MuseDiffusion: Music generation from Unclear midi with SEquence based Diffusion model

Dongha Kim
Yonsei Unviersity
dhakim@yonsei.ac.kr

Taeheon Kim
Yonsei University
thkim0305@yonsei.ac.kr

Sangmin Lee
Yonsei University
sangmin23@yonsei.ac.kr

Seungjae Lee
Yonsei University
pom0319@yonsei.ac.kr

Woohyeon Choi
Yonsei University
starwh03@yonsei.ac.kr

Jeongwoo Choi
Yonsei University
jwchoi1213@yonsei.ac.kr

Abstract

Diffusion models have achieved state of the art in many generative modeling tasks. Especially, Diffusion models are one of the most promising frameworks of conditional generation. In this project, we propose MUSE-Diffusion: a diffusion model to modify and generate midi data corresponding to the given meta information. We used ComMU [9] dataset, where meta and midi are tokenized and paired. For discrete meta and midi data, we projected discrete token input into continuous domain using embedding function. We trained the diffusion model and embedding weights in MUSE-Diffusion jointly and let the model to understand the relation between meta and midi. We also propose a metric named MSIM to evaluate our generated midi by comparing 3 basic elements of music - rhythm, melody and harmony. Our codes are available at github: https://github.com/YAIxPOZAlabs/MuseDiffusion

1. Introduction

Every day, we can hear various types of music around us. However, composing music is a very difficult task for ordinary people. If music can be generated and modified by only providing some directions, people will be able to access music composition more easily.

Recently, diffusion models have emerged as a powerful deep generative model. Denoising Diffusion Probabilistic Models [7], called as DDPM, has been shown to achieve state-of-the-art results in continuous generative tasks, and is currently an active area of research. In particular, Recent works show that diffusion models are one of the most promising frameworks of conditional generation.

We wanted to perform conditional generation of midi data using diffusion models rather than auto-regressive methods that were usually chosen for this kind of task. Because our goal is to correct wrong midi data conditioned by meta information, the structure of diffusion model seems very promising. We propose MUSE-Diffusion, *Music generation from Unclear midi with SEquence based Diffusion*.

We used ComMU [9] dataset, which is for combinatorial music generation. The meta and the midi are paired and tokenized. Each token represents pitch, velocity, chord, key, etc. Just like previous sentence provide context for the following sentence, meta sequence provides information that decides general context of midi data. If the model can learn the semantic relation between meta and midi, then our model can modify and also generate midi corresponding to meta. So, we investigate diffusion models that handle language modeling. As far as we know, this is the first attempt to use denoising diffusion model to generate music by composing note sequences. To handle discrete data, we choose DiffSeq [5] as the baseline, which is a continuous diffusion model that uses transformer as the denoising model. Through the attention mechanism in transformer [15], we can give conditional-guidance to the model and can effectively handle sequence data.

The ComMU [9] dataset contains various length of sequences. Files with much longer sequence length cause longer training times. To reduce training time, we restricted max sequence length of the model input to 256 in experiments. Also, we try to replace attention layer with Fourier transform layer according to FNet [10] and apply the technique called bucketing to reduce unnecessary padding as much as possible. Nevertheless, we can not train the model from scratch in time. So we utilize pretrained large model, Bert Encoder and pretrained embedding weights to initialize our model. Further explanation is described in Sec. 5.2.

In addition, we developed a new metric named Music Similarity Index Measure, MSIM, which is designed to evaluate the similarity between the generated output

MIDI and the corresponding ground-truth MIDI. By utilizing MSIM, we can evaluate distribution of generated output and compare it against the expected distribution.

2. Related Work

2.1. Combinatorial music generation

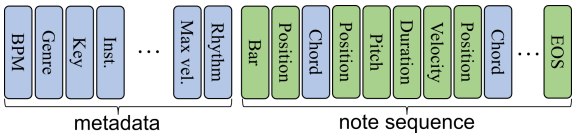

Figure 1. ComMU Dataset

ComMU [9] is a dataset for conditional music generation consists of 12 types of meta data and midi samples manually constructed by professional composers according to corresponding meta. In particular, the ComMU dataset extends the REMI [8] representation, so tokenized note sequences are expressed in the form of a 1d array with tokenized metadata. Previous project using ComMU dataset chose transformer-XL [2] as the baseline and generated the note sequence auto-regressively.

ComMU's 12 metadata are BPM, genre, key, instrument, track-role, time signature, pitch range, number of measures, chord progression, min velocity, max velocity, and rhythm.

2.2. Diffusion Models

Diffusion models are latent variable models. Diffusion has two process: forward process which adds noise to data and reverse process which denoises the input data. The model is trained to perform the reverse process $p_\theta(x_0) = \int p_\theta(x_{0:T})dx_{1:T}$ and learns to recover the data from a random noise, where Gaussian noise is usually chosen for the noise.

With reparameterization trick, we obtain:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$
$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_0$$

By simplifying and reparameterizing presented in DDPM [7], We can parameterize objective of diffusion:

$$L(\theta) = \mathbb{E}_{t,x_0,\epsilon}[||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2]$$

DDIM [14] proposes a generalization of the Markovian chain of DDPM to a non-Markovian chain.

Reverse conditional distribution is

$$q_\sigma(x_t|x_{t-1}, x_0) = \mathcal{N}(\sqrt{\alpha_{t-1}}x_0 + \sqrt{1-\alpha_{t-1}-{\sigma_t}^2}\cdot\frac{x_t - \sqrt{\sigma_t}x_0}{\sqrt{1-\alpha_t}}, {\sigma_t}^2 I)$$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on $\nabla_\theta||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x_T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)) + \sigma_t\mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

and then a sample x_{t-1} can be generated from a sample x_t with

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}(\frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta^{(t)}(x_t)}{\sqrt{\bar{\alpha}_t}}) + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\cdot\epsilon_\theta^{(t)}(x_t) + \sigma_t\epsilon_t$$

By adjusting the magnitude of σ, the stochasticity can be controlled, with smaller values of σ leading to more deterministic behavior. As a result, DDIM enables faster and more deterministic sampling process. To make control easier, DDIM consider σ with the form :

$$\sigma_{\tau_i} = \eta\sqrt{\frac{1-\bar{\alpha}_{\tau_{i-1}}}{1-\bar{\alpha}_{\tau_i}}}\sqrt{1-\frac{\bar{\alpha}_{\tau_i}}{\bar{\alpha}_{\tau_{i-1}}}}$$

where η is a hyperparameter that we can directly control and η = 0 is DDIM and η = 1 is DDPM.

2.3. Diffusion models for Language modeling

There exist two main approaches to deal with sequence data using Diffusion Model: discrete diffusion and continuous diffusion.

Within the discrete diffusion, various methods are available. One such methods is D3PM [1], which uses a method of defining a transition matrix in discrete domain instead of continuous Gaussian noise for each time step. And DiffusER [12] learns the sequence structure by corruption and reconstruction process of data.

Continuous diffusion is the modeling method that embeds discrete text data into continuous domain and applies diffusion process. Diffusion LM [11] proposes full objective to train diffusion model and embedding function so that

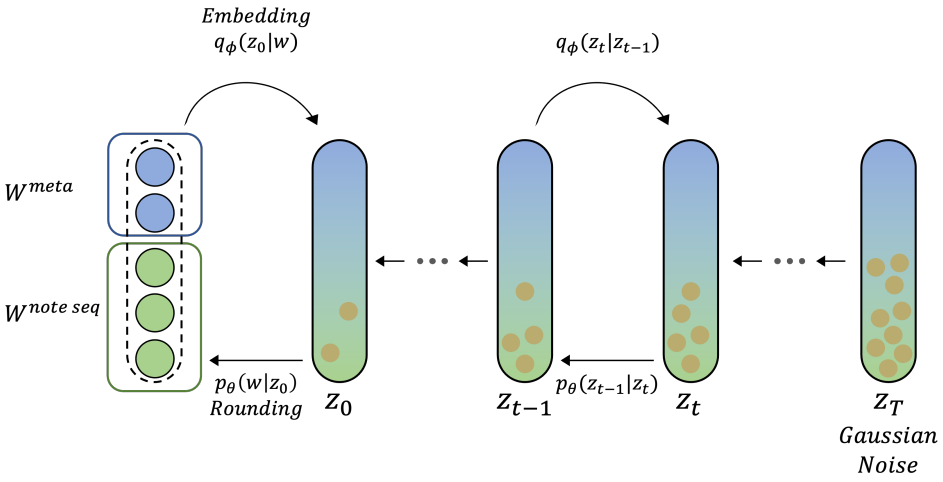

Figure 2. Model Overview

discrete text data can be converted into continuous embedding vectors and be able to perform diffusion process as DDPM. However, Diffusion LM requires extra classifier to add condition in generation, which is very inefficient. DiffuSeq [5] improves the previous work to use diffusion process to handle seq2seq task. DiffuSeq embeds both source and target into continuous domain, concatenates and uses them as model input. To generate target sequence, this model uses partial noising that adds noise only to target sequence while remains source sequence unchanged so that it works as conditioning information in every time step. Thus, DiffuSeq is classifier-free conditional diffusion model.

We decided to use continuous diffusion as our modeling method. Because continuous diffusion is easier to incorporate meta information compared to discrete diffusion and we assume model can learn better relation between meta and midi in continuous embedding space than discrete space. Moreover, vanilla Diffusion Model(DDPM) works in continuous domain that adds and remove continuous Gaussian Noise, so the ability of the diffusion model in the continuous domain is already proven to work well.

3. MUSE-Diffusion

3.1. Forward Process

To transform meta-midi tokens to continuous vectors, we refer to DiffusionLM [11] and DiffuSeq [5] An embedding function, called EMB, maps the discrete meta-midi tokens into a continuous space and this allows MUSE-Diffusion to learn a unified feature space of w^{meta} and $w^{\text{note seq}}$. And we also add a new Gaussian transition $q_\phi(z_0|w^{\text{meta+note seq}}) = \mathcal{N}(EMB(w^{\text{meta+note seq}}), \beta_0 I)$ to the original forward process. Then, the forward process $q(z_t|z_{t-1})$ gradually adds Gaussian noise to the data until it becomes Gaussian noise z_T. DiffuSeq modify the forward process named partial noising, which is to only impose noising on target(note sequence).

3.2. Reverse Process

Reverse process is to recover the original data z_0 by denoising z_t : $p_\theta(z_{0:T}) = p(z_T)\Pi_{t=1}^{T} p_\theta(z_{t-1}|z_t)$ and $p_\theta(z_{t-1}|z_t) = \mathcal{N}(z_{t-1}; \mu_\theta(z_t, t), \sigma_\theta(z_t, t))$. With the partial noising in forward process, meta data is intact and can work as condition information on every denoising step.

The reverse process round a predicted meta-midi vectors z_0 to tokens. According to argmax $p_\theta(w|z_0)$, the most probable meta-midi tokens are reconstructed for each position. If z_0 lies exactly on the embedding vector, then arg-max rounding would be sufficient to recover token. But this is too ideal case and Diffusion LM said that the model fails to generate, empirically. Diffusion LM assumes that this phenomenon is due to the insufficient emphasis on modeling the structure of z_0. So they reparameterize to force the model to explicitly predict z_0 for each denoising time step t, rather than mean μ like conventional diffusion models. Then sampling $z_{t-1} = \sqrt{\bar{\alpha}} f_\theta(z_t, t) + \sqrt{1-\bar{\alpha}}\epsilon$ where $\bar{\alpha}_t = \Pi_{s=0}^{t}(1-\beta_s)$ and $\epsilon \sim \mathcal{N}(0, I)$. Our objective is simplified loss induced from variational lower bound. In fact, this is not longer a valid lower bound, but prior work showed that it empirically made training stable and improved model performance.

$$L(w) = \mathbb{E}_{q_\phi}[\sum_{t=2}^{T} ||y_0 - \tilde{f}_\theta(z_t, t)||^2 + ||EMB(w^y) - \tilde{f}_\theta(z_1, 1)||^2 + R(||z_0||^2)]$$

$\tilde{f}_\theta(z_t, t)$ denotes the fractions of recovered z_0, that is a

note sequence. Although we compute the MSE with respect to the note sequence, with the attention mechanism in the transformer [15], MUSE-Diffusion takes meta into account when reconstructing the note sequence. This means the gradient of this term also affects the learning of meta. Third term is regularization for embedding learning. We use the same embedding function for meta and note sequence.

3.3. Generation

To generate midi using only meta data, we randomly sample note $\text{seq}_T \sim \mathcal{N}(0, I)$ and concatenate it with $EMB(W^{meta})$ to obtain z_T. Then, MUSE-Diffusion gradually denoises z_T until arriving at z_0. Since $f_\theta(z_t, t)$ is the model's output, there is no guarantee that the meta part of $f_\theta(z_t, t)$ will always be the same as the original x_0. So like DiffuSeq, we use an anchoring function. The anchoring function executes rounding on z_t to map it back to word embedding space and replaces the part of recovered z_{t-1} corresponding to w^{meta} with the original x_0. In other words, to sample z_{t-1} with the equations $z_{t-1} = \sqrt{\bar{\alpha}} f_\theta(z_t, t) + \sqrt{1-\bar{\alpha}}\epsilon$, $f_\theta(z_t, t)$ is updated by concatenate (original x_0, $\tilde{f}_\theta(z_t, t)$) before getting z_{t-1}.

3.4. Modification from corrupted midi

At first, we had an idea to use DDIM inversion to modify the corrupted midi into noise so that the noise contains high level feature of input. But this method is inappropriate to MUSE-Diffusion's structure in that the conditioning information is the same in forward and reverse process, so the model always returned the sequence that is equal to the input.

To modify corrupted midi to correct midi corresponding to given meta, we impose noise to the corrupted midi until 0.75 * ddim steps and denoise it.

4. Metric

4.1. MSIM

Since our model is based on diffusion, which reconstructs ground-truth MIDI, we need metric to compare paired MIDI.

To solve this problem, we define new metric named MSIM(Musical Similarity Index Measure). Similar to SSIM [16], it separates the task of similarity measurement into three comparisons: rhythm, melody, harmony.

We use groove similarity [3] as baseline to compare rhythm, but our comparison is based on MIDI, so we can get actual velocity and max amplitude based on it. So, instead of calculating RMS of amplitude in 32 separate sections, we decided to calculate actual amplitude of 32 points.

$$\text{MA} \propto (0.00542676376 * \text{velocity} + 0.310801)^2$$

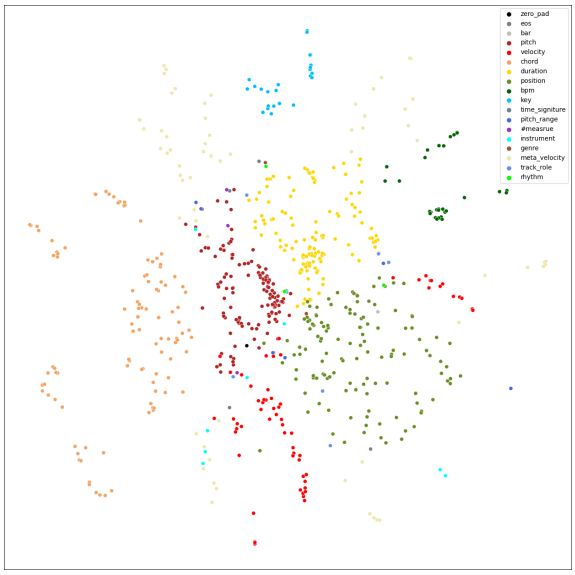

Figure 3. T-SNE plot of trained embedding function. Each color represents the tokens with the same symbolic meaning

Considering dynamic range 20dB, max amplitude of each note can be calculated with expression above. And since it's MIDI, we should consider ADSR to check rhythm of music. We set A=0s, D=length of bar, S=0dB, R=0s, to let each note's amplitude decay over enough, but not too long time. Max amplitude of each note at specific time is as follows:

$$\text{A}(t) = \sum_{n \in \text{active note}} n.\text{MA} * \max(0, 1 - \frac{(t - n.\text{pos})}{\text{bar length}})$$

Music tends to repeat same rhythm in each bar, so we calculate groove vector for each bar and add them to get rhythm vector. With this insight, we define groove vector and rhythm vector as follows:

$$\text{GV} = < \text{A}(0), \text{A}(1/32), \cdots, \text{A}(31/32) >$$

$$\text{RV} = \text{norm}(\sum_{\text{GV}} \frac{\text{GV}}{||\text{GV}||})$$

To calculate rhythm similarity, we should just calculate dot product of two RVs as follows:

$$\text{R} = \text{RV}_1 \cdot \text{RV}_2$$

To evaluate melody, we evaluate progression of pitch. Since moving same number of semitones are considered similar in music, we define progression vector as follows:

$$\text{p}(i) = \text{number of } i \text{ semitone progression}$$

$$\text{PV} =< \text{p}(0), \text{p}(1), \cdots, \text{p}(11) >$$

Like rhythm vector, melody vector and melody similarity are defined as follows:

$$\text{MV} = \frac{\text{PV}}{||\text{PV}||}$$

$$\text{M} = \text{MV}_1 \cdot \text{MV}_2$$

To evaluate harmony, we used chroma similarity [4] as baseline. Since our model has midi instead of mp3 file, it just counts each pitch's appearance. So, chroma vector, harmony vector, and harmony similarity are as follows:

$$\text{c}(i) = \text{number of appearance of } i$$

$$\text{CV} =< \text{c}(do), \text{c}(do\#), \cdots, \text{c}(si) >$$

$$\text{HV} = \frac{\text{CV}}{||\text{CV}||}$$

$$\text{H} = \text{HV}_1 \cdot \text{HV}_2$$

Each similarities are guaranteed as positive since all vectors have only positive vectors.

By assigning weight to each similarities and multiplying them, we can get MSIM as follows:

$$\text{MSIM} = \text{R}^{\alpha}\text{M}^{\beta}\text{H}^{\gamma} \ (\alpha, \beta, \gamma > 0)$$

We simply assign $\alpha = \beta = \gamma = 1$ to calculate MSIM for our experiment.

4.2. 1NNC

We should also check if generated samples generally follow distribution of ground truth. To solve this problem, we used simple 1NNC based on MSIM. There are several IQA methods that uses pretrained model [6, 13], but MIDI data don't have widely used SOTA classifier to use, so we decided to compare only distribution of music. 1NNC uses KNN-classifier by getting nearest neighbor except itself, and calculates accuracy of the classifier. Accuracy≈0.5 means well-trained, low accuracy means overfitting, and high accuracy means underfitting.

4.3. Future Works

For calculating each similarities in MSIM, we suggest few improvements. For harmony similarity, it has a problem that midi with only 'do' is completely different from midi with only 're'. To solve this problem, we can use principle of overtone: calculate chromagram of each pure tones and add them instead of just counting each tones' appearance.

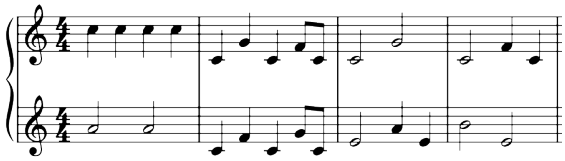

Figure 4. Comparing upper and lower sequence, first bar outputs harmony similarity as 0, second bar outputs melody similarity as 1, and 3rd to 4nd bar as one midi outputs rhythm similarity as 1, melody similarity as 1, and harmony similarity as 0.

For melody similarity, 'do-sol-do-fa-do' is the same as 'do-fa-do-sol-do', but they are not. Similarly, for rhythm similarity, changing the order of bars will output exactly same similarity. We can think of some methods like counting progression of progressions, or giving penalty with some other language comparing methods like dynamic time warping.

5. Experiments

5.1. Dataset

ComMU dataset contains 622,855 pairs of train meta-midi data and 45,780 pairs of validation data in a form of 1D vector including augmentation. There is various length of midi data, from 17 to 2,073. However, it took too much time to train the model with the input data length over 2,000. Thus, we restrict the maximum data length to 256, so the number of train data is 433,075. To see the detail of the data distribution, please check the original paper [9].

In order to further reduce the training time, we apply data bucketing when we load the data. Data Bucketing is a method to form batches by grouping similar length sequence together when processing data in batch units. By doing so, we can minimize zero-padding, which can reduce training time and help with denoising step.

We preprocess the data in two ways. First, we move all the chord related tokens inside midi sequence into meta sequence so that chord information helps to create correct midi data. Then, we randomly corrupt the data so the model get the wrong data and learns to create correct note sequence .We conduct four types of data corruption. First, "Masking Token" randomly replaces each token into a masking token. Second, "Masking Note" randomly replaces each note with a masking token. Third, "Randomize Note" changes the velocity, pitch, and duration values of each note to random value. Finally, "Random Rotating" swaps the position of two randomly selected bars.

5.2. Implementation details

We use the same pipeline as Diffuseq [5] to train our model. Our model is based on 12 layers of Transformer En-

TOKEN	ENCODE VALUE
PAD	0
EOS	1
BAR	2
NOTE PITCH	3-130
NOTE VELOCITY	131-194
CHORD	195-303
NOTE DURATION	304-431
POSITION	432-559
BPM	560-600
KEY	601-625
TIME SIGNATURE	626-629
PITCH RANGE	630-637
NUMBER OF MEASURES	638-640
INSTRUMENT	641-649
GENRE	650-652
MIN VELOCITY	653-717
MAX VELOCITY	653-718
TRACK-ROLE	719-725
RHYTHM	726-728

Table 1. "Encoded tokens to represent the meta and midi in ComMU Dataset."

coder, with embedding dimension 500 and diffusion steps to 2,000. We initialize the Transformer from Diffuseq trained weight and embedding function from embedding weight trained by ComMU to perform the same task but using auto-regressive model.

In higher t, where the data is almost Gaussian noise, it is more difficult for the model to denoise than in smaller t. When we divided denoising time step into four part, Q0 to Q3, we observed that losses were relatively higher at Q3 where noise is the highest. Therefore, we trained the model by selecting time t equal to the batch size and performing importance sampling based on recent 10 losses.

In experiments, we move chord information in note sequence to meta data, because target sequence is much longer than source sequence and chord should not be changed.

5.3. Evaluation Metric

To evaluate how well the correction of corrupted input is conducted, we used MSIM. We compute MISM value with the ground-truth and corrupted midi for the same meta information. Then we compute MSIM value with the ground-truth and modified note sequence returned by our model when the corrupted midi is given as input. To further evaluate the quality of generated note sequence, we check whether pitch and velocity value in note sequence are in correct range given in meta data. (CP&CV)

We also draws t-SNE plot of our model's embedding function, EMB. We can see that tokens with the same symbolic meaning are gathered together, meaning that the model correctly learns the meaning of each token. By this well trained embedding function, our model can modify and generate midi data correctly.

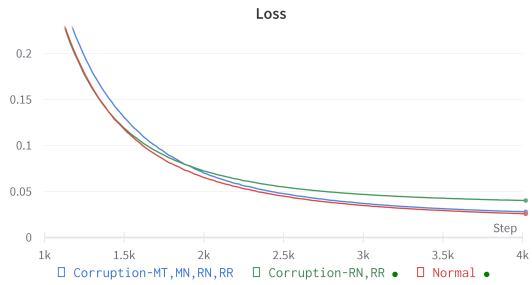

Figure 5. Training loss graphs for 3 cases depending on whether or not corruption. Losses are computed by regularizing embedding plus normal Gaussian at T plus predicting $\mathbf{z}_0$

6. Conclusion and Future Works

6.1. Conclusion

In this project, We propose MUSE-Diffusion, a diffusion model to modificate MIDI data that does not match meta information. Even though it was the first attempt to use denoising diffusion model to generate music by composing note sequences, Our method has shown promising results in generating MIDI outputs that closely match the given meta conditions. Additionally, we suggest MSIM that is new metric for our project. With MSIM, we evaluate musical similarity of generated MIDI and ground-truth MIDI. We believe that our approach will be helpful in the field of conditional music generation and hope that our work will inspire further research and development in this area.

6.2. Future Works

The decision to set a maximum sequence length of 256 for this project was based on considerations of computing power and training time. However, it is true that the maximum length of the ComMU dataset is 2073. Increasing the maximum length can potentially lead to more diverse output. Moreover, since there is a strict order to be followed in note seqeunce, it will be fine-grained if we add the constraints on MUSE-Diffusion's output by putting the classifier outside.

References

[1] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021. 2

[2] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019. 2

[3] Simon Dixon, Fabien Gouyon, Gerhard Widmer, et al. Towards characterisation of music via rhythmic patterns. In *ISMIR*, 2004. 4

[4] Takuya Fujishima. Realtime chord recognition of musical sound: A system using common lisp music. In *Proceedings of the International Computer Music Conference 1999, Beijing*, 1999. 5

[5] Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and LingPeng Kong. Diffuseq: Sequence to sequence text generation with diffusion models. *arXiv preprint arXiv:2210.08933*, 2022. 1, 3, 5

[6] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 5

[7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 1, 2

[8] Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 1180–1188, 2020. 2

[9] Lee Hyun, Taehyun Kim, Hyolim Kang, Minjoo Ki, Hyeonchan Hwang, Kwanho Park, Sharang Han, and Seon Joo Kim. Commu: Dataset for combinatorial music generation. *arXiv preprint arXiv:2211.09385*, 2022. 1, 2, 5

[10] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*, 2021. 1

[11] Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *arXiv preprint arXiv:2205.14217*, 2022. 2, 3

[12] Machel Reid, Vincent J Hellendoorn, and Graham Neubig. Diffuser: Discrete diffusion via edit-based reconstruction. *arXiv preprint arXiv:2210.16886*, 2022. 2

[13] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016. 5

[14] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 2

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 4

[16] Zhou Wang, Alan C Bovik, and Hamid R Sheikh. Structural similarity based image quality assessment. In *Digital Video image quality and perceptual coding*, pages 225–242. CRC Press, 2017. 4