GENERATING MUSIC WITH A SELF-CORRECTING NON-CHRONOLOGICAL AUTOREGRESSIVE MODEL

Wayne Chi *[1] **Prachi Kumar** *[1] **Suri Yaddanapudi**[1]
Rahul Suresh[1] **Umut Isik**[1]
* Equal Contribution [1] Amazon Web Services
waynchi@amazon.com, kumprach@amazon.com, yaddas@amazon.com,
surerahu@amazon.com, umutisik@amazon.com

ABSTRACT

We describe a novel approach for generating music using a self-correcting, non-chronological, autoregressive model. We represent music as a sequence of edit events, each of which denotes either the addition or removal of a note—even a note previously generated by the model. During inference, we generate one edit event at a time using direct ancestral sampling. Our approach allows the model to fix previous mistakes such as incorrectly sampled notes and prevent accumulation of errors which autoregressive models are prone to have. Another benefit is a finer, note-by-note control during human and AI collaborative composition. We show through quantitative metrics and human survey evaluation that our approach generates better results than orderless NADE and Gibbs sampling approaches.

1. INTRODUCTION

There have been two primary approaches to generating music with deep neural network-based generative models. In the first class, music generation is essentially treated as an image generation problem [1, 2]. In the second class, music generation is treated as a musical time series generation problem, analogous to autoregressive language modeling [3–7]. The human process of music composition, however, is often non-chronological. Notes can be filled in anytime throughout the music piece to create new chords and melodies, add harmony, or embellish existing motifs.

In this work, we propose ES-Net [1] , a method that uses elements from both the image-based and time series generation techniques. Our method operates on piano roll images with a 2D convolutional neural network, but autoregressively adds or removes notes one at a time in an arbitrary, non-chronological order. We model the conditional distribution of note add or remove events given pre-existing notes. After sampling from the distribution, we re-input the resulting piano roll into the model to get the distribution of the next add and remove events. From a probabilistic point of view, this corresponds to considering each piano roll as obtained from a randomly ordered sequence of add and remove events and autoregressively modeling the distribution of such sequences of events.

Poor samples due to accumulation of errors is a well-documented problem with autoregressive models [8–11], especially when directly sampling from the conditional distribution (i.e. direct ancestral sampling). While other sampling techniques such as Gibbs sampling [12] can be used to bypass this problem, we show that direct ancestral sampling is sufficient if the data representation includes removal of past samples. This allows the model to detect previous mistakes and fix them.

Our primary use case is melody assistance for users generating musical compositions. Users can feed in a melody as a conditional input and have the model generate musical accompaniments as well as fix any off-beat or out of tune inputs. One distinct advantage of our approach is that it allows note-by-note control for users. A user can undo and redo the generation of individual notes or explicitly add and remove individual notes to collaborate with the model and guide the music composition process. Thus, this approach allows users to have a finer degree of control during sampling and better promotes human and AI collaboration.

The remainder of the paper is organized as follows. In Section 2 we discuss related works. In Section 3 we show how to model a distribution of musical pieces using a new representation of music. In Section 4 we discuss our training procedure. In Section 5 we discuss our sampling procedure. In Section 6 we provide empirical results in the form of quantitative metrics and human evaluation compared against other approaches. Finally, in Section 7 and 8 we describe future work and conclusions.

[1] Code: https://git.io/esnet
Samples: https://git.io/esnet-samples

2. RELATED WORK

Following the introduction of NADE [13, 14] and orderless NADE [15] there have been several works built upon the concept of ordered and unordered autoregressive models. Coconet—the algorithm behind Google's Bach Doo-

dle [2] —is a machine learning model that also uses a convolutional model to generate music by adding counterpoints to existing user input [12]. The difference with this work is that Coconet's inference uses Gibbs sampling rather than direct ancestral sampling. DeepBach [16] generates Bach style chorales using pseudo-Gibbs sampling. PixelCNN [17] models an image autoregressively and generates pixels one by one in a pre-specified order while our generation is unordered. In a NLP setting, recent works also explore non left-to-right ordering [18, 19] and deletion [20].

In general, there is a rich history of using deep learning to generate music [21]. Many of them use autoregressive based approaches. RNN-RBM models temporal dependencies to generate polyphonic music in a single track [22]. Hierarchical RNNs have been used to encode different features of pop music [23]. LSTMs were able to successfully model and generate music as well [24]. Music Transformer is able to capture and generate music with long term structure and motifs [3]. So far, these approaches have been mostly chronological while ours is non-chronological. While GAN-based approaches clearly differ from ours, these methods have shown the ability to generate high quality music. MuseGAN is a GAN-based approach for multi-track piano roll generation [1]. MidiNet uses a CNN-based GAN to generate music [2]. C-RNN-GAN generates music using a RNN based architecture with adversarial training [25]. SeqGAN use GANs for sequence generation and apply it to music generation [26] .

3. PROBLEM DEFINITION

We consider a musical piece $x \in X$ as a point in $\{0,1\}^{T\times P}$ where T is the number of time steps and P is the number of note pitches. This represents a simplified piano roll (PR)—a discrete representation of music as an image matrix across pitch and time. There exists a probability density function $p^{\mathrm{PR}}(x)$ on $\{0,1\}^{T\times P}$ of musical pieces. Note in particular that this does not model velocity and that notes adjacent in time are treated as one continuous held note; we discuss ways to represent velocity and repeated notes in Section 7. Instead of modeling $p^{\mathrm{PR}}(x)$ on $\{0,1\}^{T\times P}$ directly, we model the distributions as $p^{\mathrm{ES}}(s)$ on the set of **edit sequences** (ES). An **edit sequence** of length M is a tuple of M-many **edit events** where an **edit event** is a matrix $e^{(t,p)} \in \{0,1\}^{T\times P}$ that has one entry equal to one, and all other entries equal to zero (i.e. a one-hot matrix). We denote the set of all edit events by $\mathcal{E}$ and of edit sequences of length M by $\mathcal{E}^M$. The following maps edit sequences to piano rolls:

$$\pi : \bigcup_{M=1}^{\infty} \mathcal{E}^M \to \{0,1\}^{T\times P} \tag{1}$$

$$\pi(e_1,\ldots,e_M) = \sum_{i=1}^{M} e_i \pmod 2. \tag{2}$$

[2] https://magenta.tensorflow.org/coconet

where (2) allows edit events to handle either note addition or removal depending on if a previous edit event exists at the same time and pitch.

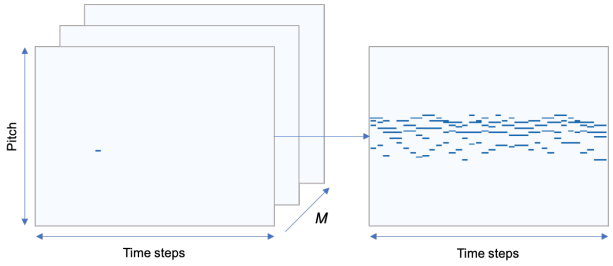


Figure 1. Mapping from an edit sequence (left) of length M to a piano roll (right). Each slice in an edit sequence is the addition or removal of a note.

The mapping between the two joint probability distributions is as follows:

$$\begin{aligned} p^{\mathrm{PR}}(x) &= p^{\mathrm{PR}}(\{(t_1,p_1),\ldots,(t_N,p_N)\}) \\ &= \sum_{s\in\pi^{-1}(x)}^{\infty} p^{\mathrm{ES}}(s) \end{aligned} \tag{3}$$

where N is the number of notes in the piano roll, (t_i,p_i) is the time and pitch of a note or edit event, $\pi^{-1}(x)$ is the inverse image set of $\pi(x)$, and s is a sequence of edit events $(t_1,p_1)\ldots(t_M,p_M)$ where $M \geq N$. We can further factorize $p^{\mathrm{ES}}(s)$ as:

$$\begin{aligned} p^{\mathrm{ES}}(s) &= p^{\mathrm{ES}}\big((t_1,p_1),\ldots,(t_M,p_M)\big) \\ &= \prod_{i=1}^{M} p^{\mathrm{ES}}\big((t_i,p_i)|(t_1,p_1),\ldots,(t_{i-1},p_{i-1})\big) \end{aligned} \tag{4}$$

We assume that $p^{\mathrm{ES}}((t_i,p_i)|(t_1,p_1),\ldots,(t_{i-1},p_{i-1}))$ is ordering invariant (i.e. the ordering of edit events in an edit sequence does not affect the resulting piano roll).

Our goal is to train a model to map the distribution of edit sequences $p^{\mathrm{ES}}(s)$. By sampling autoregressively from $p^{\mathrm{ES}}(s)$, we will generate a sequence of edit events that can be mapped back into a piano roll representation and then converted to MIDI.

3.1 Orderless NADE

We compare our approach to orderless NADE which generates music by randomly choosing an ordering and sampling notes one by one until termination. We can represent the iterative notewise addition of orderless NADE as a special case of edit sequences where edit events can only represent note addition. Let us call this distribution $p^{\text{O-NADE}}(x)$. Since notes are only added, $M = N$ for unconditioned generation; thus, there is a finite set of orderings and we can factorize $p^{\text{O-NADE}}(x)$ as:

$$\sum_{\sigma\in S_N}\prod_{i=1}^{N} p^{\text{O-NADE}}\big((t_{\sigma(i)},p_{\sigma(i)})| \\ (t_{\sigma(1)},p_{\sigma(1)}),\ldots,(t_{\sigma(i-1)},p_{\sigma(i-1)})\big)$$

where S_N is the set of all permutations $\{1, 2, ..., N\} \rightarrow \{1, 2, ..., N\}$. This factorization is equivalent to orderless NADE [15]. In practice, the orderless NADE approach leads to poorer musical samples due to accumulation of errors which we confirm in Section 6.

4. TRAINING

Given an input piano roll, $\mathcal{I}$, and target piano roll, $\mathcal{T}$, we train our model to output the conditional probabilities $p^{\text{ES}}((t_i, p_i)|(t_1, p_1), \ldots, (t_{i-1}, p_{i-1}))$ of the next edit event in edit sequences that can recreate the target from the input piano roll. For each piano roll in the training set, we generate the input piano roll by (a) masking existing random notes and (b) adding extraneous random notes to the target piano roll. We train the model to recreate $\mathcal{T}$ from $\mathcal{I}$; augmentation (a) trains the model to add notes and (b) trains the model to remove notes. For each target piano roll, we generate multiple augmented inputs with varying number of notes masked and added, in order to train the model to handle varying number of differences between the input and target. We find that masking between 0 to 100 percent of all existing notes and adding 0 to 1.5 percent of all possible extraneous notes gives us the best results.

Our goal is to have the model output the conditional probabilities for the next edit event. Since we assume ordering invariance in (4), we can also assume that every note difference between $\mathcal{I}$ and $\mathcal{T}$—whether it requires the addition or removal of a note—is equally likely to be the next edit event. Thus, we model the distribution of edit events for the next step as the uniform distribution U supported on the symmetric difference $\mathcal{I}\Delta\mathcal{T}$ between $\mathcal{I}$ and $\mathcal{T}$ (i.e the exclusive or of each note between $\mathcal{I}$ and $\mathcal{T}$).

We use the Kullback-Liebler divergence between U and the model's output distribution as the loss function:

$$\mathcal{L}(\mathcal{I}, \mathcal{T}, P) = D_{KL}(P \parallel U), \tag{5}$$

where P is the softmax over the model's logits at each time and pitch. Normally, binary cross-entropy loss—where the label is the next note—would be used, but since we assume ordering invariance in (4), the next note is equally likely to be any of the future notes. Therefore, training with (5) is equivalent to training many times where the label is randomly chosen from future notes.

4.1 Model

We train a model based on the U-Net architecture [27]. This choice is not critical as our approach should generalize to other CNN architectures. We describe our approach for reproducibility. Our U-Net contains five downsampling blocks and five upsampling blocks. In each block there contains a batch normalization layer, two 2D convolutional layers each with a 3x3 kernel, a max pooling layer, and a drop out layer with a 0.5 dropout rate. We begin with 32 filters. We double the number of filters after each downsampling block and halve the number of filters after each upsampling block. We use the Adam optimizer [28] with a learning rate of 0.001. We use RELU for our activation function, except for the final layer where we output a linear activation at each time and pitch. Finally, we apply softmax over the logits when calculating the loss and during sampling.

5. INFERENCE

We sample from the model's output probabilities through direct ancestral sampling. We feed the input melody to the model, sample from the softmax over all times and pitches to determine the next edit event, modify the input melody based on that edit event, and then feed that melody back into the model. We repeat this over multiple iterations and condition each time on our previous predictions. Since we do not differentiate between adding and removing notes during training, the sampling process is the same for any type of edit event. We allow users to restrict the number of notes to remove; this prevents the model from completely overwriting the original input. We also allow users to control how many sampling iterations are performed. Lastly, we allow the user to change the temperature during sampling. By changing the shape of the distribution, users can make compositions more or less "creative" at the risk of lowering quality. We surface these hyperparameters to allow users to more freedom and customizability when generating music compositions.

6. EMPIRICAL EVALUATION

We compare our approach against orderless NADE and Gibbs sampling using quantitative metrics and human survey evaluation. We also describe a notewise approximate log likelihood calculation for our approach and explain why log likelihood is not a good metric for comparing our approach to orderless NADE. We build an orderless NADE model using the approach described in Section 3.1 and training with only masked notes. We use Coconet [12] to represent Gibbs sampling. While our main focus is to only use Coconet for sampling technique comparisons, there are a few notable differences between Coconet and our approach. First, Coconet does not explicitly train the model to remove notes, but notes—including the input—may be removed during the Gibbs sampling masking process; our approach explicitly models note removal. Second, Coconet assumes that there are four instruments and that "each instrument plays exactly one pitch a time" [12]; our approach has no such constraint and can generate music across all times and pitches. Third, Coconet trains a CNN that preserves the same size for each layer; we train a model based on the U-Net architecture. Since Coconet is trained on the JSB Chorales dataset, we evaluate our results and Orderless NADE's results using the same dataset and the same train-val-test split in order to provide a fair comparison. For all other parameters (e.g. temperature), we maintain identical settings for each approach in order to benchmark fairly.

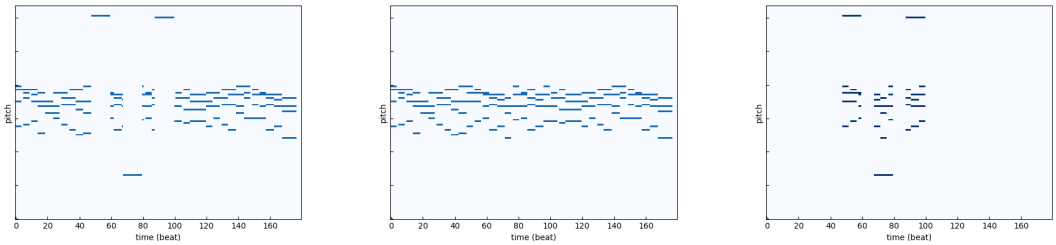


Figure 2. Input piano roll (left), target piano roll (middle), and symmetric difference between the input and target piano rolls (right)

6.1 Data

We use the Infinite Bach dataset [3] and the JSB Chorales dataset [4]. The JSB Chorales and Infinite Bach datasets contain MIDI files—382 for JSB Chorales and 498 for Infinite Bach—of chorales harmonized by J.S. Bach. The MIDI files in Infinite Bach dataset are generally longer in duration allowing for approximately three times more samples overall compared to the JSB Chorales dataset. Since the Gibbs sampling model is trained on JSB Chorales, we use the JSB Chorales dataset for benchmarking.

For both datasets, we preprocess the data by: 1) mapping MIDI to its piano roll representation using a sixteenth-note quantization, 2) converting multi-track inputs into a single track by merging all tracks, and 3) splitting each MIDI into multiple 2 or 8 bar samples.

6.2 Log Likelihood

We calculate log likelihood using equation (3). Since for each piano roll x the inverse image $\pi^{-1}(x)$ is infinite, the sum cannot be calculated exactly; thus, we calculate an approximate log likelihood for a subset of all possible edit sequences in $\pi^{-1}(x)$. This value lower bounds the true log likelihood value. We compare this lower bound to the log likelihood for orderless NADE. Since our method removes notes as well, the proposed model is modeling a distribution with larger support so we do not expect the likelihood value of our method to be better than orderless NADE's. Our likelihood values show that—in the toy case when the sum can be sufficiently expanded—the likelihood lower bound value approaches that of orderless NADE.

Consider a graph where each vertex corresponds to a piano roll state and each edge corresponds to an edit event. A path in the graph corresponds to an edit sequence described in equation (2). As we traverse over a path, we calculate the log likelihood of the edit sequence corresponding to that path.

For each input $\mathcal{I}$ and target $\mathcal{T}$ pairing, we calculate our log likelihood over multiple levels, traversing over edit sequences of length $K + 2d$ at level d. K is the minimum number of edit events needed to reach the target from the input. All K edit events are unique along time and pitch. For level $d = 0$, there exist $K!$ different edit sequences. We calculate the average log likelihood over a randomly chosen subset of these edit sequences and approximate it over all $K!$ edit sequences. During the traversal we keep track of the most probable (time, pitch) predictions that do not occur in the edit sequences, and add them to a pool Q. We keep these predictions as they will appear in the most probable edit sequence paths at level $d = 1$. For level $d = 1$, we traverse down the same paths, but we add two edit events with the same time and pitch chosen from Q to the path. This increases the path length to $K + 2$ and results in the same target pianoroll since the two new edit events cancel out. We approximate the log likelihood sum over all possible edit sequences. We repeat this for each (time, pitch) pair in Q. This process can be repeated until level $d = D$ expanding our coverage of the edit sequence graph along the most probable paths.

We calculate the approximate log likelihood as:

$$\frac{1}{K} \log \sum_{d=0}^{D} \sum_{Q} \frac{(K+2d)!}{2^d} \frac{1}{|S|} \sum_{s \in S} p^{\mathrm{ES}}(s)$$

where S is a random subset of $K+2d$ length edit sequences that can transform $\mathcal{I}$ to $\mathcal{T}$. [5] As we increase the levels of our approximation, our log likelihood will converge towards orderless NADE which we see in Table 1 at $d = 1$.

Since music completion is a task with high uncertainty, the large number of low probability predictions leads to underflow issues, which we avoid by using the log-sum-exp trick. Also, since log likelihood in this case is highly dependent on the number of notes in a piece, we compute an approximate *notewise* log likelihood by dividing the approximate log likelihood by the minimum number of note additions and removals needed to reconstruct the target pianoroll. We do not use log likelihood to compare our approach with Gibbs sampling used in Coconet as they use framewise log likelihood, which is different than our calculation [12].

6.3 Quantitative Metrics

We calculate several quantitative metrics to compare the quality of generated music using our approach, orderless NADE, and Gibbs sampling. For each approach, we generate 3405 bars of music—the same number of bars in the

[3] https://github.com/jamesrobertlloyd/infinite-bach

[4] https://github.com/czhuang/JSB-Chorales-dataset

[5] We divide the $K+2d$ factorial by 2^d as we cannot "remove" before we "add" a note.

Approach	Notewise Approximate Log Likelihood
ES-Net	-0.635
Orderless NADE	-0.558

Table 1. Notewise approximate log likelihood for reconstructing 10 missing notes from each test sample.

training data—and compare them to the training data. We generate the music by conditioning on 150 8-bar monophonic inputs. We evaluate on the following metrics designed in [1, 29]:

- PC - Number of unique pitch classes used. Notes whole octaves apart from each other (e.g. C4 and C5) belong to the same pitch class.
- P - Number of unique pitches used.
- ISR - In-scale rate which is the proportion of all notes that lie in C Major [6] .
- PR - Polyphonic rate which is the proportion of timesteps where the number of pitches being played is greater than or equal to 4.

We use pypianoroll [29] to calculate these values.

	PC	P	ISR	PR
Training Data	6	46	0.541	0.917
ES-Net	**6**	**46**	**0.540**	**0.930**
Gibbs Sampling	**6**	**46**	0.535	0.898
Orderless NADE	8	55	0.559	0.759

Table 2. Quantitative metrics for each approach. Closer to training data is better. Bold values are best between the three approaches.

We observe that our approach and the Gibbs sampling approach both produce music that have similar characteristics to the dataset, while orderless NADE shows less similarity to the dataset. As seen in Table 2, our approach is the closest for all four metrics, with Gibbs sampling tying for number of unique pitches and pitch classes used.

	Bhattacharyya	Kolmogorov-Smirnov		
		df	D	p
ES-Net	0.028	46	0.17	0.49
Gibbs Sampling	**0.021**	46	0.13	0.83
Orderless NADE	0.049	46	0.17	0.49

Table 3. Various metrics for how far pitch appearance frequency is from the training data. Lower is better and bolded is best for Bhattacharyya distance. The Kolmogorov-Smirnov test is unable to show significant difference between any of the approaches and the training data.

[6] The C-Major scale was chosen arbitrarily.

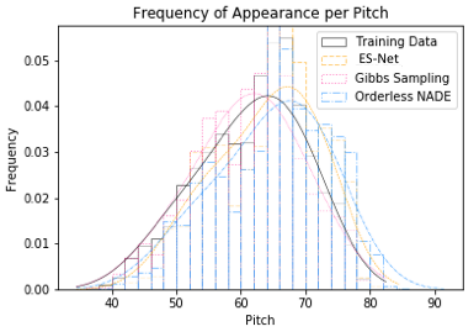


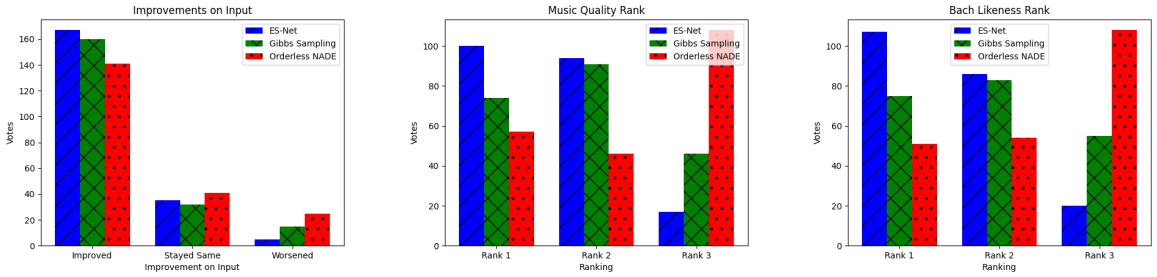
Figure 3. Frequency of occurrence for each pitch bin. Each bin is two pitches (i.e. one bin contains both pitch 31 and 32).

In Figure 3, we plot the frequency of pitch values for each approach and compare with the distribution of pitches in the training data. We observe that the distribution of pitches for all three approaches is very similar to that of the training data. In Table 3, we evaluate the similarity of each approach's pitch appearance frequency to the training data using various metrics. We calculate the Bhattacharyya distance [30] showing Gibbs sampling as the closest to the training data and orderless NADE as the furthest from the training data. We perform Kolmogorov-Smirnov tests and are unable to show significant differences between each approach and the training data.

6.4 Human Evaluation

We conducted a human opinion test in order to compare our approach against orderless NADE and Gibbs sampling. We generated 8 bar samples with a pitch range from 36 to 81. We assume 4/4 time (i.e. 4 beats per bar) and quantize to 16 time steps per bar (i.e. 1/16th note). We assume two notes continuous in time as one note. For orderless NADE, we sample 400 times to generate samples that approximate to 4 pitches per time step. Coconet optimizes the number of iterations it requires. Since our approach allows for the model to both add and remove notes, there is no fixed number of iterations to run the model; instead, the model eventually stabilizes and adds or removes the same set of notes repeatedly. We sample for 10,000 iterations for our approach. When conducting the surveys, we chose a large number of iterations to ensure stabilization; through later experiments, however, we found that the output almost always stabilizes before 2000 iterations.

Each survey contained fifteen randomly chosen sets of comparisons where each set of comparisons contained a random sample from each of the three approaches. Each of the samples in each set were randomly ordered. All three samples in a set were conditioned on the same input track which was also given to the participant. In order to simulate real user input, we created input tracks by taking two bar user inputs from the Bach Doodle Dataset [31]—a dataset of real user inputs to Coconet and its resulting composition—and repeated them four times to form eight



(a) (b) (c)

Figure 4. Human Survey Evaluation Ratings: (a) describes whether users thought a sample improved on the input. (b) describes user rankings for music quality. (c) describes user rankings for how similar a sample is to real Bach data. Bars are ordered from left-to-right as ES-Net, Gibbs Sampling, and Orderless NADE.

bars. Bach Doodle ranks their inputs based off of user feedback from the resulting Coconet composition; we chose an equal number of samples randomly from each feedback level. Each survey contained the same fifteen sets of comparisons. These inputs are monophonic (i.e. only one pitch per time step). For each set of comparisons, users were asked (a) if each sample improved on the input, (b) to rank the samples based on music quality, and (c) to rank the samples based on similarity to music composed by Bach.

We receive a total of 207 ratings for question (a), 211 ratings for question (b), and 213 ratings for question (c). [7] For question (a), we see that all approaches are comparable and each approach almost always improved the input as seen in Figure 4(a). For questions (b) and (c), we see in Figures 4(b) and 4(c) that our edit sequence approach is the best approach while the orderless NADE approach is the worst. We perform a Kruskal-Wallis H-test across all ratings for questions (b) and (c). We show that there is a statistically significant difference ($\mathcal{X}^2(2) = 64.47$, $p < 0.001$ for question (a) and $\mathcal{X}^2(2) = 73.07, p < 0.001$ for question (b)) between the three models. We use the Wilcoxon signed-rank test to conduct a pairwise post-hoc analysis. We show that there is a statistically significant difference ($p < 0.001$ for questions (b) and (c)) between our approach and both the Gibbs sampling and orderless NADE approaches.

7. FUTURE WORK

We currently trained on a limited number of datasets, both of which are based on Bach chorales. There is no reason, however, that our approach should be limited to any feature of Bach. By training on other datasets, we will be able to evaluate how well our approach generalizes.

We show that allowing the model to remove notes increases music quality which we believe is due to the model correcting its past mistakes. During our training process, we generate random notes in order to mimic those mistakes. Rather than merely mimicking those mistakes, however, we can generate real mistakes by feeding outputs from the model back into itself. We believe that this self-adversarial training paradigm will allow the model to capture more realistic sampling mistakes and further improve performance.

Our current data representation does not convey features such as note velocity, repeated notes, or explicit note duration. These features, however, can add to the technical and emotive quality of music. We can map these new features as additional channels and concatenate this information with our existing piano roll. This new data representation will allow our model to learn from these new feature dimensions and produce more expressive and technically challenging music.

An advantage of our algorithm is the ease with which we can extend our approach to other use cases. For instance, currently our model generates fixed length outputs depending on the length of the training samples. In this way, we can extend user melodies up to a fixed length; however, we never explicitly train our model to extend inputs. By augmenting our dataset so that the latter portion of each sample is masked out, we can explicitly train our model to extend melodies. Then, during sampling, we can generate a fixed length output, feed the latter portion of that output back into the model to generate a new output, concatenate those two outputs together, and repeat. This would allow us to extend melody repeatedly rather than up to a fixed length output.

8. CONCLUSION

We show that by modeling removal of notes, we can train a model to produce better music by fixing past mistakes and preventing accumulation of errors. We discuss how our note-by-note approach allows for a finer degree of control and better human and AI collaboration. We demonstrate how to map an edit sequence representation into a piano roll representation and how we can use that to model a distribution of musical pieces. We discuss how we train our model by masking and adding erroneous notes and how we sample from our model during inference. Finally, we show through quantitative metrics and human evaluation that our approach is able to generate musical compositions that are of better quality than orderless NADE and Gibbs sampling.

[7] Some users did not answer all three questions per set of samples. Partial or incomplete rankings were discarded.

9. REFERENCES

[1] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[2] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation," *arXiv preprint arXiv:1703.10847*, 2017.

[3] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, I. Simon, C. Hawthorne, N. Shazeer, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure," 2018.

[4] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[5] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[6] C. Payne, "Musenet, 2019," *URL https://openai.com/blog/musenet*, 2019.

[7] J. Wu, C. Hu, Y. Wang, X. Hu, and J. Zhu, "A hierarchical recurrent neural network for symbolic melody generation," *IEEE Transactions on Cybernetics*, 2019.

[8] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 1171–1179.

[9] F. Huszár, "How (not) to train your generative model: Scheduled sampling, likelihood, adversary?" *arXiv preprint arXiv:1511.05101*, 2015.

[10] A. M. Lamb, A. G. A. P. Goyal, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," in *Advances In Neural Information Processing Systems*, 2016, pp. 4601–4609.

[11] A. Venkatraman, M. Hebert, and J. A. Bagnell, "Improving multi-step prediction of learned time series models," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[12] C.-Z. A. Huang, T. Cooijmans, A. Roberts, A. Courville, and D. Eck, "Counterpoint by convolution," *arXiv preprint arXiv:1903.07227*, 2019.

[13] H. Larochelle and I. Murray, "The neural autoregressive distribution estimator," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 29–37.

[14] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle, "Neural autoregressive distribution estimation," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 7184–7220, 2016.

[15] B. Uria, I. Murray, and H. Larochelle, "A deep and tractable density estimator," in *International Conference on Machine Learning*, 2014, pp. 467–475.

[16] G. Hadjeres, F. Pachet, and F. Nielsen, "Deepbach: a steerable model for bach chorales generation, june 2017," *arXiv preprint arXiv:1612.01010*.

[17] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*, 2016.

[18] M. Stern, W. Chan, J. Kiros, and J. Uszkoreit, "Insertion transformer: Flexible sequence generation via insertion operations," *arXiv preprint arXiv:1902.03249*, 2019.

[19] W. Chan, N. Kitaev, K. Guu, M. Stern, and J. Uszkoreit, "Kermit: Generative insertion-based modeling for sequences," *arXiv preprint arXiv:1906.01604*, 2019.

[20] J. Gu, C. Wang, and J. Zhao, "Levenshtein transformer," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 181–11 191.

[21] J.-P. Briot, G. Hadjeres, and F.-D. Pachet, "Deep learning techniques for music generation–a survey," *arXiv preprint arXiv:1709.01620*, 2017.

[22] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "High-dimensional sequence transduction," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 3178–3182.

[23] H. Chu, R. Urtasun, and S. Fidler, "Song from pi: A musically plausible network for pop music generation," *arXiv preprint arXiv:1611.03477*, 2016.

[24] B. L. Sturm, J. F. Santos, O. Ben-Tal, and I. Korshunova, "Music transcription modelling and composition using deep learning," *arXiv preprint arXiv:1604.08723*, 2016.

[25] O. Mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," *arXiv preprint arXiv:1611.09904*, 2016.

[26] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[27] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[29] H.-W. Dong, W.-Y. Hsiao, and Y.-H. Yang, "Pypianoroll: Open source python package for handling multitrack pianoroll," *Proc. ISMIR. Late-breaking paper;[Online] https://github.com/salu133445/pypianoroll*, 2018.

[30] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bull. Calcutta Math. Soc.*, vol. 35, pp. 99–109, 1943.

[31] C.-Z. A. Huang, C. Hawthorne, A. Roberts, M. Dinculescu, J. Wexler, L. Hong, and J. Howcroft, "The Bach Doodle: Approachable music composition with machine learning at scale," in *International Society for Music Information Retrieval (ISMIR)*, 2019. [Online]. Available: https://goo.gl/magenta/bach-doodle-paper