Generative Modeling of Bach Chorales by Gradient Estimation

Eric Zhang
Harvard University
ekzhang@college.harvard.edu

Romil Sirohi
Harvard University
rsirohi@college.harvard.edu

Abstract

We introduce a new generative model for music composition, based on annealed Langevin dynamics and a noise-conditional score matching algorithm. Unlike implicit models such as GANs, this learns an explicit distribution of the input data. We study if Langevin dynamics and score matching can combine the controllability of Markov chain Monte Carlo (MCMC) methods with the global view and fast convergence of stochastic gradient descent, to produce high-quality, structured musical compositions.

1 Introduction

A fair amount of recent research has focused on computer-assisted or computer-driven music composition. Music has been an integral component of human civilization for millennia, and composers have captivated human imagination. That's not to mention the talented musicians who have interpreted musical works and created their own magic. In this sense, music generation is a two part problem, involving both composition and audio synthesis.

Included in any Western classical musician's journey is the study of Bach chorales. Indeed, beginning music theory students are often taught to compose in the style of these compositions, almost following strict formulae. These 389 chorales are polyphonic but quite homogeneous given their size [8]. Each chorale is roughly one minute long. They are set to a Lutheran hymn and sung by four voices. The melody is sung by the soprano voice (the highest), and the alto, tenor, and bass voices create texture and harmony.

These parts have to be carefully composed. They articulate the syllables of the hymn, create contrary motion, and emphasize cadences. Bach chorales generally follow "common practice" four-part harmonies and are considered a gold standard of the Baroque era of Western classical music. Beyond harmonizing, each voice must proceed in a certain direction. While maintaining harmony and counterpoint, the voices must tell a coherent story, which makes sense in the context of the piece as a whole. This is a moderate task for a music theory student: possible to do with some amount of training, but not trivial.

In 1988, this problem was framed as an optimization problem. Researchers constructed 300 constraints, relying upon expert knowledge. They then found music that satisfied the constraints [5]. However, not all of them matched the nuances of Bach's music. More recent approaches to this problem have shied away from using human expert knowledge in favor of neural networks. A 1992 approach breaks down the music composition problem into multiple tasks and trains a neural net on each task [9]. In 2016, a model called BachBot used a type of recurrent neural network known as long-short term memory (LSTM) to automatically compose Bach chorales [8]. A similar approach was taken in [11]. This technique has been extended to encoder-decoder BiLSTM architectures for harmonization in [6], as well as variational latent variable methods for melodies in [13].

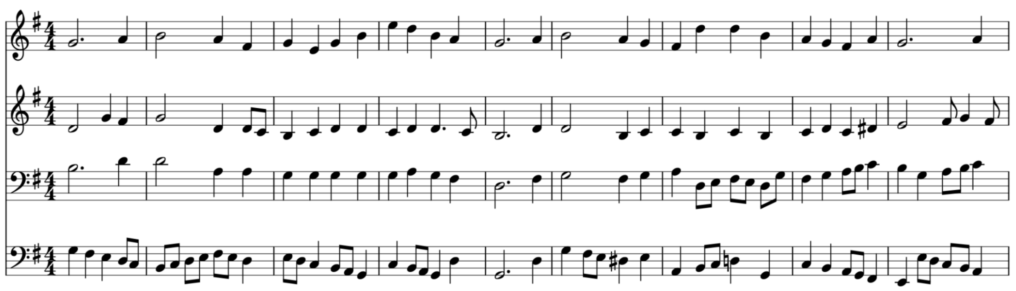

Figure 1: Example of a score sampled by DeepBach.

More recently, OpenAI tackled the music composition problem using transformers on raw audio waveforms [12]. Although very interesting, this approach has its limitations, as it precludes human performance. Therefore, we would focus on music modeling in the score-based domain, to generatively model symbolic representations of music. This has similarities to language modeling.

Many of these approaches have the drawback of not being flexible. For example, they may only be able to compose music sequentially, rather than ”fill in” or alter partially written music. Other methods have the drawback that they are too stringent, for example, only outputting music in the key of C, or in a pre-specified key by transposition.

This paper draws heavily from a programming languages (PL) inspired approach called DeepBach [8], where care is given to how the notes are represented. DeepBach encodes notes and rests per every sixteenth, and assumes a simple 4/4 time signature. It then constructs a distribution over those notes, jointly with a distribution over metadata like the placement of fermatas. The authors train neural networks that function as pseudo-Gibbs samplers, which can model the conditional distribution of individual harmonies at a given part of the chorale. See Fig. 1 for a representative sample of DeepBach's output.

The DeepBach approach is a simple and controllable autoregressive model for Bach chorale generation, which makes it easy to train and use. In particular, the Gibbs sampling approach is very flexible and can be adapted in interpretable ways. However, there are many instances where DeepBach is unable to capture long-term structure. Some casual listeners have remarked that the compositions “sound good but go nowhere.” This could be due to a combination of vanishing LSTM gradients, and pseudo-Gibbs sampling getting stuck in 1-optimal local minima.

Our contribution is to look in the direction of designing generative deep learning models for music that strongly *avoid local minima*, while *retaining controllability*.

2 Approach

We adopt a novel Langevin diffusion approach due to Song and Ermon in [16] that estimates gradients of the data distribution using a *noise conditional score network (NCSN)*. This has been shown to offer performance competitive with that of top GAN models on generative image modeling datasets like FFHQ and CelebA. Therefore, we have the potential to retain controllability thanks to the iterative Markov chain Monte Carlo procedure, while estimated gradients offer the possibility of escape from local minima by simultaneously optimizing the entire chorale as a sequence.

There are many deep learning approaches that have recently been applied to score-based music composition or other similar language modeling tasks.

- **Generative adversarial networks:** Although GANs acheive very promising results in modeling latent distributions of images, it's difficult to train them on sequence tasks (discrete tokens), as gradients need to propagate from the discrminator to the generator [20].

- **Transformers:** Transformers have been applied to the task of music generation and achieved state-of-the-art results on at least one dataset [10]. However, transformers are computationally expensive, so they're not easily controllable through masking and iterative MCMC-like algorithms.
- **Markov random fields:** MRFs have been used for generative models to optimize an energy function, notably for bitmap image generation in ConvChain. This lends credence to MCMC for discrete probabilistic modeling in the image domain, similar to DeepBach. However, as previously mentioned, it doesn't learn global structure. Also, the alternative approach of gradient ascent is impractical due to adversarial perturbations.

We also compare to baseline results from sequence models (LSTM and Transformer), which predict the conditional distribution of the notes at each timestep from the previous notes before it.

We adapt Song and Ermon's technique for sampling from a generative model. Their approach is based on the "score" of the data density, which is the gradient of the log data density. Since it is a gradient, it is a vector field pointing in the direction of greatest increase in density. Our model has two basic components:

1. **Score matching:** To model the score distribution $p(x)$ as a learned function of x. Although directly computing the data density is computationally intractable due to estimation of the partition function, we discuss in the next section a loss function for the score that can be optimized tractably.
2. **Sampling:** Once the score function has been estimated, a directed MCMC algorithm known as *stochastic gradient Langevin dynamics (SGLD)* is used to sample from the modeled data distribution. This combines gradient ascent on the score function with a certain amount of random noise, which speeds up convergence of the MCMC.

Unfortunately, as there may be little training data in low density regions, directly applying score estimation is a difficult task [16]. It could be more than an entire region: one could even imagine the score to be (at least approximately) low dimensional relative to the distribution space. These issues could have very adverse effects under Langevin dynamics, where it can be difficult to leave these regions, or to cross them back into high density regions.

Song and Ermon use two methods to deal with the curse of low dimensional data manifolds. One is to perturb the data with noise. By adding even a tiny amount of isotropic Gaussian noise, the data can less resemble a low dimensional hyperplane. The second approach is to use annealed Langevin dynamics. Under this system, the amount of Gaussian noise added at each Langevin update step is gradually decreased over the course of sampling, which improves mixing on multimodal distributions.

3 Encoding

Sticking to the programming language roots, we give much thought to the encoding of the chorales. This is critical since it affects the performance and expressivity of the learning algorithm, by introducing a useful inductive bias. We also do not want to assume too much domain-specific knowledge, as we want our approach to be adaptable to other problems. Generative music modeling, however, is inherently multidimensional: even the most compact notation of Bach chorales must be able to distinguish between many similar ideas in musical sequence data; we cannot lose information to the extent that our encoding could correspond to different musical ideas.

For the purposes of this project, we do not encode the key signatures or accidentals explicitly, nor do we handle the minority of pieces that are in time signatures other than $\begin{smallmatrix}4\\4\end{smallmatrix}$ common time. This suffices for a proof-of-concept given our short project timeline, and it also substantially simplifies the problem. We discretize the music into sixteenth notes. Each measure is split into 16 semiquavers, which each correspond to four input tokens each representing a note sung by a voice. Since Bach chorales, characteristic of their era, contain no smaller beat subdivisions, this encoding stays true to the character of the chorale. We write a chorale as

$$x_i = \{V_1, V_2, V_3, V_4\},$$

where $V_1, \ldots, V_4$ correspond to the soprano, bass, alto, and tenor voices, respectively. Then each $V_i = \{n_{i,t}\}$ where the ordered sequence has cardinality equal to the duration of the Bach chorale in semiquavers—typically about one minute, or 200–400 tokens.

We use the `music21` library [3] to perform the transformation of the Bach chorale into our encoding system, and to decode from our encryption system into legible sheet music. The `music21` library also provides a method that attempts to ascertain the key signature; in our experience, this algorithm is often faulty. However, a key signature can easily be deduced by a trained musician simply looking for patterns like the tonic triad, so we do not see this as a major issue.

The DeepBach paper further encodes and tracks metadata around the piece, such as the location of fermatas. We felt that using a fermata to signal the end of a phrase, while a useful inductive bias for the algorithm, may be out of scope given the time constraints of our project. Musically, the end of a phrase can be communicated through a sensible chord progression, called a *cadence*. A possible extension of our work would be to encode more metadata about the piece.

3.1 Notation

The Bach chorale dataset is represented as $\{x_i\}$, where each chorale is x_i is indexed by i. We say our data comes from a distribution with probability density function $p_{\text{data}}(x)$, which is unknown. Instead, we model our problem as having finitely many i.i.d. samples from this distribution, i.e., the original chorales that Bach wrote.

4 Score Matching

Given a probability distribution with density $p(x)$, we define the *score* function, following [16], to be equal to the gradient of the log-density. Mathematically, this can be written as $\nabla_x \log p_{\text{data}}(x)$. The objective in score matching is to find $s_\theta(x)$ such that

$$\mathbb{E}_{p_{\text{data}}(x)} \| s_\theta(x) - \nabla_x \log p_{\text{data}}(x) \|_2^2 \tag{1}$$

is minimized. Thus, $s_\theta(x)$ is favored if it is close to the true score function. Typically, we only have access to a limited number of samples, not to the actual distribution, so we do not know $p_{\text{data}}(x)$. The innovation in score matching is we never try to approximate $p_{\text{data}}(x)$ itself, but instead the score. We will see that score matching avoids estimating the intractable *partition function*, which is often seen in generative models such as [7].

Theorem 4.1. *We can write the score matching objective as attempting to minimize*

$$\mathbb{E}_{p_{data}(x)} \operatorname{Tr} \nabla_x s_\theta(x) + \frac{1}{2} \| s_\theta(x) \|_2^2, \tag{2}$$

plus some constant that does not depend on our parameters.

Proof. We do not provide a proof, but this theorem is a consequence of integration by parts. See the supplementary materials of [18] for the mathematical details. □

This updated loss is an expectation over p_{data}, which we can obtain by sampling from the training data, and it no longer contains the intractable $\nabla_x \log p_{\text{data}}(x)$ term. However, there are still some computational difficulties. Note that

$$\nabla_x s_\theta(x) = \nabla_x^2 \log p_{\text{data}}(x|\theta)$$

is the trace of the Hessian. The computation of this Hessian is costly, as it requires k backpropagation steps for a k-dimensional output, thus requiring quadratic time [18]. As we need to maintain flexibility in our high-dimensional data encoding, we need to apply a

Hessian-free workaround. One common method is to instead take *random projections* of the multivariate distribution, approximating $\operatorname{Tr}\nabla_x s_\theta(x)$ using

$$\mathbb{E}_{p_v} v^T \nabla_x s_\theta(x) v = \operatorname{Tr}\nabla_x s_\theta(x),$$

where p_v refers to the distribution of a multivariate standard normal $v \sim \mathcal{N}(0, I)$. Then we can also approximate our loss as

$$\mathbb{E}_{p_v}\mathbb{E}_{p_{\text{data}}(x)} v^T \nabla_x s_\theta(x) v + \frac{1}{2}\|s_\theta(x)\|_2^2. \tag{3}$$

Computing $v^T s_\theta(x) v$ can be done efficiently with forward mode auto differentiation, but still requires four times more computation than the approach detailed below [16].

The method we apply is called denoising score matching. The intuition here is to create a distribution $q_\sigma = p_{\text{data}} + \text{noise}$ for some noise that depends on σ. Then, for small enough σ such that $q_\sigma \approx p_{\text{data}}$, we have that $\nabla_x \log q_\sigma(x) \approx \nabla_x \log p_{\text{data}}(x)$. This brings us to denoising score matching:

Definition 4.2. *Denoising score matching* is a variant of score matching that finds s_θ to minimize

$$\mathbb{E}_{q_\sigma, p_{\text{data}}}\|s_\theta(\tilde{x}) - \nabla_x \log q_\sigma(\tilde{x}|x)\|_2^2. \tag{4}$$

Then the optimal $s_\theta(x) = \nabla_x \log q_\sigma(x)$ almost surely, though we may not find that $s_\theta(x)$. In particular, the above expectation is still hard to compute. We can adapt denoising score matching to be Gaussian noise conditional score matching: define $s(\tilde{x}, \sigma_i) = \nabla_x \log q_{\sigma_i}(x)$. Then we can use Adam's law (law of iterated expectation) to write the conditional denoising score objective as

$$\mathbb{E}_{p_{\text{data}}(x)}\mathbb{E}_{\tilde{x}\sim N(x,\sigma^2 I)}\|s(\tilde{x}, \sigma) - \nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x)\|^2.$$

Minimizing over s_σ gives $\nabla_x \log q_\sigma(\tilde{x})$ under mild conditions [18]. Since $q_\sigma(\tilde{x}|x)$ is well known from the PDF of the normal distribution, this computation is quite tractable. As $i \to \infty$, $\sigma_i \to 0$, and the optimal s approaches $\nabla_x p_{\text{data}}(x)$ as desired.

This provides us with the crux of our score matching argument:

Theorem 4.3. *The function $s_\theta(\tilde{x})$ that minimizes*

$$\mathbb{E}_{q_\sigma, p_{data}}\left\|\sigma s_\theta(\tilde{x}) - \frac{\tilde{x} - x}{\sigma}\right\|_2^2$$

will closely approximate $\nabla_x \log p_{data}(x)$.

We can write an approximation of this in closed form: The expectations are simply determined empirically, by taking draws of $\tilde{x}$, and using the data for x. Recall the strong law of large numbers: almost surely,

$$\lim_{n\to\infty}\frac{1}{n}(f(X_1) + \ldots + f(X_n)) = \mathbb{E}f(X).$$

The deep neural network that learns $s_\theta(x)$ via gradient descent is called a *noise-conditional score network*, and it can have any architecture that lends an appropriate inductive bias to the problem being modeled. For example, in previous papers, deep convolutional neural networks using inverted residual [15] and atrous spatial pyramid pooling [2] blocks were used as score networks for images. However, our problem is not a computer vision task, so we use a different architecture discussed in Section 6.

5 Langevin Dynamics

After estimating the score of distribution over chorales, it becomes necessary to draw from the estimated distribution. This is after all how composition works: a composer has a deep understanding of how music should be and is able to draw from this distribution. The algorithm already knows the score function, but it is non-trivial to draw samples from it.

Fortunately, there is an idea from physics that can be helpful. Langevin diffusion can be used to simulate a distribution given simply its score function. Formally,

Theorem 5.1. *As $t \to \infty$,*

$$X_t = X_{t-1} + \alpha s_\theta(X_{t-1}) + \sqrt{2\alpha} z_t$$

for $z_t \sim N(0, I)$, and small α, X_t converges to X drawn from the distribution of which s_θ is the score.

A proof is not provided, since this result is standard in the MCMC literature [14]. For reference, this can be written in continuous time as

$$\mathrm{d}X = \alpha s_\theta X + \sqrt{2\alpha}\, \mathrm{d}W.$$

The above is simply a discretized version of this Ito process. The α term controls the diffusion step size. It also controls the rate at which X is able to discover new areas of the support of the distribution, so we'd like it to be high at first and then to decay. It's analogous to the noise terms σ_i, so we'd expect it to be proportional. One compelling idea is to decay α geometrically [16]. For example, $\alpha \propto \sigma_i^2$, where σ_i decay geometrically. This proves effective in practice, since it results in a small step size relative to the score [16]. Since samples are from q_σ, which is perturbed, the data cannot lie in a low dimensional manifold. At early stages in training, a large step size means that it is hard for the data to reside in low probability regions. Earlier samples become more likely to land in high density regions. Song and Ermon apply a trick using Tweedie's formula: rather than outputting the final iteration x_T of that algorithm, they add a $\sigma^2 s_\theta(x_T)$ to the output [17].

6 NCSN Transformer Architecture

To model the learned score function $s_\theta(x)$, we need a neural network architecture that can learn to recognize common harmonies, rhythms, melodic structure, and motives from sequential data. Given a chorale of length N, our function is of the shape $s_\theta : \Sigma^{4N} \to \mathbb{R}^{4N|\Sigma|}$, where Σ is the set of tokens in our chorale representation.

There are several options for doing this sequence modeling. We choose to pass the learned music into a linear embedding layer, followed by an encoder from the *transformer* architecture [19]. This encoder combines multi-head self-attention, a positional embedding, layer normalization, and dropout steps. It makes up the bulk of our deep learning architecture, followed by a decoder layer that interprets the outputs as a score function s_θ.

The code for our noise-conditional score network, based on transformers, is made available in the repository at `langevin_music/network/ncsn.py`. To make this model work, we took into account other implementation details from the NCSNv2 paper ([17]), which suggests to train a single score gradient neural network, and scale the gradient by the standard deviation of the noise. Based on the maximum absolute deviation between chorales in our input dataset, we also chose to set the noise scales in a geometric sequence, with the largest noise scale being $\sigma_1 = 20$ and the smallest noise scale being $\sigma_{20} = 0.05$.

We trained the neural network on the Bach chorales dataset from `music21`. Since there were few chorales in the original dataset, we augmented it by transposing each chorale into different keys by semitones, as long as the ranges for each voice did not lie outside of a standard acceptable pitch range. Our empirically selected pitch ranges, which encompassed all but three outlier notes among Bach's 389 chorales, were C4 to G5 for soprano, F3 to D5 for alto, C3 to A4 for tenor, and C2 to E4 for bass. The justification for augmentation was to increase dataset diversity. By themselves, key signatures do not affect the sound of a piece, as pitch is heard relatively, and furthermore tuning in Bach's time was very different from standard tuning (A4 = 440 Hz) today. Dataset augmentation code can be found in `langevin_music/dataset/chorales.py`.

To minimize the score-based loss discussed in Section 4, we used the Adam optimizer with learning rate set to $\alpha = 10^{-3}$ on the autoregressive mean squared-error loss. All other parameters were left at their defaults. We use the *denoising score matching* algorithm described in Section 4, each time perturbing the input x by Gaussian noise of a random noise level and training for many iterations.

7 Results and Comparison

We present our results in this section. As mentioned earlier, we had some loss of performance from not implementing the full encoding scheme used in DeepBach, including rhythmic metadata, sequence masking, and fermatas, due to a short project timeline. We also did not have time to perform a grid search on hyperparameters. Our training was limited to what was tractable to compute on a self-funded AWS EC2 GPU instance and limited budget.

Despite the difficulties of training a deep learning project in such a short timeline, we were able to train several different neural network architectures, including character-level token modeling and score matching. The generated music is generally able to maintain a strong tonal center and some amount of rhythmic diversity.

7.1 Sequential LSTM

We implemented a standard LSTM sequence model for a baseline comparison. The data encoding is identical to how we train the Langevin music model. The LSTM predictor has a quite simple architecture, with an embedding layer, three LSTM hidden layers, and a log-softmax activation producing the final categorical logits. To prevent overfitting and add regularization, we add dropout with probability $p = 0.2$. We train the recurrent neural network using truncated backpropagation through time (BPTT) with sequence length 32 to predict the next 4-tuple of tokens in the stream.

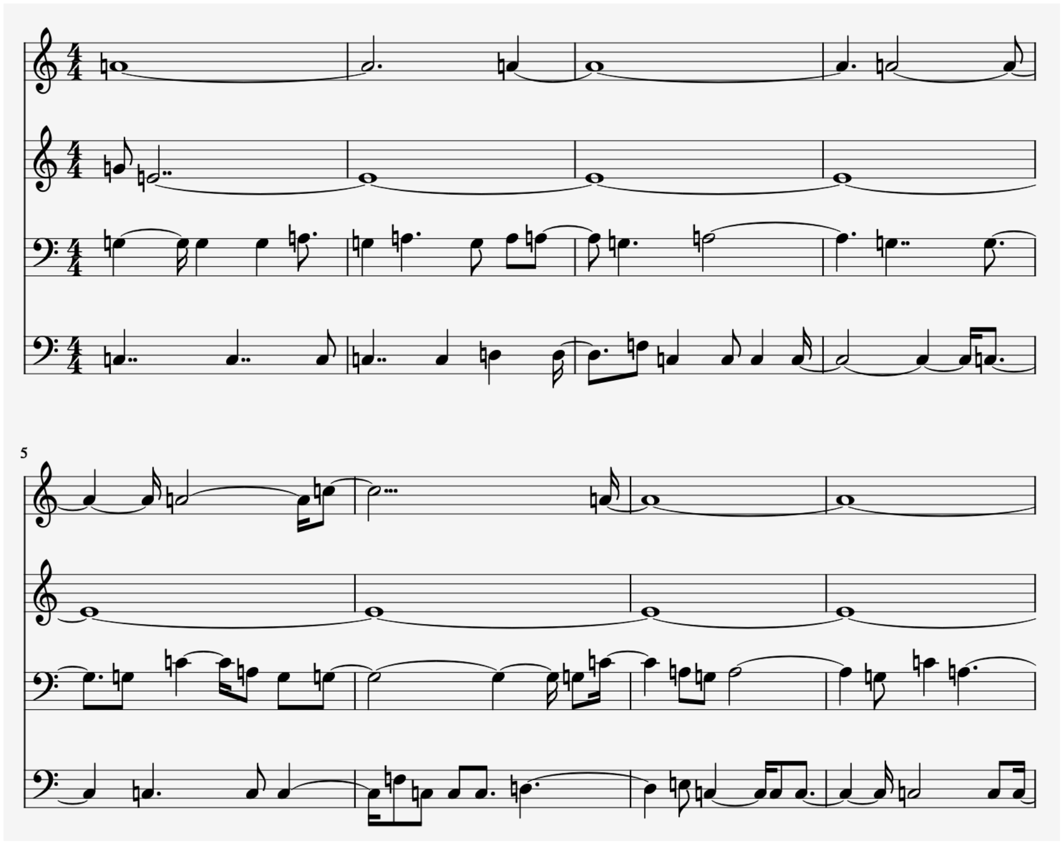

Figure 2: Eight measures generated from cold sampling the sequential LSTM.

The chorales generated by the sequential LSTM are not terribly interesting, but it still able to understand harmony to some extent. With cold sampling shown in Fig. 2, the token with the maximum log-probability is chosen at each step. This leads to some conservative behavior; the alto holds the same tied note for eight measures! We qualitatively evaluate this output as follows.

- **Harmonically:** The neural network has strongly grounded itself in the C major pentatonic mode. This is very interesting. The composition still sounds reasonably consonant since it's hard to introduce much dissonance just from the pentatonic scale. Note that we did not explicitly teach it anything about scales.
- **Rhythmically:** Unfortunately, the neural network is not very rhythmically grounded. Notes start at random, typically unaccented beats. This could likely be easily fixed by adding rhythmic metadata to the input encoding.
- **Melodically:** The melody is uninteresting, consisting mostly of tied notes.

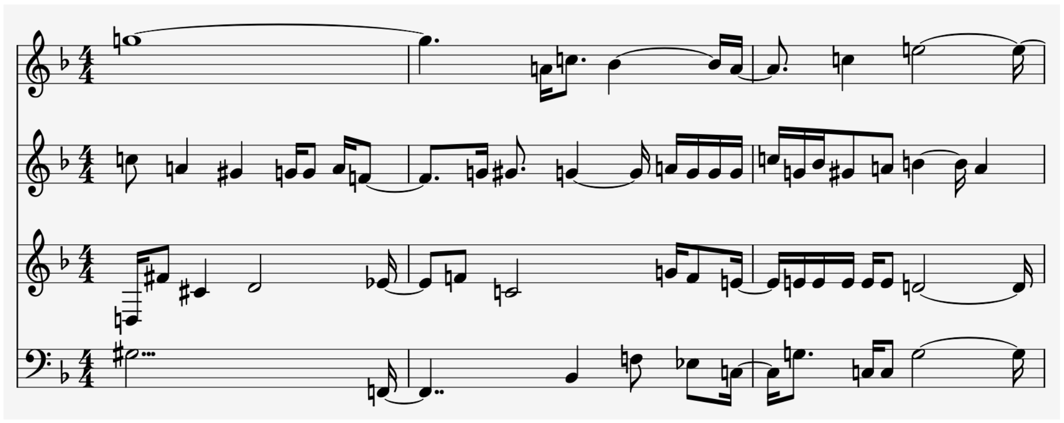

Figure 3: Three measures generated from warm sampling the sequential LSTM.

We saw similar results with cold sampling for the other models, with very little variance in the results, so we omit further discussion of cold sampling. Instead, we perform *warm* sampling (temperature $T = 1$), where each token is chosen with probability proportional to the output of the neural network. The warm sampling results for the LSTM model, shown in Fig. 3, are more varied and have some interesting ideas, but no coherent direction.

7.2 Sequential Transformer

For comparison purposes, we also implement an autoregressive transformer model that views the chorale as a sequence of four-tuples $\{n_{i,t}\}_{t=1}^{T}$, and models the conditional distribution of tuple $t+1$ from the notes at times $1, 2, \ldots, t$, using a transformer encoder. This is similar to the architecture described in Section 6, except we predict the next note directly, rather than an estimate of the data distribution gradient.

Using transformers to model sequences in this token-level manner has a history of use in the literature. For example, large autoregressive language models use a similar approach, modeling the conditional distribution of the next word in the sequence and sampling iteratively [1, 4]. The architecture of this network is similar to past work from the *Google Magenta* team [10]. However, our model is specialized to chorales rather than general MIDI piano rolls, to reduce training cost and provide a fairer comparison.

A representative score generated by the sequential transformer model is shown in Fig. 4. In comparison to the LSTM, the transformer is able to generate a far more convincing musical excerpt, despite the method of sampling (iterative, token-by-token) being identical. It has a good diversity in harmony, not just taking pitches from the pentatonic scale, and it is definitely in F major. Furthermore, there is an imperfect authentic cadence at the end of measure 7 that correctly resolves, which indicates that the model has learned how to end phrases, despite not being supervised with metadata about fermatas.

We should note that this generated excerpt tends to be a little weird in having note changes on unaccented beats. This kind of syncopation is not characteristic of Bach's music, and we believe that it is just an artifact of us not adding rhythmic metadata to the input chorales. For long compositions, the problem gets worse because the probability of the model

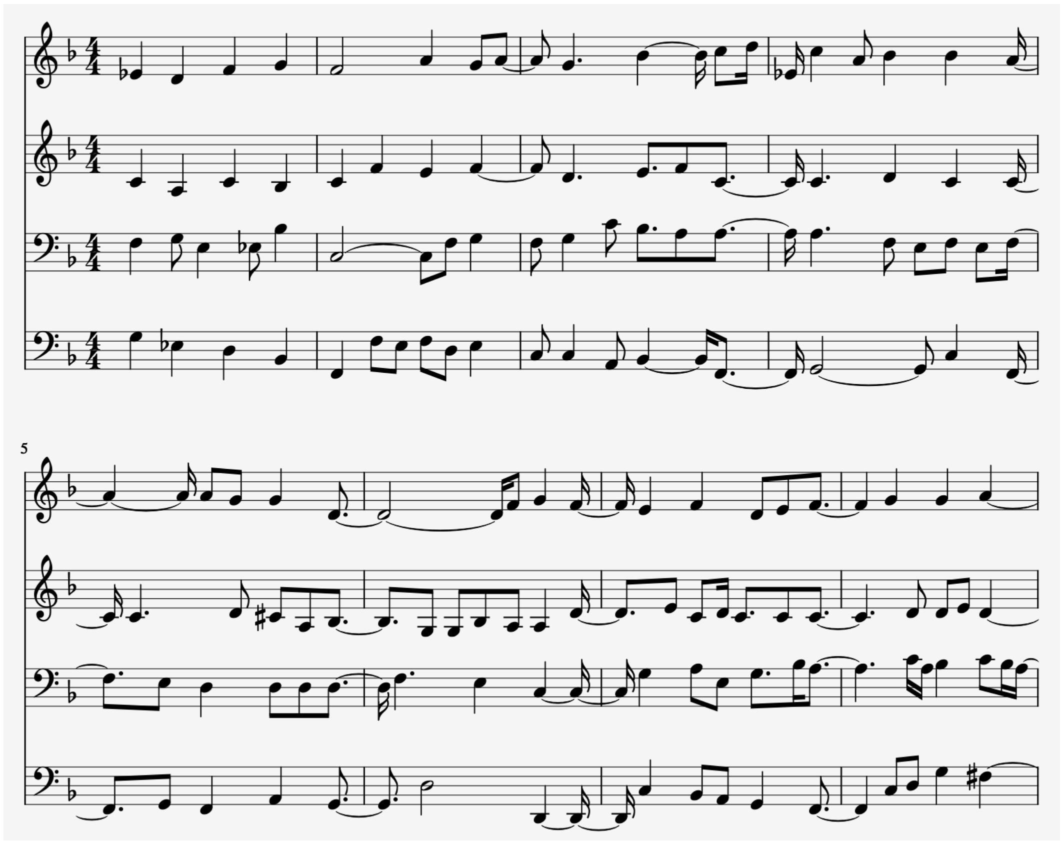

Figure 4: Eight measures generated from warm sampling the sequential transformer.

accidentally “slipping” into an offbeat increases, since we don’t provide supervision with the concept of a measure or beat. This speaks to the importance of having a metronomic, rhythmic grounding for highly-structured text data.

7.3 NCSN Transformer

The noise-conditional score network has a similar architecture to our sequential transformer, except we do not place the tokens in a categorical embedding. Instead, we do a one-hot map of tokens into orthogonal unit vectors. It is possible that using a learned or random embedding (as per the Johnson-Lindenstrauss lemma) could reduce the dimensionality of the problem and improve performance, but we did not test this approach.

Our sampling procedure is simply to start from a random unit vector $x \in \mathbb{R}^{4N|\Sigma|}$, distributed according to a multivariate normal, and perturb it by many steps of annealed Langevin dynamics. We gradually decreased the noise scale after each of T iterations. We also did a final “denoising” gradient descent step, as described previously.

The results of sampling the NCSN transformer model with annealed Langevin dynamics are shown in Fig. 5. Unfortunately, these results are overall not as convincing as the sequential transformer. The output music does indicate some amount of mixing, but generally, the global-optimizing Langevin dynamics system was not able to converge to a satisfying result.

We believe that the poor performance was due to the high-dimensional discrete nature of the problem space, and that this could be remedied by reducing the dimensionality with an initial embedding step for all of the pitches. We therefore do not immediately conclude that noise-conditional score matching *cannot* be applied to sequential modeling tasks like language or music. Rather, it may take additional work to overcome some of the challenges of dimensional sparsity, which we shed light on.

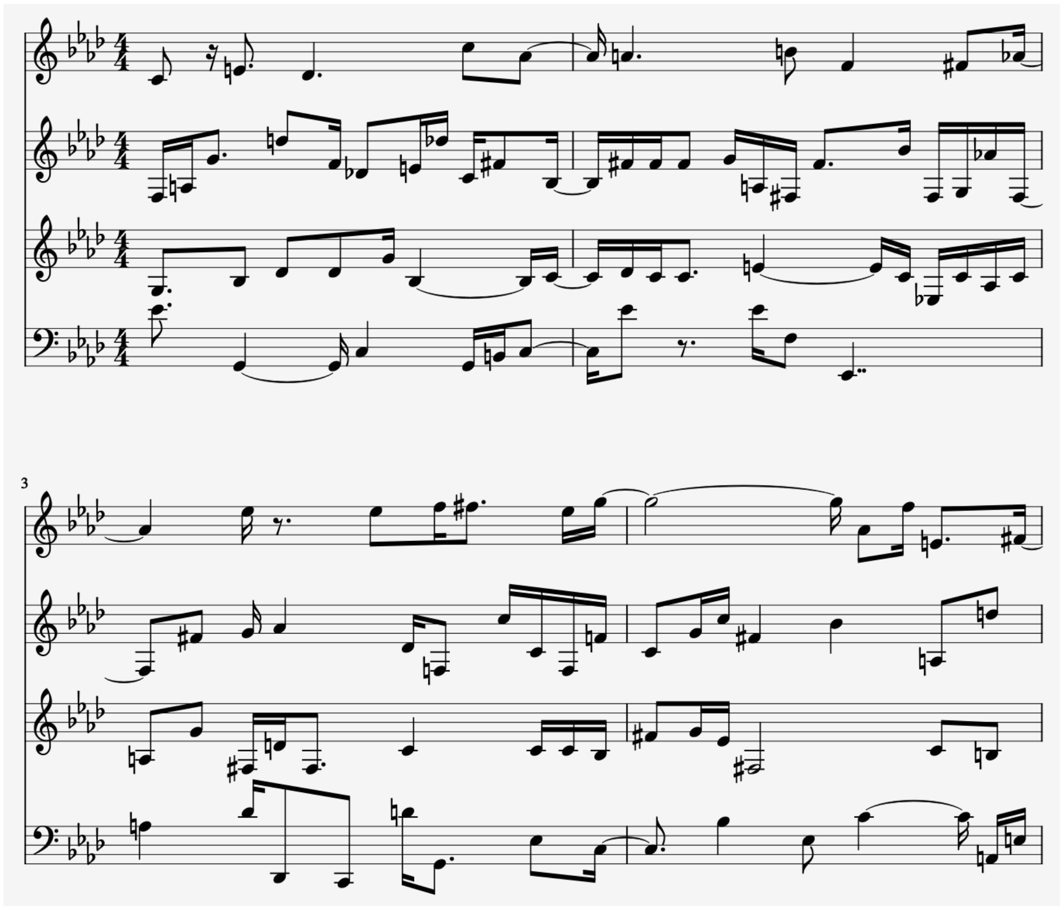

Figure 5: Four measures generated from sampling NCSN with annealed Langevin dynamics.

8 Future Work

Our stated goal in this project was to explore the previously unexplored space of noise conditional score matching for music generation and analyze its challenges.

We think that score matching would be a natural fit given the success of other MCMC methods such as DeepBach, and its success in other generative domains such as image synthesis. The recent trend in the state-of-the-art for the image modeling and language modeling fields has been converging on similar ideas, with each field borrowing architectures from the other. Symbolic music is a language, and in theory, MCMC methods offer an unparalleled level of interpretability and control, while the gradient estimation procedure allows for the learning of global structure such as repeated motives.

Our results indicate that noise conditional score matching requires additional tweaking to be able to successfully model token-based sequential data such as musical scores. A potentially fruitful avenue for future work would be to attempt to do score matching on a distribution of denser support, such as a learned embedding of musical pitches. It should also be relatively straightforward to incorporate additional metadata such as a rhythmic metronome to our models, which would improve performance.

In the interest of openness, all code for this project has been made publicly available on GitHub at `https://github.com/ekzhang/langevin-music`. This includes a simple and original PyTorch implementation of dataset preparation, sequential LSTM models with backpropagation through time, our music transformer model, and our NCSN transformer model with annealed Langevin dynamics. We hope that our code may be useful as a reference for future work in this area.

References

[1] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[2] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint 1706.05587*, 2017.

[3] Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, 2010.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Kemal Ebcioğlu. An expert system for harmonizing four-part chorales. *Computer Music Journal*, 12(3):43–51, 1988.

[6] Andrei Faitas, Synne Engdahl Baumann, Torgrim Rudland Næss, Jim Tørresen, and Charles Patrick Martin. Generating convincing harmony parts with simple long short-term memory networks. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 325–330. Universidade Federal do Rio Grande do Sul, 2019.

[7] Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. *arXiv preprint arXiv:1912.03263*, 2019.

[8] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. DeepBach: a steerable model for Bach chorales generation. *PLMR*, 2017. Code available.

[9] Hermann Hild, Johannes Feulner, and Wolfram Menzel. Harmonet: A neural net for harmonizing chorales in the style of js bach. In *Advances in neural information processing systems*, pages 267–274, 1992.

[10] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

[11] Feynman T Liang, Mark Gotham, Matthew Johnson, and Jamie Shotton. Automatic stylistic composition of bach chorales with deep lstm. In *ISMIR*, pages 449–456, 2017.

[12] Christine Payne Prafulla Dhariwal, Heewoo Jun. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2019.

[13] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. *arXiv preprint arXiv:1803.05428*, 2018.

[14] Gareth O Roberts, Richard L Tweedie, et al. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341–363, 1996.

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[16] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *NeurIPS*, 2019.

[17] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in Neural Information Processing Systems*, 33, 2020.

[18] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation. In *Uncertainty in Artificial Intelligence*, pages 574–584. PMLR, 2020.

[19] 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*, 30:5998–6008, 2017.

[20] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-first AAAI conference on artificial intelligence*, 2017.