IMPROVING SYNTHESIZER PROGRAMMING FROM VARIATIONAL AUTOENCODERS LATENT SPACE

Gwendal Le Vaillant *
Numediart Institute
University of Mons
Mons, Belgium
glevaillant@he2b.be

Thierry Dutoit
Numediart Institute
University of Mons
Mons, Belgium
thierry.dutoit@umons.ac.be

Sébastien Dekeyser
ISIB
HE2B
Brussels, Belgium
sdekeyser@etu.he2b.be

ABSTRACT

Deep neural networks have been recently applied to the task of automatic synthesizer programming, i.e., finding optimal values of sound synthesis parameters in order to reproduce a given input sound. This paper focuses on generative models, which can infer parameters as well as generate new sets of parameters or perform smooth morphing effects between sounds.

We introduce new models to ensure scalability and to increase performance by using heterogeneous representations of parameters as numerical and categorical random variables. Moreover, a spectral variational autoencoder architecture with multi-channel input is proposed in order to improve inference of parameters related to the pitch and intensity of input sounds.

Model performance was evaluated according to several criteria such as parameters estimation error and audio reconstruction accuracy. Training and evaluation were performed using a 30k presets dataset which is published with this paper. They demonstrate significant improvements in terms of parameter inference and audio accuracy and show that presented models can be used with subsets or full sets of synthesizer parameters.

1. INTRODUCTION

Automatic synthesizer programming consists in transforming sounds into synthesizer presets, which can be defined as sets of values of all synthesis parameters. While the task of finding an optimal preset can be fulfilled manually, it is often cumbersome and requires expert skills because a lot of synthesizers provide dozens to hundreds of controls. This topic is an active research field [1, 2, 3, 4] and models based on neural networks have shown considerable improvements.

This paper focuses on models which can program synthesizers from audio and generate new presets as well, such as the Flow Synth model [3]. An improved architecture, based on a Variational Autoencoder (VAE) neural network for audio reconstruction and on an additional decoder for synthesizer parameters inference, is introduced in Section 3. In contrast to previous generative models, this architecture is scalable and is able to handle all parameters of a given synthesizer. Experiments were conducted on the DX7 Frequency Modulation (FM) synthesizer and required a large dataset. More than 30k human-made DX7 presets were gathered and the curated dataset has been made publicly available.

Section 4 introduces a novel convolutional encoder-decoder structure which is able to extract latent features related to variations of pitch and intensity from multi-channel input spectrograms. This aspect was neglected in similar projects, which focused on learning how to reproduce sounds corresponding to a single note. General results and usability of the final architecture are discussed in Section 5.

* This work was also supported by IRISIB and HE2B-ISIB (Brussels, Belgium)

2. STATE OF THE ART

2.1. Variational Autoencoder

2.1.1. Original formulation

VAEs [5] are deep latent variable models which learn mappings between a space of observed data $\mathbf{x} \in \mathbb{R}^E$ (e.g., audio samples or spectrograms) and a latent space of vectors $\mathbf{z} \in \mathbb{R}^D$ with $D \ll E$. They are built upon an encoding model $p_\theta(\mathbf{z}|\mathbf{x})$ and a decoding model $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})\, p_\theta(\mathbf{z})$ parameterized by θ.

The latent prior $p_\theta(\mathbf{z})$ is most often defined as $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I_D)$. For audio spectrograms modeling applications, $p_\theta(\mathbf{x}|\mathbf{z})$ is usually a free-mean, fixed-variance multivariate Gaussian distribution [6]. Those means $\hat{\mathbf{x}}$ are outputs of a decoder neural network:

$$\hat{\mathbf{x}} = \text{DecoderNeuralNetwork}(\mathbf{z}; \theta) \tag{1}$$

Given the above assumptions, $p_\theta(\mathbf{x}, \mathbf{z})$ is easy to compute and to optimize using Stochastic Gradient Descent (SGD). It is considered to be a generative model because it allows drawing realistic samples $\mathbf{x}$ from latent codes $\mathbf{z}$.

However, the latent posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ is intractable [5] thus impossible to optimize. The original VAE formulation proposes to approximate $p_\theta(\mathbf{z}|\mathbf{x})$ with a parametric model $q_\phi(\mathbf{z}|\mathbf{x})$ such as:

$$\begin{cases} q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu_0, \sigma_0^2) \\ \mu_0, \log \sigma_0^2 = \text{EncoderNeuralNetwork}(\mathbf{x}; \phi) \end{cases} \tag{2}$$

where $\sigma_0^2 \in \mathbb{R}_+^D$ contains diagonal coefficients of the diagonal covariance matrix. From these models, the exact log-probability of a dataset observation $\mathbf{x_n}$ under the marginal distribution $p_\theta(\mathbf{x})$ cannot be computed nor optimized. Nonetheless, we can maximize the following Evidence Lower-Bound (ELBO):

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]}_{\text{reconstruction accuracy}} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]}_{\text{regularization term}} \quad (3)$$

with $\log p_\theta(\mathbf{x}) \geq \mathcal{L}_{\theta,\phi}(\mathbf{x})$ [5]. The first term is considered to be a reconstruction accuracy because it measures the log-probability of samples generated from encoded latent vectors. The second term is equal to $-D_{KL}\left[q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})\right]$ thus encourages encoded probability distributions to remain close to the prior $p_\theta(\mathbf{z})$.

In practice, these expectations can be computed using a one-sample Monte-Carlo approximation [5], i.e., a single $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. However, for gradients to be back-propagated through the encoder neural network, $\mathbf{z}$ must be reparameterized as $\mathbf{z} = \mu_0 + \sigma_0 \odot \epsilon$ where $\epsilon \sim \mathcal{N}(0, I_D)$. Finally, $\mathcal{L}_{\theta,\phi}(\mathbf{x}_n)$ values are estimated using a minibatch of dataset samples $\mathbf{x}_n$ and (θ, ϕ) model parameters can be optimized using minibatch SGD.

2.1.2. Latent space properties

The encoder introduces a data bottleneck and is encouraged to compress observations $\mathbf{x}$ while allowing accurate reconstructions $\widehat{\mathbf{x}}$. Therefore, latent distributions $q_\phi(\mathbf{z}|\mathbf{x})$ can be expected to hold meaningful information about input data, e.g., amplitude, timbre or envelope features when $\mathbf{x}$ is an audio spectrogram. Thanks to the regularization term (Equation 3), the latent encoding can also be expected to be continuous, i.e., similar $\mathbf{x}_n, \mathbf{x}_m$ observations should be encoded as $q_\phi(\mathbf{z}|\mathbf{x}_n), q_\phi(\mathbf{z}|\mathbf{x}_m)$ densities with similar joint support.

However, Zhao et al. [7] proved that ELBO optimization favors fitting the observations $\mathbf{x}_n$ over performing correct $\mathbf{z}$ inference, mainly because the reconstruction term (Equation 3) encourages encoded probability densities to have a disjoint support. This phenomenon is further amplified by the usually very large difference between $\mathbf{x}$ and $\mathbf{z}$ spaces dimensions ($E \gg D$). E.g., whereas $D_{KL}\left[q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})\right]$ is a sum of D terms (assuming Gaussian distributions), $\log p_\theta(\mathbf{x}|\mathbf{z})$ is a sum of E negative terms if components of $\mathbf{x}$ are modeled as independent. The regularization loss becomes orders of magnitude smaller than the reconstruction loss.

To prevent the aforementioned issues, the β-VAE formulation [8] applies a $\beta \gg 1$ factor to the regularization term. This hyperparameter also leads to a lower entanglement of latent variables, at the expense of blurrier reconstructions $\widehat{\mathbf{x}}$.

2.2. Normalizing Flows

2.2.1. Bijective probability distributions transforms

Normalizing Flows [9] are invertible neural networks primarily intended to transform probability distributions. Let $\mathbf{z}_0, \mathbf{z}_1$ denote random vectors and $T : \mathbf{z}_0 \mapsto T(\mathbf{z}_0) = \mathbf{z}_1$ denote an invertible and differentiable transform whose inverse is also differentiable. Given $q_{Z_0}(\mathbf{z}_0)$, the probability density of $\mathbf{z}_1$ can be obtained by a change of variables:

$$q_{Z_1}(\mathbf{z}_1) = q_{Z_0}(\mathbf{z}_0)\left|\det J_T(\mathbf{z}_0)\right|^{-1} \quad (4)$$

where J_T is the Jacobian matrix of the flow transform T, whose determinant should be easy and efficient to compute. In practice, most flow transforms [9, 10, 11, 12] have lower- or upper-triangular Jacobian matrices and implementations ensure non-zero diagonal coefficients for numerical stability.

2.2.2. Latent space normalizing flows

The baseline VAE approximates the true posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ with a simplified parametric model $q_\phi(\mathbf{z}|\mathbf{x})$ such as a multivariate Gaussian. Although it can lead to good reconstruction results, this posterior approximation can be improved by using a more flexible model [9], e.g., the output of a sequence of normalizing flows $T_1, ..., T_K$.

Let ψ denote the parameters of the full latent flow transform $T = T_K \circ ... \circ T_1$ and $\mathbf{z}_{k+1} = T_{k+1}(\mathbf{z}_k)$ denote successive latent vectors. $\mathbf{z}_0$ is the "$\mathbf{z}$" variable in Equation 2 and its closed-form density is parameterized by the encoder neural network. The deep latent variable of Equations 1 and 3 becomes $\mathbf{z}_K$. Thanks to the deterministic relationship between $\mathbf{z}_K$ and $\mathbf{z}_0$, the expectation from Equation 3 can be written with respect to $\mathbf{z}_0 \sim q_\phi(\mathbf{z}_0|\mathbf{x})$. Using Equation 4, the ELBO with a latent normalizing flow becomes:

$$\mathcal{L}_{\theta,\phi,\psi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}_0|\mathbf{x})}\Bigg[\log p_\theta(\mathbf{x}|\mathbf{z}_K) + \log p_\theta(\mathbf{z}_K) - \log q_\phi(\mathbf{z}_0|\mathbf{x}) + \sum_{k=1}^{K}\log\left|\det J_{T_k}(\mathbf{z}_{k-1})\right|\Bigg] \quad (5)$$

2.2.3. Flow models

Two common flow models are Masked Auto-regressive Flows (MAF) [11] and Real-valued Non-Volume Preserving flows (RealNVP) [12]. On one hand, a single MAF layer is fully auto-regressive, i.e., it introduces dependencies between all components of an output vector, and can be considered as a universal approximator [13]. Thus, a multi-layer MAF-based model has a large expressive power. However, an MAF inverse computation is sequential and approximately D times slower than a forward computation. On the other hand, a single RealNVP layer is a *coupling layer*. Half of the outputs are set equal to the first half of the inputs, while an affine transformation is applied to the other half of inputs. The scale and offset coefficients are parameterized by the unmodified first half of the inputs and the layer's own parameters. Nonetheless, a sequence of RealNVP layers and permutations forms a fully auto-regressive model. A RealNVP-based model provides more scalability – at the expense of a lower parameter efficiency. Moreover, the forward and inverse directions require the same amount of computation and are equally numerically stable [12].

2.3. Automatic synthesizer programming

2.3.1. From audio to synthesizer parameters

The main goal of automatic synthesizer programming [1] is to find optimal values of synthesis parameters so that synthesized sounds gets as close as possible to target input sounds. Applications range from transferring a sound from a synthesizer to another, to reproducing natural sounds (voice, acoustic instrument, ...) from a synthesizer. Early works used CPU-intensive genetic algorithms [14], and more recent techniques such as long short-term memory neural networks [1] and Convolutional Neural Networks (CNN) com-

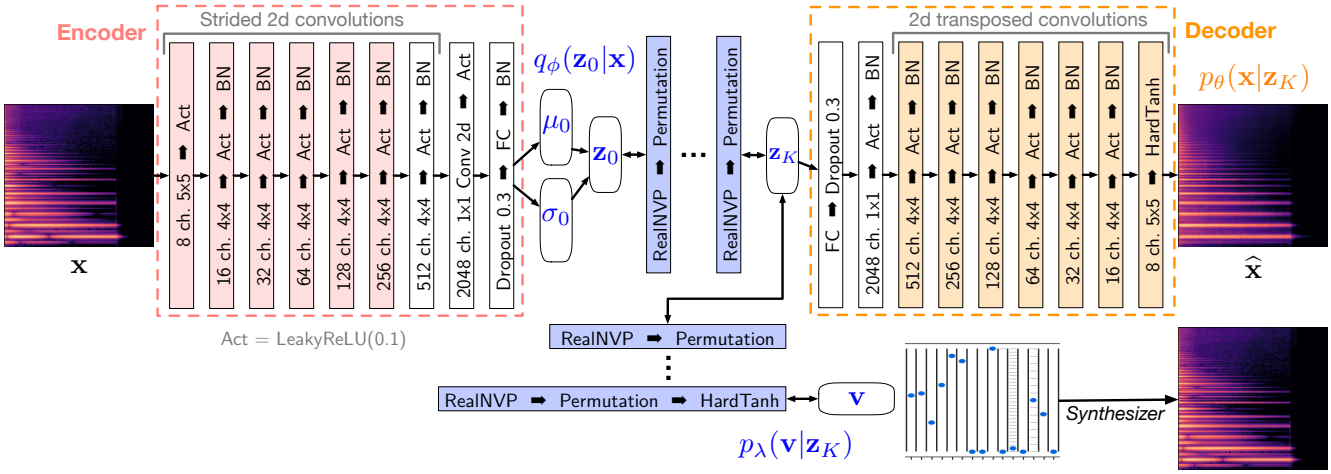

Figure 1: *Block diagram of our single-channel spectrogram model (general architecture inspired from [3]). The number of channels of encoding (resp. decoding) convolutional layers denote the number of output (resp. input) channels.*

bined with Multi-Layer Perceptrons (MLP) [2] have demonstrated improving performances.

These models directly infer synthesizer presets from Mel-Frequency Cepstral Coefficients (MFCC) [1], audio spectrograms or raw waveforms [2]. They do not rely on a perceptually regularized, continuous latent space from which new samples can be generated. Hence, we will qualify them as *non-generative*. A comprehensive literature review about non-generative synthesizer programming is available in the SpiegeLib paper [4].

2.3.2. *Differentiable sound synthesis*

One very interesting approach to this automatic programming problem is the recent Differentiable DSP model [15] which produces sound using oscillators, filters and noise sources implemented as neural networks. This model learns how to synthesize sound instead of learning how to program an external synthesizer.

However, in contrast to such fully-differentiable solutions, the perceptual and interactive qualities of usual software synthesizers have been studied and improved for a long time. They also provide non-differentiable synthesis methods (filters, oscillators, ...) which give them unique sonic properties. Moreover, their source code is carefully optimized because a low CPU-footprint is necessary for polyphonic and real-time usage. Therefore, only widespread synthesizers such as VST-format plugins are considered in this paper.

2.3.3. *Generative models*

Recently, Esling et al. [3] introduced the Flow Synth architecture, based on a spectral VAE. Instead of using a single-path neural network (e.g., MLP, CNN) from audio spectrograms $\mathbf{x}$ to synthesizer presets $\mathbf{v}$, they proposed to add a neural network which infers presets from latent codes of a VAE. This approach assumes that the latent space holds enough meaningful audio information.

They compared some feedforward models to several VAE-based models on limited subsets of numerical parameters (16, 32 and 64) of a sound synthesizer named *Diva*. Reported test measurements included errors on inferred presets $\hat{\mathbf{v}}$ and distances between target spectrograms $\mathbf{x}$ and audio spectrograms synthesized from $\hat{\mathbf{v}}$. First, results showed that their VAE-based models improved the spectral error on synthesized audio, although error on inferred presets $\hat{\mathbf{v}}$ was smaller with feedforward baseline models. Second, comparative results about the different preset regression networks of VAE-based models were presented. In terms of synthesized audio accuracy, they demonstrated that the usual MLP regression networks underperformed compared to flow-based ones. Third, they demonstrated through examples how new presets can be generated from exploring the neighborhood of a latent vector. Moreover, thanks to the invertibility of the regression flow, any preset could be easily encoded into the audio latent space.

There is nevertheless room for improvement. As the authors admit, the performance of their model decreases as the number of synthesizer parameters increases. This issue must be addressed because it prevents the model from using all sonic possibilities of a given synthesizer. Regarding audio reconstruction and presets generation, reducing inference errors would generally improve the model. Moreover, the architecture has been validated on a single subtractive synthesizer, hence different tests would be interesting. Categorical parameters, which were excluded from the aforementioned experiments, should also be considered because they make up for a large part of synthesizer functionalities.

3. SPECTRAL VARIATIONAL AUTOENCODER AND SYNTHESIZER PARAMETERS REGRESSION

3.1. Model

3.1.1. *General architecture*

As previously stated, we focus on a VAE structure similar to Flow Synth [3], which integrates a regression flow to infer synthesizer parameters. The detailed architecture is presented in Figure 1.

The CNN-based encoder and decoder are symmetrical. In order to reduce the high computational cost of convolutions, stride is (2, 2) for convolutional layers whose kernel is larger than one.

3.1.2. Synthesizer parameters regression flow

Considering the very promising results of Esling et al. [3] on subsets of parameters, an invertible transform $\mathbf{v} = U(\mathbf{z}_K)$ is optimized. This regression model requires the dimension D of latent vectors $\mathbf{z}_0, ..., \mathbf{z}_K$ to be that of the synthesizer parameters vector $\mathbf{v}$. U is made of L invertible flow layers i.e. $U = U_L \circ ... \circ U_1$ and λ denotes their parameters. While previous works [3] used an MAF variant [10] as regression flow, we use a RealNVP model in order to ensure scalability and to help improve training stability.

To infer $\mathbf{v}$ from the latent space, Papamakarios et al. [13] suggest to optimize $\mathbb{E}_{p^*(\mathbf{v})}[\log p_\lambda(\mathbf{v})]$ where $p^*(\mathbf{v})$ is the true distribution of dataset samples (maximum likelihood optimization). This expectation can be estimated by computing inverse samples $\mathbf{z_0} = T^{-1}\left(U^{-1}(\mathbf{v})\right)$, their log-probability under the $q_\phi(\mathbf{z}_0|\mathbf{x})$ distribution, and log-determinants of Jacobian matrices of successive layers of T^{-1} and U^{-1} (see Equation 4). This approach assumes that normalizing flows provide sufficient expressive power to model complex probability distributions such as $p^*(\mathbf{v})$. Esling et al. [3] also relied on several closed-form priors and on modeling an error ϵ such that $\mathbf{v} = U(\mathbf{z}_K) + \epsilon$.

However, we could not successfully train these methods on large (> 100) sets of parameters. Reasonable learning rate reduction (down to a 0.01 factor) and long warmup periods (100 epochs) helped but did not ensure models convergence. The required learning rate reduction resulted in prohibitively long training durations.

Nonetheless, we observed that RealNVP gave consistent results when directly maximizing $\log p_\lambda(\mathbf{v}|\mathbf{z}_K)$ without using the inverse flow transforms. Furthermore, synthesizer parameters inference seemed equivalent or better compared to baseline MLP regression networks.

Thus, we propose to use a flow model as a usual feedforward neural network for regression, which can be seen as an additional decoder. During optimization, the regression flow is only used for sampling and its tractable Jacobian determinant is left aside. This straightforward approach allows to take advantage of the expressive power of auto-regressive networks, while keeping the bijective relationship between latent vectors and synthesizer parameters, and a known transform between their probability densities.

3.1.3. Synthesizer parameters vector

Most synthesizers provide numerical (e.g., oscillator amplitude, cut-off frequency, ...) as well as categorical (e.g., waveform, filter type, ...) controls. Preset files usually store all parameters' values as numerical data in the $[0, 1]$ range, using quantization for categorical and discrete numerical controls. While such representations make parameters easier to handle and to automate, it induces an ordering of categorical variables which is often inappropriate.

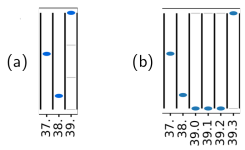

Figure 2: *Two representations of a subset of three parameters from a preset. (a) VST representation: all parameters are stored as numerical although parameter* 39 *contains categorical data. (b) Learnable representation: parameters* 37 *and* 38 *are left unchanged while* 39 *is one-hot encoded.*

Experiments presented in the following sub-sections compare different learnable representations of presets. *Num only* indicates a numerical representation of all parameters, including categorical ones; *NumCat* indicates that numerical parameters are learned as continuous numerical variables and that categorical parameters are one-hot encoded (Figure 2); *NumCat++* indicates that categorical and low-cardinality (up to 32) discrete numerical parameters are one-hot encoded. We assume that regression flows are able to handle such heterogeneous vectors of data and will discuss this hypothesis in sub-section 3.4.

3.2. Dataset

3.2.1. Dexed synthesizer

Our model was trained to program the Dexed [1] VST synthesizer, which is an open-source software clone of the famous Yamaha DX7 FM synthesizer. VST synthesizers can be used inside the RenderMan [2] Python wrapper to render audio files from presets.

FM synthesizers are able to produce diverse sounds with rich spectral content, and Dexed automatic programming has been studied in previous works [1, 4]. However, these studies relied on randomly generated presets which do not accurately represent musicians' use of a DX7 synthesizer.

Hence, we collected more than 12k DX7 public-domain *cartridges* – which contains 32 presets each – from various online sources. Cartridges were read using parts of Dexed source code. A large majority of those presets were duplicates, which were discarded. Some presets produced audio at a lower than -40dB peak RMS volume, and were also removed. The final dataset contains approximately 30k items.

Then, we wanted to better understand what our dataset was made of. For all presets, a single MIDI note (pitch 60, velocity 100) was played and held for 3.0s and audio was recorded for 4.0s at 22,050 kHz. A 100ms fadeout was applied at the end of audio files. In order to label presets, we performed Harmonic-Percussive Source Separation (HPSS) [16] with a residual component [17] on all audio files. Samples with more than 35% harmonic (resp. percussive) spectral energy were labeled *harmonic* (resp. *percussive*). 27.4k presets were *harmonic* and 1.5k were *percussive*. 1.7k presets, which contained mostly HPSS residuals, had no descriptor at the end of this process. They were arbitrarily assigned a *sfx* label.

Presets and labels were stored in a SQLite database (26MB) available from our Github repository [3]. Authors and sources of the original DX7 cartridges are also reported in this database.

3.2.2. Parameters

Dexed provides 155 automatable parameters but ten of them (main volume and filter, and on/off switches of the six oscillators) were constant across the whole dataset. Moreover, we decided to set the transpose control to its middle C3 position. Hence, the total amount of learnable parameters is 144.

DX7 FM synthesis relies on six identical oscillators with 21 parameters each. To test our models' scalability, all experiments were also conducted on a reduced set of 81 parameters, which includes the three first oscillators and general DX7 controls.

[1] https://github.com/asb2m10/dexed
[2] https://github.com/fedden/RenderMan
[3] https://github.com/gwendal-lv/preset-gen-vae

Table 1: *Comparison of flow- and MLP-based regression models for presets inference. Reported values are averages of mean values from the five training folds, plus or minus one standard deviation.*

				Parameters		**Audio**		
Params count	Regression model	Learnable representation	Latent dim. D	Numerical MAE (10^{-1})	Categorical Accuracy (%)	Spectrogram MAE log	Spectrogram SC	MFCC MAE
81	Flow	Num only	81	1.391 ± 0.031	58.1 ± 1.8	0.547 ± 0.021	1.08 ± 0.07	17.9 ± 0.9
		NumCat	140	1.187 ± 0.021	84.3 ± 1.0	0.510 ± 0.014	1.28 ± 0.06	17.2 ± 0.6
		NumCat++	340	$\mathbf{0.916} \pm 0.025$	$\mathbf{85.3} \pm 1.0$	$\mathbf{0.470} \pm 0.009$	1.15 ± 0.06	$\mathbf{12.1} \pm 0.5$
81	MLP	Num only	340	1.242 ± 0.022	66.6 ± 0.7	0.496 ± 0.005	0.99 ± 0.01	16.1 ± 0.3
		NumCat		1.187 ± 0.013	83.9 ± 0.9	0.548 ± 0.022	1.22 ± 0.03	17.1 ± 0.4
		NumCat++		1.026 ± 0.028	83.3 ± 0.7	0.471 ± 0.011	$\mathbf{0.98} \pm 0.01$	12.2 ± 0.4
144	Flow	Num only	144	1.465 ± 0.008	63.4 ± 2.3	0.740 ± 0.035	1.03 ± 0.05	19.2 ± 0.6
		NumCat	224	1.284 ± 0.013	85.1 ± 0.3	0.650 ± 0.011	1.09 ± 0.01	18.6 ± 0.6
		NumCat++	610	$\mathbf{1.049} \pm 0.011$	$\mathbf{86.0} \pm 0.6$	$\mathbf{0.615} \pm 0.008$	1.03 ± 0.01	$\mathbf{14.7} \pm 0.3$
144	MLP	Num only	610	1.360 ± 0.016	68.6 ± 0.8	0.699 ± 0.007	0.97 ± 0.01	19.1 ± 0.1
		NumCat		1.274 ± 0.004	84.4 ± 0.6	0.654 ± 0.009	1.03 ± 0.02	18.6 ± 0.2
		NumCat++		1.110 ± 0.021	84.4 ± 0.5	0.637 ± 0.007	$\mathbf{0.94} \pm 0.03$	15.0 ± 0.4

3.2.3. Spectrograms

Audio files were rendered as described in sub-section 3.2.1 and were *not* amplitude-normalized. MIDI notes played depended on the experiment, and each MIDI note corresponded to approximately 10GB of audio files. These files were transformed into 257-bins Mel-Spectrograms computed from the log-magnitude of a STFT (Hann window, width 1024, hop size 256, 347 time frames). Then, a -120 dB threshold was applied to these Mel-spectrograms, which were finally scaled into $[-1, 1]$ using the minimum and maximum amplitudes of the entire dataset.

3.3. Implementation

3.3.1. Neural networks

All models were implemented using PyTorch and nflows[4]. Details about CNNs and Fully-Connected (FC) layers are provided in Figure 1. Latent and regression flows are made of $K = L = 6$ layers. Internal scale and translation coefficients of each RealNVP layer are computed using a 2-layer MLP (300 hidden units) with residual connection, Batch Normalization (BN) and 0.4 dropout probability. A hardtanh activation with $[0, 1]$ range is applied to the regression flow's output.

3.3.2. Loss functions

A Mean-Square Error (MSE) reconstruction loss is computed on decoder outputs $\hat{\mathbf{x}}$ and the negative regularization loss is described in equation 5. An MSE loss is evaluated on each numerical parameter, while a categorical cross-entropy loss is computed on each categorical sub-vector. Cross-entropy softmax activations have a 0.2 temperature to compensate for the limited range of categorical logits. Nonetheless, we observed that categorical representations of discrete numerical parameters lead to higher losses for a given accuracy. Thus, we apply an empirical 0.2 factor on all categorical losses. The total regression loss is the sum of per-parameter losses.

To perform fair comparisons between variants of our models (sub-sections 3.4.1 and 3.4.2), we decided to normalize all three losses, i.e., divide them by the dimension of corresponding data. This prevents the model from favoring, for instance, parameters inference over spectral reconstruction when D increases.

[4] https://github.com/bayesiains/nflows

Such a normalization implies a $\beta \approx D/E$ regularization factor, which roughly ranges from 150 to 1000 across presented tests. We multiplied the normalized latent loss by an arbitrary 0.2 factor to improve the tradeoff between VAE regularization and $\hat{\mathbf{x}}$ reconstruction accuracy. Resulting β values are similar to those of Higgins et al. [8] for 178x218 pictures modeling tasks, who also reported that the β hyper-parameter is hard to tune, even on a logarithmic scale.

3.3.3. Training and evaluation

Presets were randomly split into a held-out test set (20%) and a train/validation set (80%) used during models development. Each model was trained five times in order to perform a k-fold cross-validation procedure (64% train set, 16% validation set). Results presented in Tables 1 and 2 were obtained from the test set only.

Models were trained during 400 epochs using the Adam optimizer with a minibatch size of 160. In order to help normalizing flows training, the learning rate increased linearly from 2×10^{-5} to 2×10^{-4} over the first 6 epochs. A scheduler divides the learning rate by 5 when validation losses did not improve during 12 epochs, and training could be stopped earlier if learning rate became lower than 10^{-7}. A linear β-warmup [18] from 50% to 100% was performed from epoch 0 to epoch 25. During early tests, we remarked that β-warmup starting from 0% significantly decreased validation performances. Models training lasted about 2.5 hours on two Nvidia GTX 1070 GPUs.

3.4. Results

3.4.1. Flow regression

This first experiment studies how the flow-based regression network handles the three proposed representations of presets. MIDI notes played had a pitch of 65 and an intensity of 85.

Results are presented in Table 1. The Mean Absolute Error (MAE) of numerical DX7 parameters and accuracy of categorical DX7 parameters are reported. To measure audio accuracy, audio files were rendered using inferred presets. The MAE between true and synthesized log STFTs, as well as the 40-band MFCCs (86 time frames) MAE are presented. Spectral Convergence (SC), which measures a discrepancy between the largest components of linear-scale STFTs [19], is also reported. While these three metrics provide audio similarity measurements, an in-depth perceptual

evaluation is left for future works. Comparative audio examples are available from the paper's companion website [5].

The *Num only* representation, which induces an unnatural ordering of categorical variables, leads to the worst results for most metrics. *NumCat*, which is the most natural representation, generally improves performance, and *NumCat++* helps improve it even more. The error on MFCCs, which can be considered as an error on timbre and general harmonic structure, is particularly reduced by this last model. This might be partially explained by the one-hot representation of some frequency discrete numerical controls, which are likely to take specific values such as $+0$, ± 7 or ± 12 semitones. We conclude that the heterogeneous representations of presets introduced in sub-section 3.1.3 do improve performance of the regression flow.

Surprisingly, good SC values can be observed for models which perform poorly otherwise. However, SC measures an error on a non-perceptual linear scale. After listening to some presets inferred by *Num only* models, it seems that these models lead to simpler and non-diversified sounds, but reconstruct quite well the main spectral components hence present the best SC values.

Regarding scalability, the performance of our model degrades only slightly when using six oscillators (144 parameters) instead of three (81 parameters). This could be expected because the three added oscillators are the deepest ones in DX7 FM architectures (i.e., routing of oscillators signals, also called *algorithms*). They are usually responsible for subtle modulations of higher harmonics, which are harder to identify from Mel-Spectrograms. Interestingly, the accuracy of categorical parameters inference improves as the number of learned parameters increases. These elements allow to conclude that presented models are scalable and should be usable for any synthesizer that provides a larger number of controls.

3.4.2. MLP regression - constant latent space dimension

The *NumCat++* representation introduces a quite large increase of D, which eases the compression task of the spectral VAE. Therefore, it is necessary to know if the benefits of *NumCat++* models come from the representation itself, or from the D increase.

For this second experiment, the three parameter representations were tested with a constrained large latent space dimension D. However, the invertible transform U had to be replaced by a non-invertible, feedforward neural network. We chose a 4-layers MLP with 1024 hidden units and ReLU hidden activations. BN and dropout (0.4 probability) were appended to the two first MLP layers only. Similar to Figure 1, a hardtanh activation was applied to the last layer's output. The largest MLP regression network contains 3.0M parameters (5.9 MFLOP) whereas the largest flow regression network contains 1.6M parameters but requires more operations per item (11.3 MFLOP).

Results are also reported in Table 1. First, the *Num only* representation leads to the worst results regarding parameters inference and for most audio criteria. Second, compared to *NumCat*, the *NumCat++* models improve all metrics but categorical parameters accuracy, which is similar or slightly decreases. Therefore, we conclude that performances observed in sub-section 3.4.1 were not caused by the D increase alone. Third, these results also allow to compare MLP- and flow-based regression models. Consistent with [3], we confirm that MLP regression networks generally underperform.

[5] https://gwendal-lv.github.io/preset-gen-vae/

4. MULTI-CHANNEL SPECTROGRAMS

4.1. Multiple input notes

Most synthesizers provide parameters to modulate sound depending on the pitch and intensity of played notes. For instance, a cut-off frequency can depend on note pitch, while an oscillator's amplitude can be related to the articulation of note via MIDI velocity (intensity). To the best of our knowledge, these parameters have been neglected in all related studies.

For a model to learn these parameters, multiple spectrograms corresponding to different MIDI notes should be provided. Therefore, we propose an architecture that encodes and decodes multiple spectrograms (Figure 3) generated from the same ground truth preset. This approach is conceptually related to the Generative Query Network [20] which feeds multiple 2D renders of a 3D scene to a model that infers a high-level representation of that scene. Here, a 3D scene corresponds to a preset and a high-level scene representation corresponds to a $\mathbf{z}_K$ latent code. A 2D view of a scene (from a given position and angle) corresponds to a particular audio rendering of a preset (using a given MIDI note).

VAE inputs are multi-channel images, each channel corresponding to a single note. The experiment presented below processes six-channel spectrograms whose associated (pitch, intensity) MIDI values are $(40, 85)$, $(50, 85)$, $(60, 42)$, $(60, 85)$, $(60, 127)$ and $(70, 85)$. This architecture relies on a single-spectrogram encoder neural network which is sequentially applied to each input channel. The outputs are small 256-channel feature maps which contain deep, high-level information about each spectrogram. These feature maps are then stacked and mixed together by a convolutional layer. The decoder follows a similar principle.

Compared to the previous model (Figure 1), this new VAE processes six times more data thus requires more parameters. Our multi-channel VAE neural network contains less than two times more parameters (42.8M vs. 27.3M) so that it can be considered parameter efficient.

4.2. Experiment

To compare single- and multi-channel architectures fairly, and to demonstrate the interest of the latter, we used a slightly modified version of the single-channel encoder presented in Figure 1. Firstly, the 512-channel encoder and decoder layers were deepened to 1800 channels such that the total number of parameters was 43.0M. Secondly, the two first elements of μ_0 were set to the pitch and intensity (normalized into $[-1, 1]$) of the input spectrogram. The two first elements of σ_0 were set to $2/127$.

Both models were trained as described in the previous section. Training durations increased to approximately seven hours.

4.3. Results

Results are available in Table 2. Similar to the previous section, audio synthesis accuracy and parameters inference accuracy are reported. Audio accuracy evaluation was performed on the six training MIDI notes, whereas Table 1 focused on one note. Results also include inference accuracy for parameters specifically related to the pitch and intensity of played notes. Models with multi-channel input spectrograms demonstrate a significant performance increase for all criteria but SC (which was discussed earlier).

Learning all parameters of a synthesizer by using a single MIDI note was probably an ill-formed problem, because some parame-

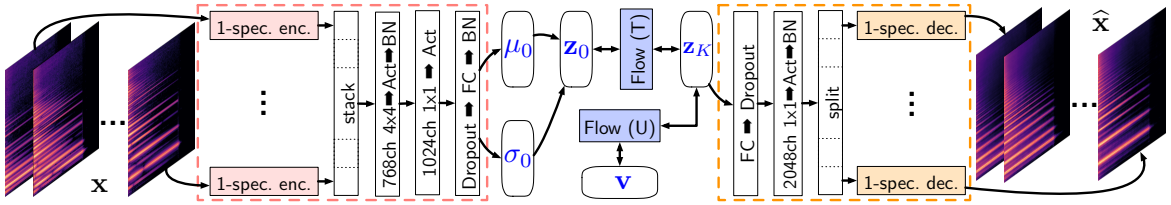

Figure 3: *Multi-channel spectrograms architecture.* 1-spec *blocks partially encode or decode a single spectrogram. They contain the blocks indicated by the same color on Figure 1. The* 1-spec enc *and* 1-spec dec *neural networks are unique and sequentially applied.*

Table 2: *Comparison of the single-channel and multi-channel models described in sub-section 4.1.*

Params count	D	Input channels	All parameters: Numerical MAE (10^{-1})	All parameters: Categorical Accuracy (%)	Pitch and intensity params: Numerical MAE (10^{-1})	Pitch and intensity params: Categorical Accuracy (%)	Audio: Spectrogram MAE log	Audio: Spectrogram SC	Audio: MFCC MAE
81	340	1	0.97 ± 0.01	83.9 ± 0.4	0.240 ± 0.003	85.6 ± 0.3	0.472 ± 0.005	1.37 ± 0.05	12.1 ± 0.3
		6	0.86 ± 0.01	86.3 ± 0.4	0.185 ± 0.006	87.3 ± 0.3	0.456 ± 0.012	1.44 ± 0.28	11.1 ± 0.3
144	610	1	1.14 ± 0.02	84.0 ± 0.3	0.308 ± 0.008	84.3 ± 0.3	0.643 ± 0.009	1.03 ± 0.02	15.2 ± 0.1
		6	0.99 ± 0.02	86.7 ± 0.4	0.240 ± 0.008	86.4 ± 0.3	0.595 ± 0.009	1.05 ± 0.04	13.9 ± 0.3

ters do require multiple pitches and/or intensities to be estimated. Hence, we argue that all automatic synthesizer programming frameworks should implement multi-channel convolutional structures such as ours.

5. DISCUSSION

5.1. Presets inference and encoding

When our model is used to program a synthesizer from audio files, the decoder neural network is not used and the data path resembles non-generative solutions such as [2]. An important contribution of VAE-based synthesizer programming, as initially proposed by Esling et al. [3], is to introduce an auditory-meaningful latent bottleneck from which new presets can be generated.

Section 3 demonstrated the usability of our model for full-size (144 parameters) DX7 presets, whereas previous similar models focused on subsets of parameters and showed degraded performance as the number of learned parameters increased. Moreover, we provided means to improve performance by turning this inference problem into a hybrid regression and classification problem.

Section 4 focused on learning *dynamic* parameters which are related to the pitch and intensity of notes played into the synthesizer. This aspect had been neglected in the relevant literature. We introduced a multi-channel spectral VAE and proved that it is able to learn such parameters. This means that these *dynamic* audio features, which quantify how the sound evolves in relation to MIDI notes played, were properly encoded into the latent space. This multi-channel architecture can be used for out-of-domain synthesizer programming, e.g. using voice or acoustic instrument input sounds. However, this matter requires a more in-depth study and is left for future research.

Our final architecture ensures that any preset – including one-hot encoded parameters – can be precisely encoded into the latent space by inverting the numerically stable RealNVP-based regression flow. Thanks to the strong latent β-regularization of the spectral VAE, the auditory latent space is continuous. As described in [3], it becomes possible to generate new similar presets by encoding a given preset $\mathbf{v}$ as $\mathbf{z}_K = U^{-1}(\mathbf{v})$, generating new latent vectors from the neighborhood of $\mathbf{z}_K$, and converting them back into presets using U. Moreover, smooth preset morphing effects can be performed by interpolating between $\mathbf{z}_K$ latent vectors. Preset inference and generation examples are available from the paper's companion website (link provided in sub-section 3.4.1).

5.2. Latent space and macro-controls

The latent space dimension's lower bound is the number of parameters, which can reach several hundreds with some synthesizers. Moreover, section 3 demonstrated that increasing the size of presets learnable representations – which constrains the latent space dimension D – improves preset inference.

Therefore, is seems hard to handle a synthesizer by using the numerous coefficients of $\mathbf{z}_K$ as independent *macro-controls*. It is also probably impossible to assign a distinct perceptual meaning to each coefficient of $\mathbf{z}_K$. Hence, in contrast to [3], we argue that the objective of disentangled semantic macro-controls might be unachievable, and might not be suited to such generative models. Instead of individual macro-controls, tools such as graphical presets interpolators [21, 22] can be used to manage large latent vectors directly for presets generation.

However, the size and entanglement of the latent space can be discussed more. The latent space dimension D of our models might seem quite large, but it remains much lower than that of the NSynth paper [23] for instance. Disentanglement is harder to assess. Although decorrelation does not imply disentanglement, it is a necessary condition. Thus, we computed Spearman rank correlation matrices on $\mathbf{z}_K$ along with corresponding p-values matrices. Correlation coefficients lie in the $[-1, 1]$ range. Matrices corresponding to training folds of the last model (multi-channel spectrograms, 144 parameters, $D = 610$) were stacked to provide the following metrics. 78% of latent variables are unlikely to be fully-decorrelated (p-value < 0.05), which is a quite high proportion. Nonetheless, the average absolute Spearman correlation coefficient is 0.10 (SD $= 0.10$), which can be considered as a low (< 0.3) correlation. This last result seems to indicate that most of latent variables contains useful audio information although a weak correlation exists between most of them.

6. CONCLUSION

This paper presented solutions to improve automatic synthesizer programming using spectral VAE models with regression flows. These models are able to infer synthesizer parameters from audio samples but can also generate new presets from the regularized VAE latent space. Compared to previous works, architecture modifications and new training methods were introduced to scale models to large vectors of synthesizer parameters. Moreover, combinations of representations – continuous or one-hot encoded – of synthesizer parameters were compared. An heterogeneous representation of categorical and numerical variables demonstrated large improvements in terms of parameters inference and synthesized audio accuracy.

Then, it was explained that some synthesizer parameters need multiple input sounds to be properly estimated. A new VAE architecture, which handles multi-channel spectrograms, was presented. Performances were significantly enhanced while neural networks' sizes had been only moderately increased.

All experiments were conducted on a software implementation of the DX7 FM synthesizer. Whereas previous works on the DX7 used datasets of randomly generated presets, we gathered and curated 30k human-made presets. The dataset, as well as models and evaluation source code, is available from our Github repository. We hope that our work will help extend VAE-based control and automatic programming to many other synthesizers and encourage developers and researchers to fork our source code for future experiments or to integrate it into existing VST plug-ins.

7. ACKNOWLEDGMENTS

We thank Alexandra Degeest and Rudi Giot for their thoughtful comments and for proofreading this paper.

8. REFERENCES

[1] M. J. Yee-King, L. Fedden, and M. d'Inverno, "Automatic programming of vst sound synthesizers using deep networks and other techniques," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 150–159, 2018.

[2] O. Barkan, D. Tsiris, O. Katz, and N. Koenigstein, "Inversynth: Deep estimation of synthesizer parameter configurations from audio signals," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 12, pp. 2385–2396, 2019.

[3] P. Esling, N. Masuda, A. Bardet, R. Despres, and A. Chemla-Romeu-Santos, "Flow synthesizer: Universal audio synthesizer control with normalizing flows," *Applied Sciences*, vol. 10, no. 1, pp. 302, 2020.

[4] J. Shier, G. Tzanetakis, and K. McNally, "Spiegelib: An automatic synthesizer programming library," in *Audio Engineering Society Convention 148*, May 2020.

[5] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *International Conference on Learning Representations*, 2014.

[6] L. Girin, F. Roche, T. Hueber, and S. Leglaive, "Notes on the use of variational autoencoders for speech and audio spectrogram modeling," in *International Conference on Digital Audio Effects*, 2019.

[7] S. Zhao, J. Song, and S. Ermon, "Infovae: Balancing learning and inference in variational autoencoders," in *The Thirty-Third AAAI Conference on Artificial Intelligence*, 2019, pp. 5885–5892.

[8] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," *International Conference on Learning Representations*, 2017.

[9] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International Conference on Machine Learning*. PMLR, 2015, pp. 1530–1538.

[10] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," in *Advances in Neural Information Processing Systems*, 2016, vol. 29.

[11] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," in *Advances in Neural Information Processing Systems*, 2017, vol. 30.

[12] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *International Conference on Learning Representations*, 2017.

[13] G. Papamakarios, E. Nalisnick, D. Rezende, Shakir Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," *arXiv preprint arXiv:1912.02762*, 2019.

[14] R. A. Garcia, "Automatic design of sound synthesis techniques by means of genetic programming," in *Audio Engineering Society Convention 113*, Oct 2002.

[15] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, "Ddsp: Differentiable digital signal processing," in *International Conference on Learning Representations*, 2020.

[16] D. Fitzgerald, "Harmonic/percussive separation using median filtering," in *International Conference on Digital Audio Effects (DAFx)*, 2010.

[17] J. Driedger, M. Müller, and S. Disch, "Extending harmonic-percussive separation of audio signals.," in *ISMIR Conference*, 2014, pp. 611–616.

[18] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther, "Ladder variational autoencoders," in *Advances in Neural Information Processing Systems*, 2016.

[19] S. Ö. Arık, H. Jun, and G. Diamos, "Fast spectrogram inversion using multi-head convolutional neural networks," *IEEE Signal Processing Letters*, vol. 26, no. 1, pp. 94–98, 2019.

[20] S. M. Ali Eslami et al., "Neural scene representation and rendering," *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.

[21] D. Gibson and R. Polfreman, "Analyzing journeys in sound: usability of graphical interpolators for sound design," *Personal and Ubiquitous Computing*, pp. 1–14, 2020.

[22] G. Le Vaillant, T. Dutoit, and R. Giot, "Analytic vs. holistic approaches for the live search of sound presets using graphical interpolation," in *International Conference on New Interfaces for Musical Expression*, 2020, pp. 227–232.

[23] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, and K. Simonyan, "Neural audio synthesis of musical notes with wavenet autoencoders," in *International Conference on Machine Learning*, 2017, p. 1068–1077.