Creating Musical Artifacts: Score-Based Generative Models for Music Generation

Henri UPTON
ENSAE 3A
henri.upton@ensae.fr

https://github.com/henriupton99/score-based-generative-models

Abstract

This work explores Score-Based Generative Modeling (SBGM), a new approach to generative modeling that uses the score function to estimate the gradient of the log-likelihood function. SBGM can provide advantages regarding other generative modeling approaches, such as Generative Adversarial Networks (Creswell et al.) and Variational Autoencoders (Estiri et al.). Among them, the ability to generate high-quality samples and to estimate the density of the data is crutial.

Based on SBGM and an algorithm for training such models, we explore the possibilities of music generation based on the MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) database.

To explore this framework, we rely heavily on the studies described in the article of Yang Song et al. (Yang Song, 2020)

1 Problem Framing

Score-Based Generative Modeling (SBGM) is based on the idea that the score function can be used to estimate the gradient of the log-likelihood function without explicitly computing it. The computational cost of computing the gradient of the log-likelihood function can be significant, particularly for high-dimensional data, making it crucial to consider (Yilun Du, 2018). The generation of new data based on a reference sample $x = \{x_1, \dots, x_N\}$ by an SBGM consists of a set of three main flexible steps:

- **Method for Score Estimation** : In practice, any neural network that maps an input vector $x \in \mathbb{R}^d$ to an output vector $y \in \mathbb{R}^d$ can be used as a score-based model, as long as the output and input have the same dimensionality.
- **Diffusion method to sequentially go from data to noise**. In our study framework, the noise dynamic is a stochastic process described by a stochastic differential equation (SDE) that describes the noise and its trajectory over the time interval $[0, T]$.
- **Analogous method to reverse the process and go from noise to data**. We reverse the process and start from noise sample $x(T)$ at time $t = T$ to new sample $x(0)$ at recovered time $t = 0$. As shown in Figure 1, it is during the second denoising phase that the need to estimate the score arises.

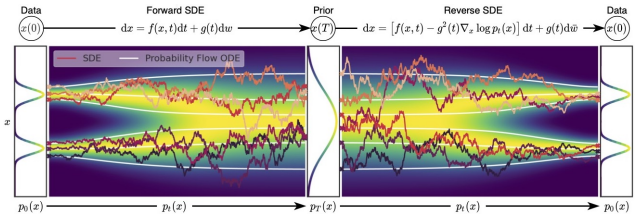

Figure 1: Steps for generating new samples on image data: perturbation of the initial image (Forward SDE) and denoising (Reverse SDE)

We will specifically detail the score function, the diffusion and denoising dynamics detailed by the blog post (Song, 2019) that serves as reference for our study.

1.1 Score based generative model

Score is the central notion of the generative model of the study. Given a probability density function $p(x)$ associated to our dataset $x = \{x_1, \dots, x_N\}$, it is defined as the gradient of the log-likelihood :

$$s(x) = \nabla_x \log p(x) \tag{1}$$

SBGM are then trained to estimate this quantity. Unlike likelihood-based models such as flow models or autoregressive models, score-based models

do not have to be normalized and are easier to parameterize. Indeed, the 'gradient shift' makes the model insensitive to normalization constants that do not depend on the data x.

1.2 From Data to Noise : Forward SDE

In order to generate samples with score-based models, we need to consider a diffusion process that corrupts data slowly into random noise. Scores will arise when we reverse this diffusion process for sample generation. (Yang Song, 2019)

A diffusion process is a stochastic process similar to Brownian motion. Let $\{\mathbf{x}(t)\}_{t=0}^{T} \in \mathbb{R}^d$ be a diffusion process, indexed by the continuous time variable $t \in [0, T]$. A diffusion process is governed by a stochastic differential equation (SDE), in the following form :

$$dx = f(x, t)dt + g(t)dW_t \tag{2}$$

where $\mathbf{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ is called the drift coefficient of the SDE, $g(t) \in \mathbb{R}$ is called the diffusion coefficient, and $\mathbf{W_t}$ represents the standard Brownian motion. Particles moving according to an SDE not only follows the deterministic drift $\mathbf{f}(\mathbf{x}, t)$, but are also affected by the random noise coming from $g(t)d\mathbf{W_t}$. From now on, we use $p_t(\mathbf{x})$ to denote the distribution of $\mathbf{x}(t)$.

For score-based generative modeling, we will choose a diffusion process such that $\mathbf{x}(0) \sim p_0$, and $\mathbf{x}(T) \sim p_T$. Here p_0 is the data distribution where we have a dataset of i.i.d. samples, and p_T is the prior distribution that has a tractable form and easy to sample from. The noise perturbation by the diffusion process is large enough to ensure p_T does not depend on p_0.

1.3 From Noise to Data : Reverse SDE

We can use the time-dependent score function $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ to construct the reverse-time SDE, and then solve it numerically to obtain samples from p_0 using samples from a prior distribution p_T. We can train a time-dependent score-based model $s_\theta(\mathbf{x}, t)$ to approximate $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, using the following weighted sum of denoising score matching objectives:

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]dt + g(t)d\bar{W}_t \tag{3}$$

where $\bar{\mathbf{W}}_\mathbf{t}$ is a Brownian motion in the reverse time direction, and dt represents an infinitesimal negative time step. This reverse SDE can be computed once we know the drift and diffusion coefficients of the forward SDE, as well as the score of $p_t(\mathbf{x})$ for each $t \in [0, T]$.

2 Practical Case : Music Generation

Although the paper under review as well as most of the literature is based on practical cases of image data, SBGMs can be applied to broader domains like music generation (Prafulla Dhariwal, 2020). In this paper, we focus on classical music generation. For this purpose, we rely on the MAESTRO Dataset (TensorFlow, 2022).

2.1 MAESTRO Dataset

The MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) dataset is a large-scale dataset of piano performances. It was designed to support research on multi-modal music performance analysis and generation.

The dataset contains over 200 hours of aligned MIDI and audio recordings of classical piano performances, which were obtained from publicly available sources. The performances were selected to cover a wide range of styles, composers, and pianists, and were manually aligned to ensure accurate synchronization between the MIDI and audio data. (Hawthorne et al., 2019)

2.2 Piano Roll Matrix as Features

A music can be perceived as a set of played notes which are characterized for each of them by a start date and an end date (we omit the velocity of the notes for this work). Thus, it is possible to extract from a track (and its metadata in .midi format) its piano roll matrix: by splitting the duration of the music into regular time intervals (according to a frequency f_s), each sample can be represented by a sparse matrix S of shape $(duration\_track \times f_s, num\_pitches)$ defined as follows : for all $j \in (1, \ldots, num\_pitches)$, $S_{t,j} = 1$ if note of pitch j is played in time interval $[t, t+1]$, and 0 otherwise.

Figure 2 shows an example of the piano roll matrix for an arbitrary track of the database. Colored parts represent the notes that are played in time (color variations are for velocity of the note, element that we omit for the generation part).

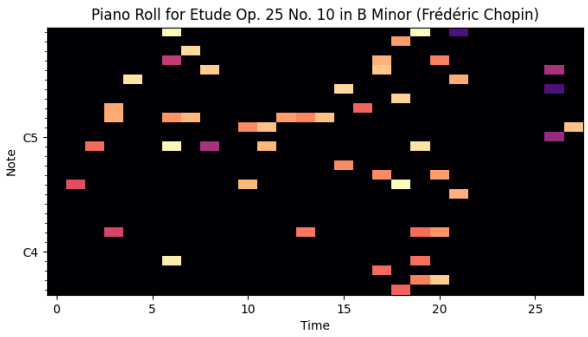

Figure 2: Piano Roll Matrix visualisation for a random sample of MAESTRO Dataset

3 Experiments Protocol

3.1 Data Preprocessing

For the purpose of training the model, it is necessary that the samples given as input are of the same shape. Thus, we set up preprocessing similar to padding to fix the shape of the piano roll matrices:

Preprocessing Hyperparameter	Value
lowest note pitch	56
highest note pitch	84
frequency f_s	1
track duration	28s

Figure 3: Set of Preprocessing Hyperparameters

3.2 Training Phase

For the training phase of the SBGM, we start from the structure of the Yang Song et al. paper by using the U-net model structure (Olaf Ronneberger, 2015) to train the score function. We use the usual baseline of model training under Pytorch with a number of epochs of batch observations to optimize the model parameters by back propagation of a cost function. As the number of parameters of a U-net is very large, we use the Adam optimizer (Adaptive Moments) which converges faster than the Stochastic Gradient Descent (SGD) method.

Figure 4 presents an exhaustive list of the selected initial training hyperparameters. They can be modified for hyperparameter tuning purposes explained in a very future section.

3.3 Sampling Phase

The paper by Yang Song et al. presents also three sampling methods that can be implemented in their SBGM: Euler-Maruyama sampler (PyMC3, 2018), Predictor-Corrector sampler (Sampler, 1998) and ODE sampler (ODE, 2023). We repeat an implementation of these three sampling methods to compare the results obtained.

Training Hyperparameter	Value
batch size	32
number of epochs	50
learning rate	10^{-4}

Figure 4: Set of Training Hyperparameters

3.4 Overfitting Detection

Several generic deep learning methods help to reduce the risk of overfitting, i.e. when the model performs very well (in terms of loss) on the training set, but generalizes very poorly when it processes new datapoints:

- **Train/Validation split**: the training data is themselves split into two parts. The train set data is used to train the model and the validation set data is used to test the overfitting during the training. Accross the epochs, when the loss on the validation set increases, the model generalizes less and less well and the training must be stopped.
- **Cross Validation** : as an extension of the first principle, it is more common in practice to perform a number K of different train/validation splits to train the model. Many modules on Python such as *Sklearn* offer tools such as *KFold* cross validation.
- **Early stopping** : in addition to the train/validation split method, it is possible to set up a *scheduler* that allows to stop the training when the error on the validation set starts to increase. Pytorch offers many simple and powerful tools to implement this method.

3.5 Hyperparameters Tuning

In addition to the methods that prevent overfitting during training, other tools allow to implement hyperparameters tuning. The most common is to implement *GridSearch*, which consists in defining discrete sets or intervals of values for each hyperparameter of the model, then testing a large set of possible combinations in order to find the composition that leads to the lowest model error. In our case study, it is possible to do hyperparameters tuning on two sets of elements:

- **Structural parameters of the model** such as the sigma of the diffusion method.
- **Model optimization parameters** such as batch size and learning rate.

Many papers have discussed the importance of this refinement phase during training, and introduce extensions to GridSearch such as *RandomSearch* or *Latin hypercube sampling*.

4 Results

Compiling and launching all the scripts available on the GitHub of this article allows to train an SBGM, to generate new musical samples according to the three sampling methods presented previously, and finally to present the piano roll matrix associated with these samples (Figure 6).

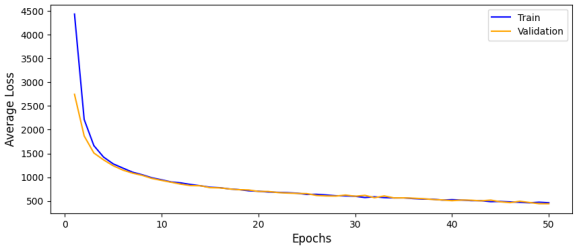

Figure 5: Training (blue) and Validation (orange) Losses values accross epochs for $N = 50$ epochs

Here are SoundCloud links to listen generated samples from the three sampling methods :

Prediction-Correction Sampler :
https://on.soundcloud.com/m6Zau

ODE Sampler :
https://on.soundcloud.com/UrBWF

Euler Maruyama Sampler :
https://on.soundcloud.com/4ZQyd

5 Conclusion

This study allowed us to have a first approach of the possibilities of Score Based Generative Models (SBGM) in the original framework of music samples generation. Many extensions can be proposed in continuity of this study:

- **Implement data augmentation to improve the robustness of overfitting detection** : contrary to some domains, it is difficult for the experimental music framework to have a large data set. Here the database consists of about 1200 training samples and 200 validation samples. Perhaps the implementation of data augmentation would be judicious to improve the capacities of the model according to the uses.

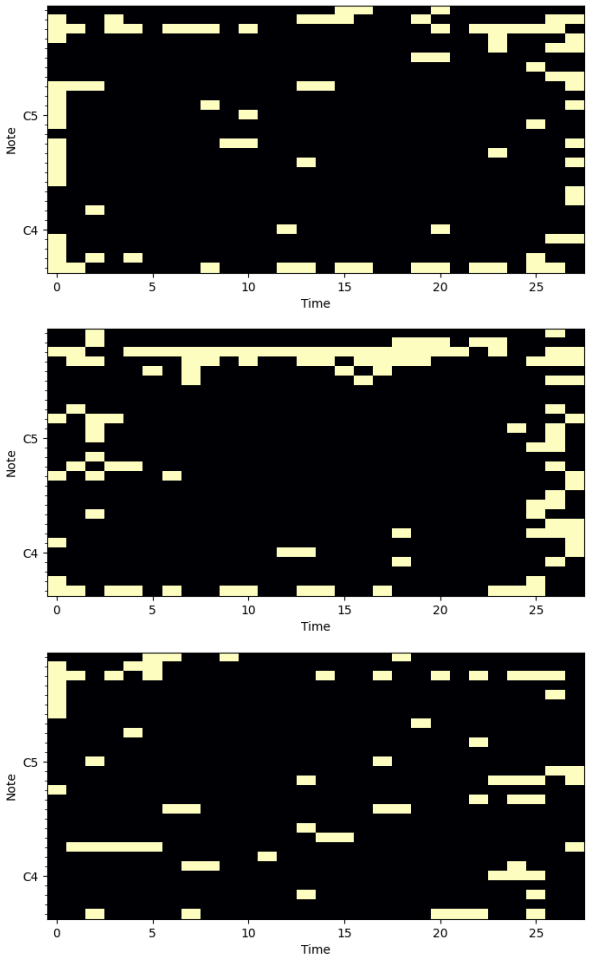

Figure 6: Piano Roll Matrix obtained with generated samples from three sampling methods : Euler Maruyama (1), PC Sampler (2), ODE Sampler (3)

- **Explore the possibilities of multi-instrumental music composition** : with an adequate dataset, it would be possible to compose tracks with the possibility of combining various instruments in addition to the classical piano. The Python library *pretty_midi* gives access to other instruments, so we need to have reference data of the desired instruments.
- **Testing the spectrum of reconstruction of altered data** : an exciting use case for SBGMs (and generative models more broadly) would be to reconstruct instants of musical samples that are altered or unavailable. More broadly, this could be applied to many domains that have sound data.

References

Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative Adversarial Networks: An Overview. *IEEE Signal Processing Magazine*.

Amir Hossein Estiri, Mohammad Reza Sabramooz, Ali Banaei, Amir Hossein Dehghan, Benyamin Jamialahmadi, and Mahdi Jafari Siavoshani. A Variational Auto-Encoder Approach for Image Transmission in Noisy Channel.

MIT PC Sampler. 1998. Predictor-Corrector Methods.

Thomas Brox Olaf Ronneberger, Philipp Fischer. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation.

Igor Mordatch Yilun Du. 2018. Implicit Generation and Generalization in Energy-Based Models. *arXiv:1903.08689*.

PyMC3. 2018. Inferring parameters of SDEs using a Euler-Maruyama scheme.

Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. 2019. Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset.

Stefano Ermon Yang Song. 2019. Generative Modeling by Estimating Gradients of the Data Distribution.

Yang Song. 2019. Score-Based Generative Modeling through Stochastic Differential Equations (Blog Post).

Christine Payne Jong Wook Kim Alec Radford Ilya Sutskever Prafulla Dhariwal, Heewoo Jun. 2020. Jukebox: A Generative Model for Music.

Diederik P Kingma Abhishek Kumar Stefano Ermon Ben Poole Yang Song, Jascha Sohl-Dickstein. 2020. Score-Based Generative Modeling through Stochastic Differential Equations.

TensorFlow. 2022. MAESTRO Dataset v3 from TensorFlow Magenta.

Scipy Intergrate ODE. 2023. Scipy Module integrate.ode Documentation.