are calculated by simply taking the intersection of the output and golden pianorolls and normalizing it by the L1 norm of output or golden, respectively.

Finally, we created a command for converting WAV to MIDI utilizing the `pretty_midi` library. This command loads a set of model weights and executes the full forward-propagation process to construct a MIDI file, which can be listened to on most personal computers. Although this evaluation approach does not offer any rigorous accuracy metrics, it is the ultimate test of the quality of any AMT tool.

Discussion

Despite our best efforts, we were not able to achieve reasonably good performance using a custom model. We consistently ran up against the issue of loss plateauing prematurely at approximately 94%, with outputs heavily biased toward zero. Despite experimenting with models of varying complexity and searching over multiple values for learning rate, decay, and batch size, we were not able to overcome the issue of too many zeros. This issue could be caused by a number of different problems with the training process—if no clear relationship between x and y can be learned, the model achieves the lowest loss by making the bias terms extremely negative, thus allowing the model to correctly guess the 78–88 necessarily inactive notes at any given time slice.

The most likely explanation for this issue is that our model was not built correctly for this specific task and thus was not able to learn parameters well enough given its depth to overcome vanishing gradients. While the relationship between x and y in this application is clear, and even a non-neural model might be able to achieve passable accuracy, the task of a neural model is to learn how to recognize a note's harmonic series so that the several peaks in power from the spectrogram are recognized as a single note. Whereas many image classification tasks involve discovering localized objects and patterns in each layer, this task involves discovering patterns spanning the entire input tensor. Due to our cursory understanding of CNN engineering, the model we built was likely unable to capture this complexity.

Otherwise, we are confident that our approaches to data processing, training, and evaluation are mostly sound, and we believe reengineering our deep model to more closely resemble the model

developed by Magenta would yield promising results.

Conclusion

We were not able to reach our ideal F1 accuracy score of 0.95, but we are optimistic that our methods are sound on a high level and offer new insight into the development of a practical AMT solution. We also hope the learnings from our experiment are a testament to the solvability and importance of the problem. We also suggest two primary directions for future development.

Next Steps

An essential step for development of our AMT tool in the short term is improving the model design. In addition to experimenting further with the convolutional part of our model, there are a couple higher level approaches we could take to catch up to Magenta's solution in terms of performance. First, we could simply use Magenta's model as a baseline. Magenta's work is open source, so we have the luxury of augmenting their model with aspects of our previous approach as well as some additional insights. For example, we theorize that a state-of-the-art AMT model would be able to "hear" chords, in addition to each individual note, with the help of more advanced feature extraction. We could also experiment with adding residual connections, providing a time step's $\hat{y}$ as input to its neighboring LSTM cells, and augmenting, or noisifying, the data to make the tool more robust to real-world input.

An ideal extension in the long term would be to extend the model to be able to transcribe ensembles. While this is a much harder task, we are confident it is possible. This would involve first recognizing which instruments are playing at any given time, identifying which notes belong to which instruments, and passing it through a CNN with knowledge of different instruments—ideally, a CNN trained only on that specific combination of instruments. Multi-channel audio signals would help greatly with this challenge as well, because it would greatly aid the matching of instruments to notes as well as the transcription of notes. However, most combinations of recording technology and acoustic surroundings yield audio that is too noisy or percussive for reliable human transcription, in which case we believe that reliable AMT is nearly impossible.

Contributions

Both authors designed the model through discussion and experimentation, and co-wrote the final report. Aitan Grossman implemented and trained the model. Josh Grossman designed and implemented the data pipeline.

References

AnthemScore. (n.d.). *Music transcription with convolutional neural networks.* Retrieved from https://www.lunaverus.com/cnn

Brown, J. C. (1991). Calculation of a constant q spectral transform. *The Journal of the Acoustical Society of America*, *89*(1), 425–434.

De Souza, J. (2014). Voice and instrument at the origins of music. *Current Musicology*(97), 21–36.

McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and music signal analysis in python. *Proceedings of the 14th python in science conference*, 18–25.

MuseScore. (2015, Jan). *Sheet music | musescore.* Retrieved from https://musescore.com/sheetmusic?text=welltemperedclavier

Raffel, C., & Ellis, D. P. (2014). Intuitive analysis, creation and manipulation of midi data with pretty_midi. *Proceedings of the 15th International Conference on Music Information Retrieval Late Breaking and Demo Papers*.

Román, M. A., Pertusa, A., & Calvo-Zaragoza, J. (n.d.). An end-to-end framework for audio-to-score music transcription on monophonic excerpts.

Verma, D. (2017, Dec). *Music transcription using a convolutional neural network.* Medium. Retrieved from https://medium.com/@dhruvverma/music-transcription-using-a-convolutional-neural-network-b115968829f4

Wang, Q., Zhou, R., & Yan, Y. (2018). Polyphonic piano transcription with a note-based music language model. *Applied Sciences*, *8*(3), 470.