Multi instrument music generation using VAEs

Saravana Rathinam[*]

Stanford University
`saravanr@stanford.edu`

Abstract

MIDI format gives a compressed and quantized representation of a Music which may give us a tractable way to generate long melodies. In this project we generate MIDI audio using Variation Auto Encoders (VAE) for multiple instruments. In addition, we generate MIDI sequences given a short begining sequence and let the model fill in the rest using a Conditional-VAE. We evaluate several β-VAE's by comparing their FID scores. We also publish a MIDI encoded numpy dataset of 140, 944 samples (2.7 Gb) that can be used for further research. Finally we discuss some tips and tricks that were useful in achieving faster training of generative models.

1 Motivation

Composing music is a skill that is acquired by many years of practice. The music itself is the result of the life experiences of the musician, their state of mind, their unconcious and concious thoughts. Their creative talent is subjective and difficult to generalize. Recent advances in Generative models for Music generation have shown impressive results where the focus has been to replace the creative process. Learning the distribution of music creation may be an intractable problem at the moment. However one approach we can take is to build tools that serve as an aid in the creative process. If a musician already has a few ideas in mind on how a song or melody should start, can the problem be modelled as a conditional generative process where given the start of the melody, can a model generate multiple possibilities of how the song can proceed?

In such a generative system, the inputs to the model would be a short MIDI sequence. The system would generate a bunch of sequences that may serve as suggested next sequences and so on. By conditioning on the input and letting the musician choose the path to take, the model can help in the creative process.

2 Related Works

There has been a lot of good work in the area of Music creation. Google's Magenta project explores the role of machine learning in the creative process. In a recent papers (Mittal et al., 2021) built a multi-stage non autoregressive generative model that enabled using diffusion models on discrete data. They generated both unconditional music as well as conditional in-filling. They used a Denoising Diffusion Probabilistic Model Ho et al. (2020) on top of a MusicVAE model that generated the continuous time latent embeddings. Similarly Choi et al. (2021) proposed an Iterative Latent Variable Refinement (ILVR) method to guide the DDPM to generate high quality images based on a given reference image. Also Song et al. (2020) produced a way to accelerate sampling process of a DDPM which can make generation process of sequences faster. In another beautiful apprach, Bazin et al. (2021) built an interactive web interface that transforms sound by inpainting. This approach is similar

[*]Project for course CS236 - Generative Models

to what Meng et al. (2021) built with SDEdit that adapts to editing tasks at test time, without the need for re-training the model.

3 Approach

Initially we planned to use Denoising Diffusion Probabilistic Models (DDPMs) to model the input distribution. DDPMs have been shown to generate high quality music samples Mittal et al. (2021) and also offer controllable generation using post-hoc conditional infilling. However DDPMs operate on continuous space. In Mittal et al. (2021) the authors used a pre-trained variational auto-encoder (VAE) to learn the continuous latent space representation of the input discrete data. This led us to first implement a VAE, whose encoder network can be used to generate continuous representation for DDPMs.

3.1 Auto Encoder and Variational Auto Encoder

A auto encoder consists of an encoder and decoder network. The encoder tries to convert the input x to a smaller latent space representation z. The dimension of z is typically smaller than the dimension of x. The decoder network of an autoencoder converts z back to the input respresentation $\hat{x}$. Once trained, the decoder can be used to generate samples of x. The disadvantage though is that in many cases this training may not be tractable. Also the latent representation z serves as a look up table instead of being a continous representation of the input, which can be interpolated along different dimensions.

A VAE (Kingma & Welling, 2014) solves this problem by trying to learn the distribution $q(z|x)$ of the latent space and then generate z by sampling from this distribution. Typically the distribution chosen is Gaussian which makes it easier to sample using the reparameterization trick. The objective for the VAE thus becomes to maximize the ELBO:

$$ELBO(x; \theta, \phi) = \mathbb{E}_{q_\phi(z|x)}[\log(p_\theta(x|z))] - \mathcal{D}(q_\phi(z|x)||p(z))$$

The first term here is the reconstruction loss and the second term the KL divergence between the prior for z and the one generated by the encoder $q_\phi(z|x)$.

3.2 Dataset

The dataset for the project a combination of the Lakh MIDI Dataset v0.1 Raffel and the MIDI dataset posted at (midi man). The Lakh MIDI data set is a collection of 176, 581 unique MIDI files out of which 45, 129 have equivalent songs in the Million Song Dataset. The (midi man) collection has about 150, 000 midi files. Of these only about 140, 944 midi files were usable because of conversion errors.

For encoding the MIDI files, We considered OctupleMIDI encoding format as proposed and implemented in Zeng et al. (2021). The encoding format was however hard to normalize and get good results. It was not clear what the mininum and maximum values were for each of the MIDI note attributes. This caused the model to generate invalid MIDI combinations which could not be converted to MIDI files. Later we moved to using Google's Magenta (Google) note sequence library. This encodes the MIDI file into a list of note sequences.

Magenta converts a MIDI file to a sequence of notes. Each note has:

1. **Pitch**: The frequencey of the note
2. **Velocity**: The intensity of the note
3. **Instrument**: The instrument where the note should be played (or synthesized)
4. **Program**: A control message that specifies which instrument should be selected to play the note
5. **Start time**: The start time of the note (seconds)
6. **End time**: The end time of the note (seconds)

The data set was split into 70% training samples, 15% test samples and 15% validation samples. The test samples were used for computing test loss and conditional generation of music. The validation set was used for computing the Fréchet Inception Distance (FID) as defined by Heusel et al. (2018) and implemented in the pytorch fid package by Seitzer (2020).

3.3 Encoding and Normalization

In the first few implementations of the VAE we did the mistake of not normalizing the inputs. The pitch, velocity, instrument and program are discrete variables which can take values between $[0, 127]$ (although in some MIDI files, the authors found this was not true. These files were excluded from the dataset). While the start time and end time are continuous time measurements in seconds. The initial normalization approach we took was to ensure the inputs were between $[-1.0, 1.0]$. This was done by dividing the inputs by 127.0 and then using coverting them to $[-1.0, 1.0]$by using $\tanh(x/127.0)$. The decoders output was then re-normalized by doing the inverse $\operatorname{arctanh}(\hat{x}) * 127.0$. This approach did not work. The decoder produced outputs which were not valid (as per MIDI spec). And the generated music sounded annoying and like noise.

The problem was with some of the inputs being categorical variables instead of continuous variables. This led us to consider alternative ways to encode the categorical variables. Initially we considered one-hot encoding, however this lead to a huge amount of memory consumption on GPU. Instead we used an embedding layer to encode these categorical variables into embedding space. The weights of the embedding layer were then used to translate back to the original categorical space. To do this we computed the $L2$ distance and then tried to find the index of lowest distance.

$$x_{categorical} = \arg\min ||\text{embedding.weight} - x_{decoder}||_2$$

. This approach worked much better in practice while not consuming too much memory.

3.4 Model Architecture

A traditional VAE has a single encoder and a single decoder. We had some inputs that are categorical variables and some that are continuous. We also want to generate conditional output, where the VAE is able to fill in the rest of the music sequence given the beginning.

The reconstruction loss for categorical variables is best measured through categorical cross-entropy, while we use the $L2$ norm for continuous output. Thus our overall loss function becomes a combination sum of KL divergence loss ($\mathbb{D}(q_\phi(z|x)||p(z))$), reconstruction losses (time and duration) and categorical cross-entropy loss (for pitch, velocity, instruments and program).

$$\underbrace{\mathbb{E}_{q_\phi(z|x_{time})}[\log(p_\theta(x_{time}|z))] + \mathbb{E}_{q_\phi(z|x_{duration})}[\log(p_\theta(x_{duration}|z))]}_{\text{L2 Reconstruction Loss}} \underbrace{-\mathbb{D}(q_\phi(z|x)||p(z))}_{\text{KL Divergence}} +$$

$$\underbrace{\mathbb{E}_{q_\phi(z|x,x_{partial})}[\log(p_\theta(x|z, x_{partial}))]}_{\text{L2 Conditional Reconstruction Loss}} \underbrace{-\mathbb{D}(q_\phi(z|x, x_{partial})||p(z))}_{\text{Conditional KL Divergence}}$$

$$\underbrace{-\sum_{i=1}^{128} x_{pitch} \cdot \log \hat{x}_{pitch} - \sum_{i=1}^{128} x_{velocity} \cdot \log \hat{x}_{velocity} - \sum_{i=1}^{128} x_{instr} \cdot \log \hat{x}_{inst} - \sum_{i=1}^{128} x_{program} \cdot \log \hat{x}_{program}}_{\text{Categorical Cross Entropy Losses}}$$

Initially we built the encoder using only Linear layers. However later we relalized that since music is inherently auto-regressive, a RNN network may be better suited. After experimenting with RNN, LSTMs and GRU layers, we finally settled with bi-directional GRU as one of the layers of the encoder. The encoder's output was finally passed through two additional linear layers to extract μ and $\log(\sigma)$.

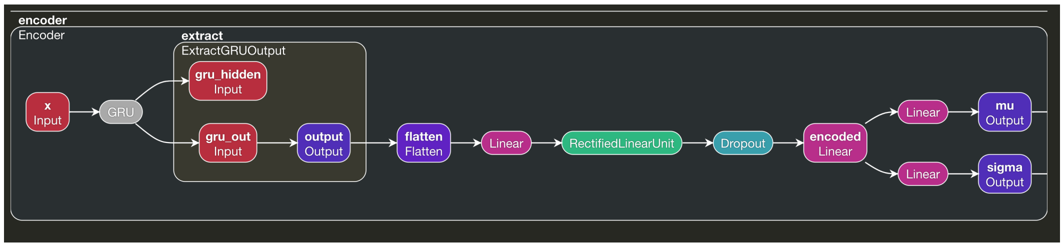


Figure 1: Encoder Architecture

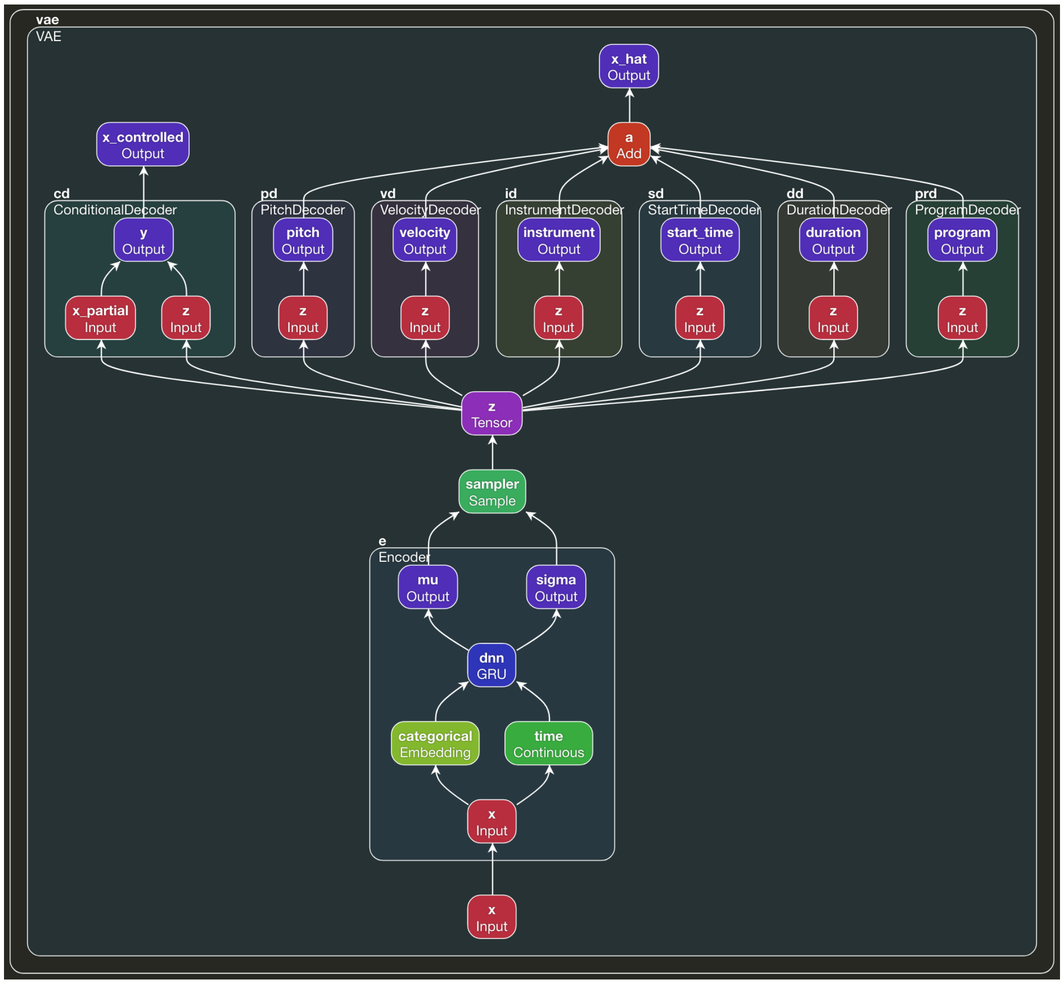


Figure 2: Conditional VAE Architecture with different Decoders for Categorical and Continuous Variables

4 Approach

The following architectures were tried for controllable generation:

- Variation Auto Encoder (VAE) with a single Encoder Decoder
- Beta VAE with $\beta \in [1, 100]$
- VAE with single Encoder and multiple Decoder sharing Z space

Considerations:

- Variables Pitch, Velocity, Instrument and Program are categorical variables.
- Variables Start time and End time are continuous.
- Hence we need different decoders for Categorical and Continuous variables.
- A separate decoder for controlled music generation.
- One-hot encoding cannot be used for categorical variables - Memory usage.

We convert the categorical variables to embedding space and use this representation in training. To decode back to categorical values, we check closes $L1$ distance of the **logits** to each of the embedding weights. The embeddings are learned during training and allow a much smaller memory foot print.

5 Result

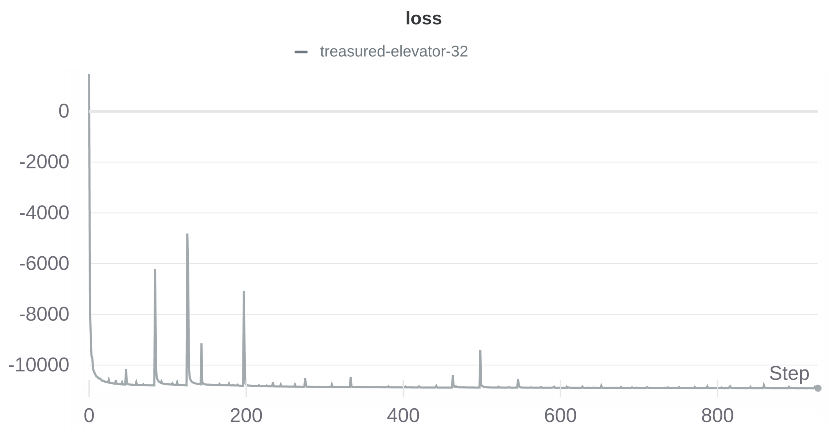
The music evaluation is a qualitative process however in related works authors have used Fréchet distance (FD) Heusel et al. (2018) and Maximum Mean Discrepancy (MMD) Gretton et al. (2012) to measure distance between the models continuous output distribution and the original data distribution in latent space.

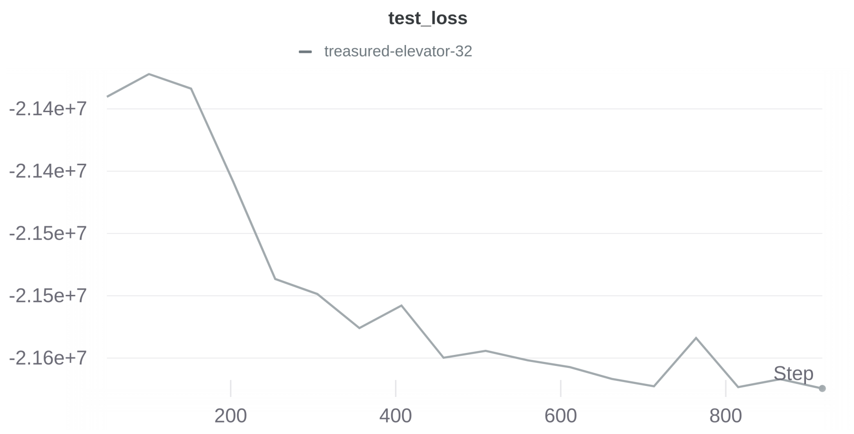
6 Technical Approach

The first baseline was established using a Variational Auto Encoder.

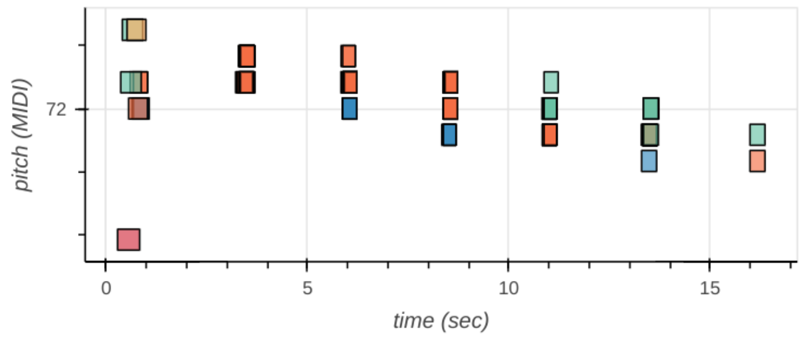
7 Prelimnary Results

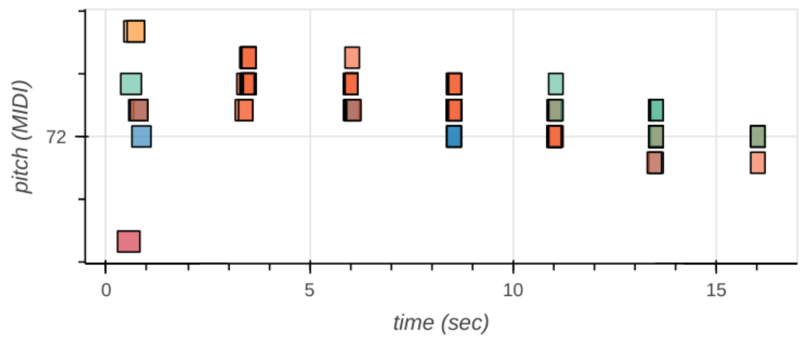
Training and Test Loss of VAE:





Pitch plot of generated MIDI files.





8 Conclusion

The encoded sequences were saved to a single numpy archive file (2.7Gb) which is the largest collection of MIDI samples for ML Research. This is now available at (for ML Research).

References

Bazin, T., Hadjeres, G., Esling, P., and Malt, M. 2021. doi: 10.30746/978-91-519-5560-5. URL http://dx.doi.org/10.30746/978-91-519-5560-5.

Choi, J., Kim, S., Jeong, Y., Gwon, Y., and Yoon, S. Ilvr: Conditioning method for denoising diffusion probabilistic models, 2021.

for ML Research, M. N. S. D. The largest midi note sequence encoded collection for ml researches. URL https://www.dropbox.com/s/rjp2rtvunrkkzuk/midi_magenta_note_seq_data_set_v1.0.npy?dl=0.

Google. Magenta: Magenta is an open source research project exploring the role of machine learning as a tool in the creative process. `https://research.google/teams/brain/magenta/`.

Gretton, A., K, K. M. B., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test, 2012. URL "`http://jmlr.org/papers/v13/gretton12a.html`".

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models, 2020.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes, 2014.

Meng, C., Song, Y., Song, J., Wu, J., Zhu, J.-Y., and Ermon, S. Sdedit: Image synthesis and editing with stochastic differential equations, 2021.

midi man. The largest midi collection on the internet. URL `https://www.reddit.com/r/WeAreTheMusicMakers/comments/3ajwe4/the_largest_midi_collection_on_the_internet/`.

Mittal, G., Engel, J. H., Hawthorne, C., and Simon, I. Symbolic music generation with diffusion models. *CoRR*, abs/2103.16091, 2021. URL `https://arxiv.org/abs/2103.16091`.

Raffel, C. Motion sensors | andriod development. URL `https://colinraffel.com/projects/lmd/`.

Seitzer, M. pytorch-fid: Fid score for pytorch. `https://github.com/mseitzer/pytorch-fid`, August 2020. Version 0.2.1.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models, 2020.

Zeng, M., Tan, X., Wang, R., Ju, Z., Qin, T., and Liu, T.-Y. Musicbert: Symbolic music understanding with large-scale pre-training, 2021.