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

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

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)

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 $\log(\sigma)$.

As illustrated in the figures above, the VAE architecture comprises of an encoder and multiple decoders which share the same z space. This sharing of space while backpropagating the loss ensures that the VAE not only optimizes the reconstruction and KL divergence losses, but also ensure the conditional VAE losses are also optimized.

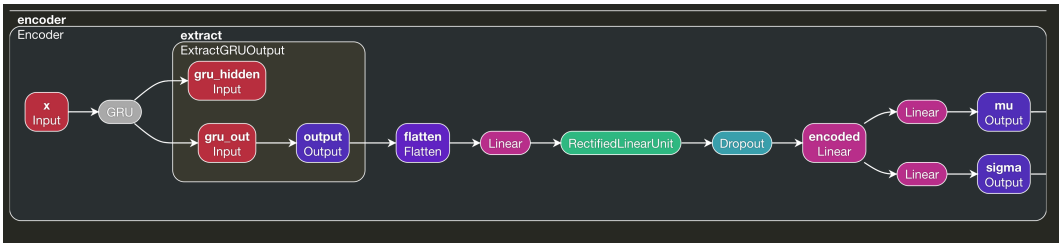

Figure 1: Encoder Architecture

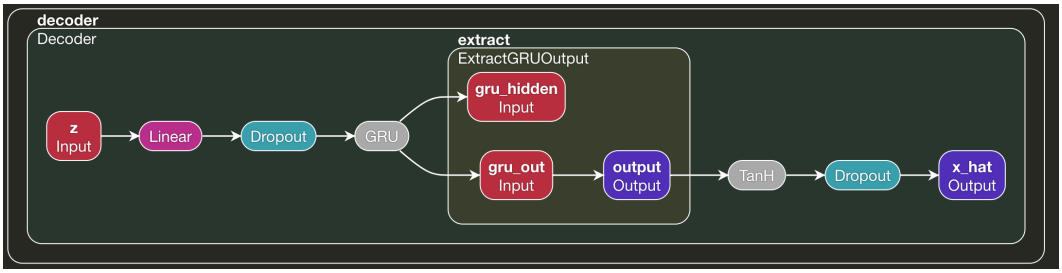

Figure 2: Decoder Architecture

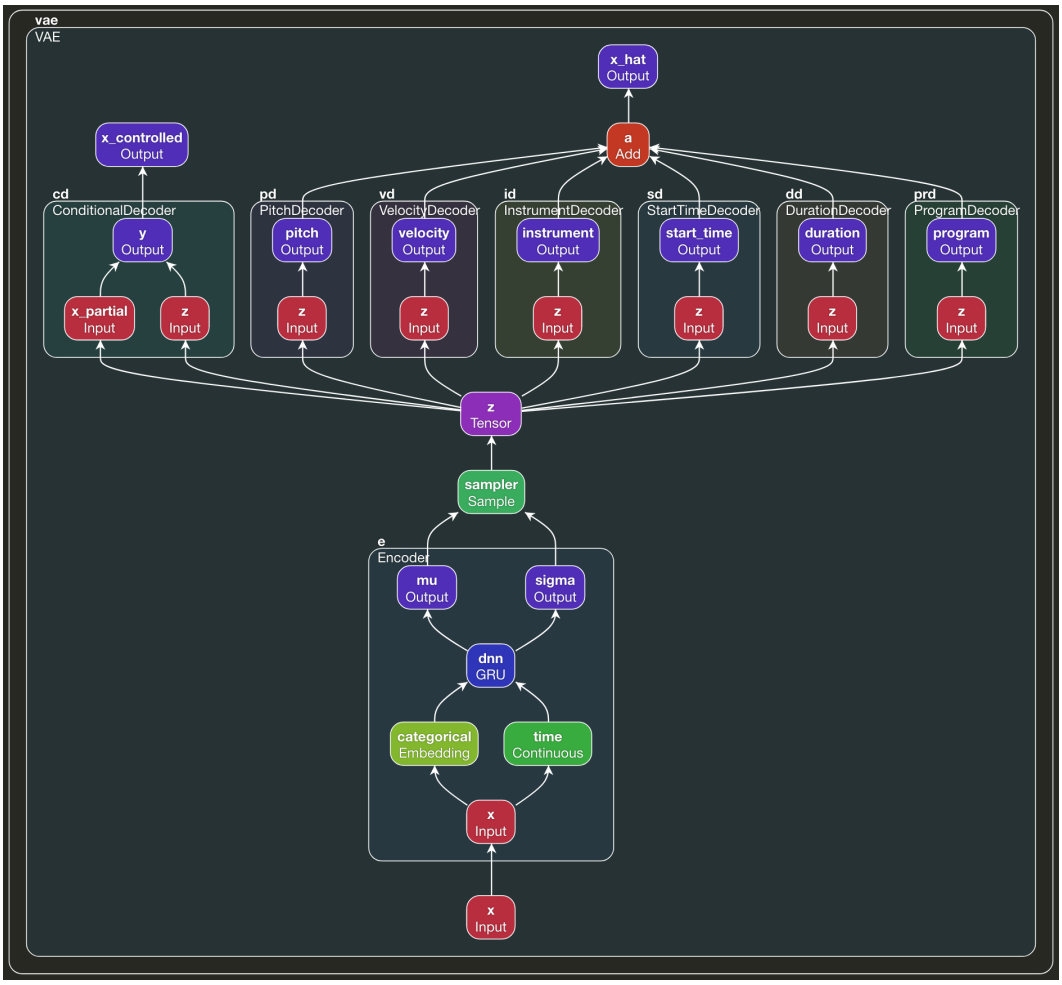

Figure 3: Conditional VAE Architecture with different Decoders for Categorical and Continuous Variables

3.5 Evaluation

The evaluation of the models was done by both measuring the test loss and computing the Fréchet Inception Distance (FID) as defined by Heusel et al. (2018) and implemented in the pytorch fid package by Seitzer (2020), on the validation set. The rule of thumb was to ensure that test loss decreases as training proceeds. All the components of the loss (reconstruction, categorical cross entropy etc.) were also tracked as part of the evaluation.

The data set was split into 70% training samples, 15% test samples and 15% validation samples. This gave us about 98660 files in training split and 21142 files each in test and validation split. The test samples were used for computing test loss and conditional generation of music. The validation split was used to measure FID score.

FID score was measured by sampling 21142 (validation set size) samples from the trained model every 10 epochs. The FID score of conditional generation was also computed for every experiment.

4 Experiments and Results

Initially we did many experiments with VAEs, but most of them failed because of the encoding problem discussed earlier. Once we figured out the proper way to encode the input (using embeddings), the following experiments were run:

- Baseline - Variational Auto Encoder with $\beta = 1$.
- Beta VAE with $\beta \in [10, 100]$
- Conditional VAE music infilling using best Beta value from above.

4.1 Baseline

The baseline was obtained with $\beta = 1$. The training was performed for 100 epochs with a learning rate of $1e - 3$. The test loss and training loss were found to decrease together. (Figure 4.)

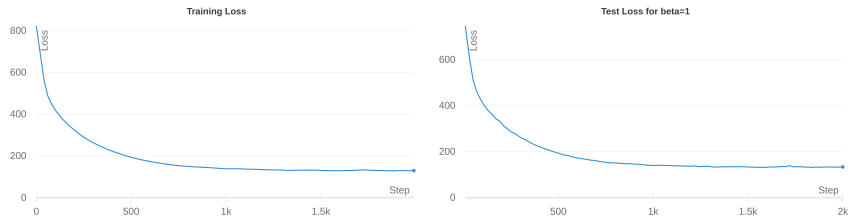

Figure 4: Training and Test Loss for $\beta = 1$

The FID scores also reduced as the training progressed. The curve for FID is not smooth because it was computed every 10 epochs while the training and test loss were computed every epoch. (Figure 5.)

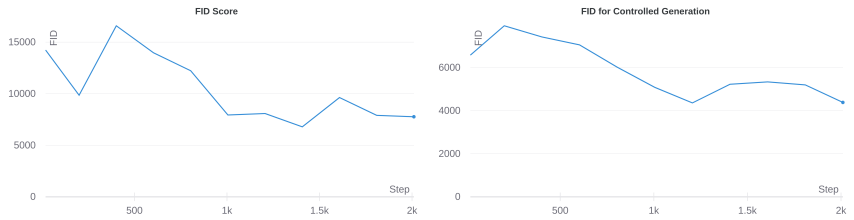

Figure 5: FID for unconditional and conditional (controlled) generation of Music for $\beta = 1$

4.2 Beta VAEs

Several beta VAEs were trained and evaluated with β in $[10, 20, 40, 80, 100]$. The following plots show the losses and FID scores obtained.

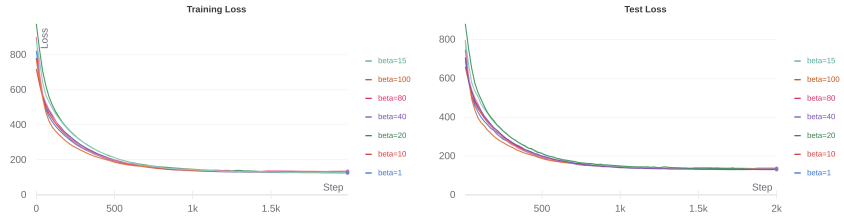

Figure 6: Training and Test Loss plots for different β values

Although the training and test loss appear to be similar for all models, the FID scores show a different story.

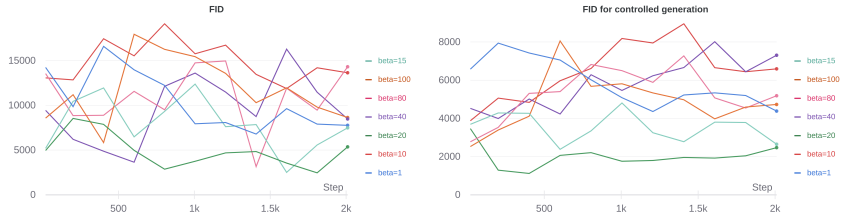

Figure 7: FID for unconditional and conditional (controlled) generation of Music different β values

The best scores were obtained with a β value of 20. This was hard to explain, since the test loss looked similar for these models. However after looking at the individual losses, we noticed a difference in how fast the categorical cross entropy loss for the sequence of instruments converged.

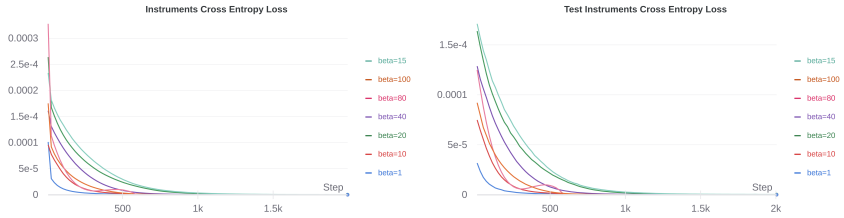

Figure 8: Categorical Cross Entropy Loss for Instruments with different β

Note the difference in rate of convergence for $\beta = 20$.

4.3 Improving Training Speed

We did about 756 training, test loss evaluation and sampling runs for the project. The initial few runs were extremely slow even though the training was done on a relatively high end GPU (NVidia RTX 3090 FE). After profiling the training loop, we found the following:

- Run time conversion of MIDI to PyTorch Tensors was causing memory leak. We could not root cause the memory leak, but we suspect it to be in the magenta MIDI to note sequence conversion layer even when we used more than 24 worker processes. Anyway, instead of converting the files at run time, we decided to cache the numpy files separately.
- Individual numpy files were slow to load. The data module had to load 140, 944 files and fetching them individually was slow. Since we were IO bottlenecked, one option would have been to use python worker threads. However we decided to instead cache all the files into a

single numpy file thereby loading it only once. The file was also copied over to *\dev\shm* (shared memory), so the file stayed entirely in memory.

- Loading across multiple workers was slow. Even though the file was in memory, the CPU had to do a lot of work transfering over the data to the GPU. Although the PCI bus is fast enough, for large quantities of data that is repeatedly fetched, this turned out to be a big bottleneck. To solve this we copied over all the data to the GPU on startup. The tensors took about 4Gb of GPU memory. This transfer however had to be done using a single worker process, since multiple processes cannot access the same CUDA address space.

  We also experimented with different batch sizes and found a batch size of 2048 to be good enough. By doing the above we were able to speed up the training from about 90 mins to about 10 mins. This order of magnitude increase in training and eval speed allowed us to train many variations of the model in a short time.

5 Conclusion

This project was a step in trying to generate long sequences of music (90 seconds+) in a more tractable way. The music generated was also multi-instrument and we also tried unconditional generation. The conditional generation was not as succesful because we conditioned on the begining of the song, which was mostly silent or had a few instruments/notes. However the unconditional generating produced some interesting and pleasant music (Rathinam, a). We also published a dataset (Rathinam, b) that can be used for further research along with the source code for encoding and decoding (Rathinam, c).

We found the best β value for a good FID score, with the given model architecture, to be $\beta = 20$. The cross entropy loss for the sequence and number of instruments in the generated MIDI file, for good FID models seems to converge slower. Its difficult to explain why there is this correlation, but our intution is that instruments tend to be grouped for different song genres and perhaps the model learns this better when the instruments loss converges slower.

Generative modeling is a fascinating area of research. A lot of academic effort has gone into generative models, but their validation is mostly limited to generating images. We can effortlessly classify an image as high quality or low quality by just looking at it. However other types of data, including Music need more effort. Even though FID scores can serve as a good metric to evaluate against, still music seems to be a process that may not conform to a particular distribution or reside on a particular manifold. The best quality music seem to be the one in outliers and these are really hard to find.

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.

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/.

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/.

Rathinam, S. Vae generated music - winter planet. https://soundcloud.com/saravana-r-389436812/cs236-multi-instrument-music-generation-using-vae?si=53eaea6cb734413d80f88bcabc25d3c4, a.

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

Rathinam, S. Multi instrument music generation using vaes source code. https://github.com/saravanr/music-controllable-diffusion, c.

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.