Piano Genie - Generative Music Improvisation

https://github.com/pulpiction/CS1470-Final-Project-J5

Jaehyun Jeon, Junewoo Park, Min Jean Cho, Oh Joon Kwon
{jjeon5, jpark49, mcho5, ok1}@cs.brown.edu

Introduction.

Piano Genie (https://magenta.tensorflow.org/pianogenie) is a creative deep learning project for generating sequentially and temporally relevant music from human agent interaction. The model was initially designed and published by Chris Donahue, Ian Simon, and Sander Dieleman of Google Magenta Foundation [1]. We have modified and translated the model into Tensor Flow 2.x architecture with Keras framework.

Piano Genie is trained on professional piano performances derived from e-Piano compeitition dataset, which allows non-musicians to improvise on simple interface with only 8 buttons. The model is based on unsupervised learning (autoencoder) and a stacked RNN architecture; the encoder learns to map 88-key piano sequences to 8-button sequence, and the decoder learns to map the button sequences back to the piano sequences.

During the actual performance, the user's input replaces the encoder's output. Then the decoder evaluates the sequences real-time to output the piano note sampled from the logits of the decoder. The model also learns to output pitch contours that mimics the contours of user button sequence by minimizing the contour loss, giving an impression of actual piano performance.

Data.

Dataset used in the project was MAESTRO dataset, which is a collection of MIDI files from International Piano e-Competition [2]. This dataset provides virtuosic performances by professional pianists which can be easily converted to integer sequences. Train and test dataset were created by extracting a random 128-length note sequence from each of $\approx$ 760 files with two random sweeps. This provided us with around $\approx$ 1500 note sequences to train and test on.

Methodology.

The model uses LSTM cells as base architecture. The encoder uses doubly-stacked bidirectional LSTM cells while the decoder uses doubly-stacked unidirectional LSTM cells. This allows the model to learn the nature of note sequences back and forth in a simultaneous manner. This can help improve on the limitations of traditional RNN where future state cannot be directly reached from the present state.

Implementation-wise, the training process takes input tensor of dimension `(batch_size, 128, num_features)` where features are generated by concatenating one-hot representations of notes (depth 88) and Δt (gap between two consecutive notes, depth 33). This input is passed through the encoder then a dense layer of size 1 for dimensionality reduction. This reduced output is quantized by IQST as suggested in the paper.

Marginal and Contour losses are computed, and the decoder features are created. The decoder features include the quantized latent pitch, last pitches, which are the first 127 notes of each sequence padded with -1's, and the one hot representation of Δt. Once these features are passed through the decoder, the reconstruction loss is computed by taking the softmax crossentropy between the computed logits and the original piano sequences. The model then optimizes the total loss, which is given as the sum of the three losses.

$$L = L_R + L_M + L_C$$

$$\begin{aligned} L_R &= -\sum_{\mathbf{x}} \log \mathbb{P}_{\text{dec}}(\mathbf{x} \mid \text{enc}(\mathbf{x})) \\ L_M &= \sum_{\mathbf{x}} \max(|\text{enc}_s(\mathbf{x})| - 1, 0)^2 \\ L_C &= \sum_{\mathbf{x}} \max(1 - \Delta\mathbf{x} \cdot \Delta\text{enc}_s(\mathbf{x}), 0)^2 \end{aligned}$$

During evaluation (the actual performance), we only use the decoder to create a piano sequence. To create the decoder features, the button input is rescaled in the range of $[-1, 1]$. The one hot representations of the last output and the Δt are concatenated along with the rescaled button input. For the next input, we use the last state tensors for creating the initial state of the decoder. The model is expected to output the matching contours of the input, i.e. output increasing piano notes if increasing button sequence is given, and similarly for decreasing sequence.

Challenges.

There were several challenges along the way. First, dataset processing had some implementation issues. We have planned to use Magenta utilities for preprocessing the data, but we ran into major compatibility issues with Tensor Flow 1.x. We could bypass this problem by installing magenta, upgrading to Tensor Flow 2.x, then commenting some deprecated utilities directly out from the library itself.

After that, the note sequences had to be extracted from the MIDI files. However, dataset iterators from Tensor Flow 1.x and old TFRecord extensions proved to be too cumbersome and unnecessary for our purpose, so we had to directly extract sequences from MIDI files then pickle the dictionary of tensors for later use. This proved to be the most effective and efficient method of preprocessing.

Building the architecture was challenging as well. The encoder was built on bidirectional doubly-stacked LSTM cells and the decoder on unidirectional doubly-stacked LSTM cells. We had to spend some time understanding the architecture then implementing it in Tensor Flow Keras. We also had to spend some time debugging the model for dimension mismatch for both training and evaluation.

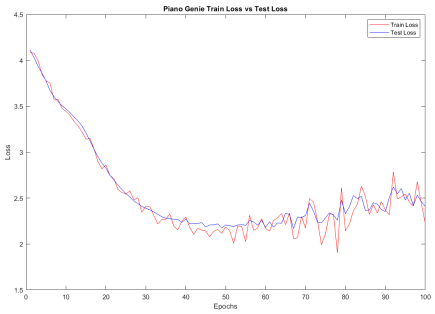
Furthermore, the model also had an intermediate step of quantization. No one in the team had prior knowledge of sound quantization techniques. Understanding the original code for quantization written in Tensor Flow 1.x was challenging with variable scope methods, but we eventually understood this part of the model and successfully implemented it. We had to ensure the quantization was done in `tf.function` for proper gradient computation.

The final challenge was creating the user interface. The original plan was to create an image segmentation interface where users can use their hands to "conduct" generative performance like theremin, but this proved to be too impractical. We resorted to a simple 8-button input on an iPhone app, which received computed tensor from our trained model via PyServer module. The computation of outputs was instant, but sampling MIDI sound fonts on the app had noticeable lag, which is planned to be resolved in the near future.

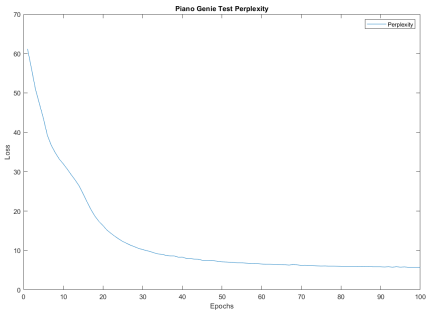
Results.

The train and test losses are given in the plot below. Since learning from musical notes can be seen as a generalization of language models, it is reasonable to use perplexity as a loss measure. Over a training session of 100 epochs, the test and train losses steadily decreased to

around 2.1. However, the decreasing loss did not guarantee the model to output reasonable-sounding sequences. In order to create a good model, observing perplexity was more helpful. The perplexity on the test dataset kept decreasing to around 6.74.

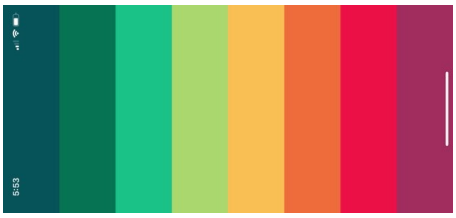


(a) Test and Train Losses over 100 Epochs.



(b) Test Perplexity over 100 Epochs.

Since this was a creative project, we had to consider the actual human perceptions of the output. Upon evaluating the trained model, the note sequences created by the model was actually "nice". Even simultaneous button sequences generated pleasant sounding chords in most cases, which was surprising. By training on some famous scores, the model seemed to have learned musical chords and sometimes outputted easily recognizable sequences from well-known works.



(a) App Demo Screenshot.

(b) Console Output Screenshot.

Reflection.

This project posed many challenges. At a theoretical level, we had to familiarize ourselves with RNN based autoencoders, which was an interesting and novel concept on its own. Implementation-wise, we had to learn how to preprocess MIDI data, create interface, and optimize the model for real-time evaluation. This was a good opportunity to learn about new deep learning architectures and how to deal with problems one may encounter implementing a model.

Some of the aspects we had hoped to implement in the initial proposal was the image segmentation for user interface for theremin-like performance. However, we were held back in schedule by debugging models and processing the dataset, which was unfortunate. The button user interface we have created for demo had some issues with MIDI sampling for sound fonts, which caused occasional input lags. However, the tensor computation itself was instant, so we were pleased to have successfully implemented the model at least.

One thing that bothered the team was that we were not able to achieve the perplexity and loss stated in the paper. This may have been due to some hyperparameter adjustments made by the authors. Taking the project further, we may want to tweak some hyperparameters for more stable training procedure, along with some optimizations for server communications and user interface. We plan to make some changes in the future as a continuation of this final project.

There have been numerous works on deep learning-based music generation architectures recently. However, most of them are computationally expensive and/or focused on generating sheet music, not real-time performance. Similar trends apply to visual arts as well, where visual GAN architectures have taken a prominent role. Many artists are fascinated by this new idea of machines generating "art", but also dismissing them as mixture of visual information without emotional values.

Therefore, we believe that there are more works to be done in the future on interaction-based real-time improvisational architecture that could aid artists in their creative processes. As a matter of fact, there have been some interesting works in visual art communities where artists collaborated with ML models for generating arts [3]. This feedback loop between human and machine agents can reinforce and facilitate creative processes, possibly providing entirely new meaning to arts.

Link to Github.

Link : https://github.com/pulpiction/CS1470-Final-Project-J5

References

[1] C. Donahue, I. Simon, and S. Dieleman, *Piano genie*, in Proceedings of the 24th International Conference on Intelligent User Interfaces, ACM, 2019, pp. 160–164.

[2] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, *Enabling factorized piano music modeling and generation with the MAESTRO dataset*, in International Conference on Learning Representations, 2019.

[3] J. Kim, *Ai art in korea "commune with ..."*. https://www.datadriveninvestor.com/2019/11/18/ai-art-in-korea-commune-with/, 2019. [Online; accessed 15-Dec-2019].