Machine Learning with Music Generation

Zhenye Na
Department of Industrial and Enterprise Systems Engineering
zna2@illinois.edu

***Abstract*—The goal of this project is to be able to build a generative model from scratch trying to create new musical melody. There exists some work in music generation area and they have already achieve some amazing results. More recent work is focusing on polyphonic music modeling, centered around time series probability density estimation. In this project, I tried using Generative Adversarial Network and Restricted Boltzmann Machines to create new music melody.**

I. Introduction

Music is the ultimate language for both communicating and relaxing. Many amazing composers throughout history created pieces of music. However, Is it possible for a computer to learn to create such music automatically through Machine Learning Algorithm?

The answer is definitely true. In fact, Taryn Southern recently released her new song which is generated by AI composer, which is really fantastic. Furthermore, inspired by Siraj Raval's amazing blog, I am trying to explore how to generate new music via Machine Learning Algorithms.

II. Background & Related work

There are several papers about both generating and classifying music. I have read some of them and below is part of them which I think is really amazing.

One of the most attractive paper applies Generative Adversarial Network, written by Hao Dong et al [1], generates multi-track, both melody and harmony.

Eck et al [4]. use two different LSTM networks - one to learn chord structure and local note structure and one to learn longer term dependencies in order to try to learn a melody and retain it throughout the piece.

III. Dataset

The main challenge is to select the appropriate dataset and how to represent it in order to feed to any Machine Learning algorithms. Here, I selected a midi files which contain single track, which is simplified to generate.

A. Midi files

Midi files are structured as a series of concurrent tracks, each containing a list of meta messages and messages. Midi files are very small, but does not lose quality. It does not contain any human voice - only sound tracks.

B. Data Representation

In this section, we review several audio data representations in the context of using deep learning methods. Majorities of deep learning approaches in MIR take advantage of 2-dimensional representations instead of the original 1-dimensional representation which is the (discrete) audio signal. In many cases, the two dimensions are frequency and time axes.

There are several representation of sound based on the tutorial in Keunwoo Choi et al [2] in Fig. 1

- **Audio signal**: The audio signal is often called raw audio, compared to other representations that are transformations based on it.
- **Short-Time Fourier Transform**: STFT provides a time-frequency representation with linearly-spaced center frequencies.
- **Mel-spectrogram**: Mel-spectrogram is a 2D representation that is optimized for human auditory perception
- **Constant-Q Transform (CQT)**: CQT provides a 2D representation with logarithmic-scale center frequencies.

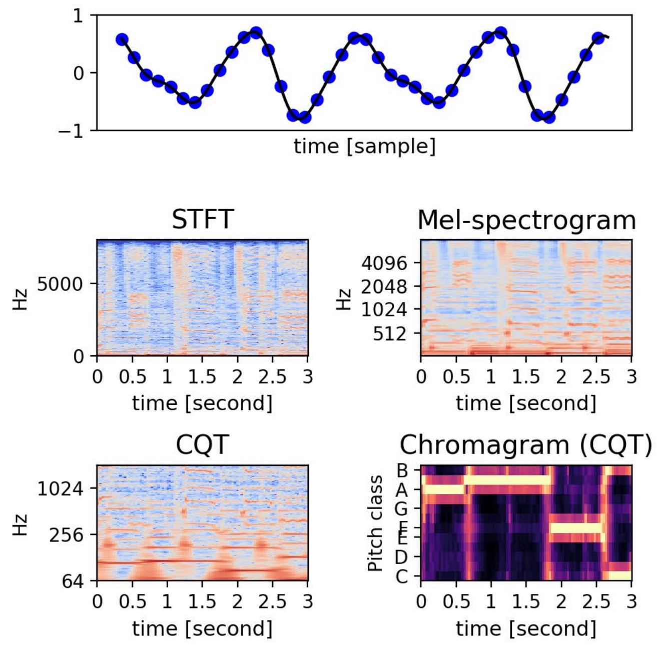

Fig. 1. Audio content representations. On the top, a digital audio signal is illustrated with its samples and its continuous waveform part. STFT, Mel-spectrogram, CQT, and a chroma-gram of a music signal are also plotted . Please note the different scales of frequency axes of STFT, Mel-spectrogram, and CQT.

C. Data Preprocessing

In order to feed MIDI files to Machine Learning algorithms, I manipulated MIDI files to a matrix format which is the most common data format. As in the Fig. 2, the y-axis is note-range and x-axis is the time steps (or duration). So I used 1 to represent the key has been pressed and 0 to represent not in a big matrix. Also, if we observe a little bit more, we will find there exists conditional probability here. The next keys in the next time-step are conditioned on the current time-step. I merely wrapped up the data manipulation function by Dan Shiebler and make it suitable in my project.

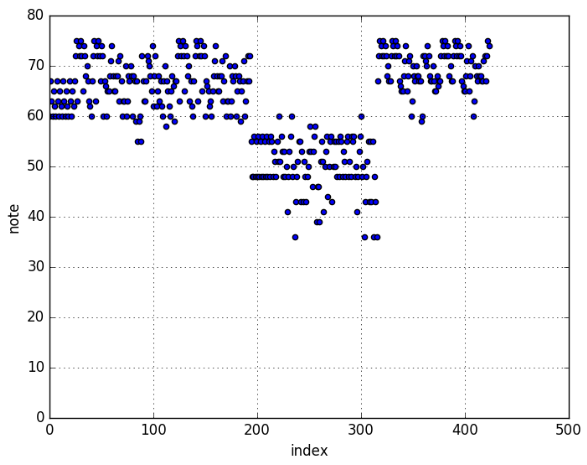

Fig. 2. Matrix representation of MIDI files

IV. Approach

In this section, I explored using two different approaches. One is successful and another failed. I will give some reasons why it does not work later.

A. Generative Models

Generative Models specify a probability distribution over a dataset of input vectors. For an unsupervised task, we form a model for $P(x)$, where x is an input vector. For a supervised task, we form a model for $P(x|y)$, where y is the label for x. Like discriminative models, most generative models can be used for classification tasks. To perform classification with a generative model, we leverage the fact that if we know $P(X|Y)$ and $P(Y)$, we can use Bayes rule to estimate $P(Y|X)$.

B. Generative Adversarial Network

The first approach is to apply GAN to midi files. The goal for this project is to generate new music. So any generative models will be appropriate for this task. However, the theory behind is a little bit different. For the generator of GAN, I use two stacked LSTM and dropout layer after each LSTM layer. Because the attribute of music is sequential, the best model for sequential data is RNN. However, basic RNN is suffering from Vanishing gradient point problems. I selected LSTM instead. At the end, I use fully-connected layer for the output layer. For discriminator, I applied two Convolution layers with LeakyReLU as activation function .

The loss function I used in cross-entropy. It is obvious that the generator (G) try its best to fool the discriminator (D), so that the loss function of G is to optimize (minimize) the distance of generated songs with label 1, which stands for real song. Meanwhile, D is trying its best to distinguish the two different categories.

The optimizer I used is RMSProp and Adam. I used Tensorflow as the Deep Learning library.

1) Dense layers
A dense layer is a basic module of DNNs. Dense layers have many other names - dense layer (because the connection is dense), fully-connected layers (because inputs and outputs are fully-connected), affine transform (because there is $Wx+b$), MLP (multi-layer perceptron which is a conventional name of a neural network), and confusingly and unlike in this paper, DNNs (deep neural networks, but to denote deep neural networks only with dense layers). A dense layer is formulated as follows.

$$y = f(\mathbf{W} \cdot x + b)$$

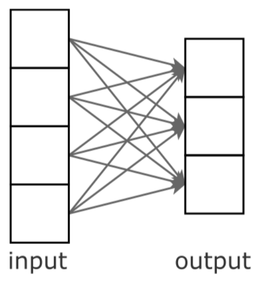

Fig. 3. An illustration of a dense layer that has a 4D input and 3D output.

2) Recurrent layers
A recurrent layer incorporates a recurrent connection and is formulated as follows

$$y_t = f_{out}(Vh_t)$$

$$h_t = f_h(\mathbf{U}x_t + \mathbf{W}h_{t-1})$$

where f_h is usually tanh or ReLU, f_{out} can be softmax/sigmoid/etc., ht: hidden vector of the network that stores the information at time t and **U**, **V**, **W** are matrices which are trainable weights of the recurrent layer. To distinguish the RNN with this formula from other variants of RNNs, it is often specied as vanilla RNN.

C. Restricted Boltzmann Machine

RBM is a neural network with 2 layers, the visible layer and the hidden layer. Each visible node is connected to each

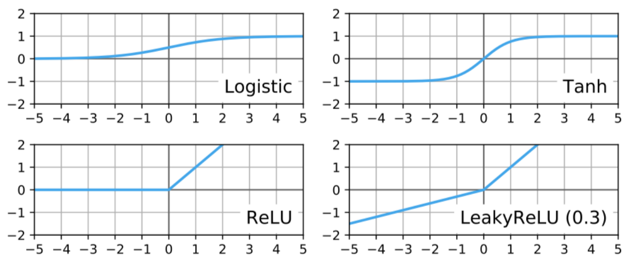

Fig. 4. Four popular activation functions - a logistic, hyper-tangential, Rectified Linear Unit (ReLU), and leaky ReLU.

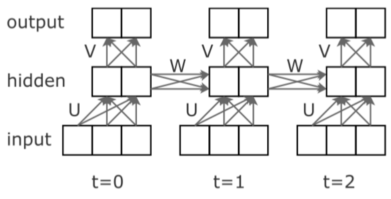

Fig. 5. Unfolded RNN with 3 time-steps

hidden node (and vice versa), but there are no visible-visible or hidden-hidden connections (the RBM is a complete bipartite graph). Since there are only 2 layers, we can fully describe a trained RBM with 3 parameters:

- The weight matrix W: size $n_{visible} \times n_{hidden}$. W_{ij} is the weight of the connection between visible node i and hidden node j.
- The bias vector bv: vector with $n_{visible}$ elements. Element i is the bias for the i_{th} visible node.
- The bias vector bh: vector with n_{hidden} elements. Element j is the bias for the j_{th} hidden node.

D. *Gibbs Sampling*

Gibbs sampling is a Markov Chain Monte Carlo (MCMC) algorithm for obtaining a sequence of observations which are approximated from a specified multivariate probability distribution, when direct sampling is difficult. This sequence can be used to approximate the joint distribution (e.g., to generate a histogram of the distribution); to approximate the marginal distribution of one of the variables, or some subset of the variables (for example, the unknown parameters or latent variables); or to compute an integral (such as the expected value of one of the variables).

Typically, some of the variables corresponding to observations whose values are known, and hence do not need to be sampled. It basically applies conditional probability again and again. So it is a good choice to use for sampling hidden state for this case.

V. Experiment

A. *Generative Adversarial Network*

The result given by GAN is not so good. 10 classical music has been divided into same length of music pieces and training in 200 epochs. The generated music is very noisy. I think there are several reasons this model does not work in my case:

- **Loss function**: The loss function I chose may be not appropriate for this problem.
- **Network Architecture**: architecture of Discriminator (D) used Convolution layers which is not the best for this problem for classifying.
- **Sampling method**: Uniform distribution which did not contain conditional probability.

B. *Restricted Boltzmann Machine*

In this model, I fixed sampling method via sing Gibbs Sampling which takes conditional probability into account. At this time, RBM catch all the feature point and generate some qualified music. However, based on the time limits and Computer computational resources. I only train on 100 songs and the result is over-fitting. It can be solved via training on the

VI. Conclusion & Future work

This project is based on my recent crush on Garage-band for music composing and the interest in deep learning generated art. Given the recent enthusiasm in Machine Learning inspired art, I hope to continue this work by fixing the GAN model and try other alternative generative model.

Furthermore, I will try to apply more reasonable models like VAE to it. Furthermore, Magenta (Tensorflow-backend for art) will be another good library for exploring.

References

[1] Hao. Dong, Wen. Hsiao, L. Yang, Y. Yang. MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. *arXiv:1709.06298*

[2] K. Choi, G. Fazekas, and K. Cho. A Tutorial on Deep Learning for Music Information.

[3] I. J. Goodfellow, J. Abadie, M. Mirza, B. Xu, D. Farley, S. Ozair, A. Courville, Y. Bengio. Generative Adversarial Networks. *arXiv:1406.2661*

[4] D. Eck, J. Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Technical Report* No. IDSIA-07-02, 2002.