UNIVERSAL AUDIO SYNTHESIZER CONTROL WITH NORMALIZING FLOWS

Philippe Esling[1], Naotake Masuda[1], Adrien Bardet[1], Romeo Despres[1], Axel Chemla–Romeu-Santos[1,2]

[1] IRCAM - CNRS UMR 9912
Sorbonne Université, Paris, France
esling@ircam.fr

[2] Laboratorio d'Informatica Musicale (LIM)
UNIMI, Milano, Italy
axel.chemla@unimi.it

ABSTRACT

The ubiquity of sound synthesizers has reshaped music production and even entirely defined new music genres. However, the increasing complexity and number of parameters in modern synthesizers make them harder to master. Hence, the development of methods allowing to easily create and explore with synthesizers is a crucial need.

Here, we introduce a novel formulation of audio synthesizer control. We formalize it as finding an organized latent audio space that represents the capabilities of a synthesizer, while constructing an invertible mapping to the space of its parameters. By using this formulation, we show that we can address simultaneously *automatic parameter inference*, *macro-control learning* and *audio-based preset exploration* within a single model. To solve this new formulation, we rely on Variational Auto-Encoders (VAE) and Normalizing Flows (NF) to organize and map the respective *auditory* and *parameter* spaces. We introduce a new type of NF named *regression flows* that allow to perform an invertible mapping between separate latent spaces, while steering the organization of some of the latent dimensions. We evaluate our proposal against a large set of baseline models and show its superiority in both parameter inference and audio reconstruction. We also show that the model disentangles the major factors of audio variations as latent dimensions, that can be directly used as *macro-parameters*. Finally, we discuss the use of our model in creative applications and its real-time implementation in Ableton Live[1].

1. INTRODUCTION

Synthesizers are parametric systems able to generate audio signals ranging from musical instruments to entirely unheard-of sound textures. Since their commercial beginnings more than 50 years ago, synthesizers have revolutionized music production, while becoming increasingly accessible, even to neophytes with no background in signal processing.

[1] All code, supplementary figures, results and plugins are available on a supporting webpage: https://acids-ircam.github.io/flow_synthesizer/

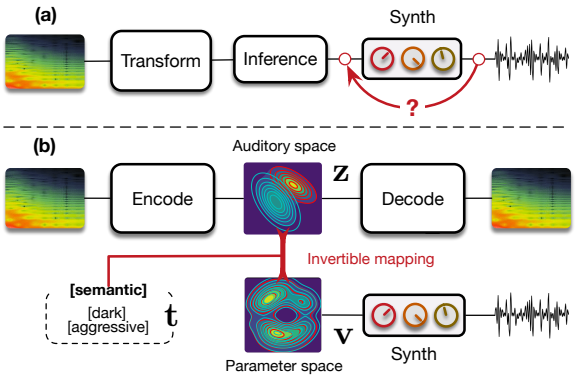

Figure 1: *Universal synthesizer control*. (a) Previous methods perform direct inference from audio, which is limited by non-differentiable synthesis and lacks high-level control. (b) Our novel formulation states allows to learn an organized latent space $\mathbf{z}$ of the synthesizer's audio capabilities, while mapping it to the space $\mathbf{v}$ of its synthesis parameters.

While there exists a variety of sound synthesis types [1], they all require an *a priori* knowledge to make the most out of a synthesizer possibilities. Hence, the main appeal of these systems (namely their versatility provided by large sets of parameters) also entails their major drawback. Indeed, the sheer combinatorics of parameter settings makes exploring all possibilities to find an adequate sound a daunting and time-consuming task. Furthermore, there are highly non-linear relationships between the parameters and the resulting audio. Unfortunately, no synthesizer provides intuitive controls related to perceptual and semantic properties of the synthesis. Hence, a method allowing an intuitive and creative exploration of sound synthesizers has become a crucial need, especially for non-expert users.

A potential direction taken by synth manufacturers, is to propose *macro-controls* that allow to quickly tune a sound by controlling multiple parameters through a single knob. However, these need to be programmed manually, which still requires expert knowledge. Furthermore, no method has ever tried to tackle this *macro-control learning* task, as this objective appears unclear and depends on a variety of unknown factors. An alternative to manual parameter setting would be to infer the set of parameters that could best

reproduce a given *target sound*. This task of *parameter inference* has been studied in the past years using various techniques. In Cartwright et al. [2], parameters are iteratively refined based on audio descriptors similarity and relevance feedback provided by the user. However, this approach appears to be rather inaccurate and slow. Garcia et al. [3] proposed to use genetic programming to directly grow modular synthesizers to solve this problem. Although the approach is appealing and appears accurate, the optimization of a single target can take from 10 to 200 hours, which makes it unusable. Recently, Yee-king et al. [4] showed that a bi-directional LSTM with highway layers can produce accurate parameters appproximations. However, this approach does not allow for any user interaction. All of these approaches share the same flaws that (i) though it is unlikely that a synthesizer can generate exactly any audio target, none explicitly model these limitations, (ii) they do not account for the non-linear relationships that exist between parameters and the corresponding synthesized audio. Hence, no approach has succeeded in unveiling the true relationships between these *auditory* and *parameters* spaces. Here, we argue that it is mandatory to organize the parameters and audio capabilities of a given synthesizer in their respective spaces, while constructing an invertible mapping between these spaces in order to access a range of high-level interactions. This idea is depicted in Figure 1

The recent rise of *generative models* might provide an elegant solution to these questions. Indeed, amongst these models, the *Variational Auto-Encoder* (VAE) [5] aims to uncover the underlying structure of the data, by explicitly learning a *latent space* [5]. This space can be seen as a high-level representation, which aims to disentangle underlying variation factors and reveal interesting structural properties of the data [5, 6]. VAEs address the limitations of control and analysis through this latent space, while being able to learn on small sets of examples. Furthermore, the recently proposed *Normalizing Flows* (NF) [7] allow to model highly complex distributions in the latent space. Although the use of VAEs for audio applications has only been scarcely investigated, Esling et al. [8] recently proposed a perceptually-regularized VAE that learns a space of audio signals aligned with perceptual ratings via a regularization loss. The resulting space exhibits an organization that is well aligned with perception. Hence, this model appears as a valid candidate to learn an organized audio space.

In this paper, we introduce a radically novel formulation of audio synthesizer control by formalizing it as the general question of finding an invertible mapping between two learned latent spaces. In our case, we aim to map the audio space of a synthesizer's capabilities to the space of its parameters. We provide a generic probabilistic formalization and show that it allows to address simultaneously the tasks of *parameter inference*, *macro-control learning*, *audio-based preset exploration* and *semantic dimension discovery* within a single model. To elegantly solve this formulation, we introduce *conditional regression flows*, which map a latent space to any given target space, while steering the organization of some dimensions to match target distributions. Our complete model is depicted in Figure 2.

Based on this formulation, *parameter inference* simply consists of encoding the audio target to the latent audio space that is mapped to the parameter space. Interestingly, this bypasses the well-known blurriness issue in VAEs as we can generate directly with the synthesizer. We evaluate our proposal against a large set of baseline models and show its superiority in parameter inference and audio reconstruction. Furthermore, we show that our model is the first able to address the new task of automatic *macro-control learning*. As the latent dimensions are continuous and map to the parameter space, they provide a natural way to learn the perceptually most significant macro-parameters. We show that these controls map to smooth, yet non-linear parameters evolution, while remaining perceptually continuous. Furthermore, as our mapping is invertible, we can map synthesis parameters back to the audio space. This allows intuitive *audio-based preset exploration*, where exploring the neighborhood of a preset encoded in the audio space yields similarly sounding patches, yet with largely different parameters. Finally, we discuss creative applications of our model and real-time implementation in *Ableton Live*.

2. STATE-OF-ART

2.1. Generative models and variational auto-encoders

Generative models aim to understand a given set $\mathbf{x} \in \mathbb{R}^{d_x}$ by modeling the underlying probability distribution of the data $p(\mathbf{x})$. To do so, we consider *latent variables* defined in a lower-dimensional space $\mathbf{z} \in \mathbb{R}^{d_z}$ ($d_z \ll d_x$), a higher-level representation that could have led to generate a given example. The complete model is defined by the joint distribution $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$. Unfortunately, real-world data follow complex distributions, which cannot be found analytically. The idea of *variational inference* (VI) is to solve this problem through *optimization* by assuming a simpler approximate distribution $q_\phi(\mathbf{z}|\mathbf{x}) \in \mathcal{Q}$ from a family of approximate densities [9]. The goal of VI is to minimize the difference between this approximation and the real distribution, by minimizing the Kullback-Leibler (KL) divergence between these densities

$$q_\phi^*(\mathbf{z}|\mathbf{x}) = \text{argmin}_{q_\phi(\mathbf{z}|\mathbf{x}) \in \mathcal{Q}} \mathcal{D}_{KL}\left[q_\phi\left(\mathbf{z}|\mathbf{x}\right) \parallel p\left(\mathbf{z}|\mathbf{x}\right)\right]$$

By developing this KL divergence and re-arranging terms (the detailed development can be found in [5]), we obtain

$$\begin{aligned}\log p(\mathbf{x}) - D_{KL}\left[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})\right] \\ = \mathbb{E}_{\mathbf{z}}\left[\log p(\mathbf{x}|\mathbf{z})\right] - D_{KL}\left[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\right] \quad (1)\end{aligned}$$

This formulation describes the quantity we want to model $\log p(\mathbf{x})$ minus the error we make by using an approximate

q instead of the true p. Therefore, we can optimize this alternative objective, called the *evidence lower bound* (ELBO)

$$\mathcal{L}_{\theta,\phi} = \mathbb{E}\big[\log p_\theta(\mathbf{x}|\mathbf{z})\big] - \beta \cdot D_{KL}\big[q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})\big] \quad (2)$$

The ELBO intuitively minimizes the reconstruction error through the likelihood of the data given a latent $\log p_\theta(\mathbf{x}|\mathbf{z})$, while regularizing the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to follow a given prior distribution $p_\theta(\mathbf{z})$. We can see that this equation involves $q_\phi(\mathbf{z}|\mathbf{x})$ which *encodes* the data $\mathbf{x}$ into the latent representation $\mathbf{z}$ and a *decoder* $p_\theta(\mathbf{x}|\mathbf{z})$, which generates $\mathbf{x}$ given a $\mathbf{z}$. This structure defines the *Variational Auto-Encoder* (VAE), where we can use parametric neural networks to model the *encoding* (q_ϕ) and *decoding* (p_θ) distributions. VAEs are powerful representation learning frameworks, while remaining simple and fast to learn without requiring large sets of examples [10].

However, the original formulation of the VAE entails several limitations. First, it has been shown that the KL divergence regularization can lead both to uninformative latent codes (also called *posterior collapse*) and variance overestimation [11]. One way to alleviate this problem is to rely on the *Maximum Mean Discrepancy* (MMD) instead of the KL to regularize the latent space, leading to the WassersteinAE (WAE) model [12]. Second, one of the key aspect of VI lies in the choice of the family of approximations. The simplest choice is the *mean-field* family where latent variables are mutually independent and parametrized by distinct variational parameters $q(z) = \prod_{j=1}^{m} q_j(z_j)$. Although this provide an easy tool for analytical development, it might prove too simplistic when modeling complex data as this assumes pairwise independence among every latent axis. Normalizing flows alleviate this issue by adding a sequence of invertible transformations to the latent variable, providing a more expressive inference process.

2.2. Normalizing flows

In order to transform a probability distribution, we can rely on the *change of variable* theorem. As we deal with probability distributions, we need to *scale* the transformed density so that it still sums to one, which is measured by the determinant of the transform. Formally, let $\mathbf{z} \in \mathcal{R}^d$ be a random variable with distribution $q(\mathbf{z})$ and $f : \mathcal{R}^d \to \mathcal{R}^d$ an invertible smooth mapping. We can use f to transform $\mathbf{z} \sim q(\mathbf{z})$, so that the resulting random variable $\mathbf{z}' = f(\mathbf{z})$ has the following probability distribution

$$q(\mathbf{z}') = q(\mathbf{z}) \left| \det \frac{\partial f^{-1}}{\partial \mathbf{z}'} \right| = q(\mathbf{z}) \left| \det \frac{\partial f}{\partial \mathbf{z}} \right|^{-1} \quad (3)$$

where the last equality is obtained through the inverse function theorem. We can perform any number of transforms to obtain a final distribution $\mathbf{z}_k \sim q_k(\mathbf{z}_k)$ given by

$$\begin{aligned} q_k(\mathbf{z}_k) &= q_0(f_1^{-1} \circ ... \circ f_k^{-1}(\mathbf{z}_k)) \prod_{i=1}^{k} \left| \det \frac{\partial f_i^{-1}}{\partial \mathbf{z}_i} \right| \\ &= q_0(\mathbf{z_0}) \prod_{i=1}^{k} \left| \det \frac{\partial f_i}{\partial \mathbf{z}_{i-1}} \right|^{-1} \end{aligned} \quad (4)$$

This series of transformations, called a *normalizing flow* [7], can turn a simple distribution into a complicated multimodal density. For practical use of these flows, we need transforms whose Jacobian determinants are easy to compute. Interestingly, *Auto-Regressive* (AR) transforms fit this requirement as they lead to a triangular Jacobian matrix. Hence, different AR flows were proposed such as *Inverse AR Flows* (IAF) [13] and *Masked AR Flows* (MAF) [14]

Normalizing flows in VAEs. Normalizing flows allow to address the simplicity of variational approximations by complexifying their posterior distribution [7]. In the case of VAEs, we parameterize the approximate posterior distribution with a flow of length K, $q_\phi(\mathbf{z}|\mathbf{x}) = q_K(\mathbf{z}_K)$, and the new optimization loss can be simply written as an expectation over the initial distribution $q_0(\mathbf{z})$

$$\begin{aligned} \mathcal{L} &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x}, \mathbf{z})\right] \\ &= \mathbb{E}_{q_0(\mathbf{z}_0)}\left[\ln q_0(\mathbf{z}_0)\right] - \mathbb{E}_{q_0(\mathbf{z}_0)}\left[\log p(\mathbf{x}, \mathbf{z}_K)\right] \\ &- \mathbb{E}_{q_0(\mathbf{z}_0)}\left[\sum_{i=1}^{k} \log \left| \det \frac{\partial f_i}{\partial \mathbf{z}_{i-1}} \right|\right] \end{aligned} \quad (5)$$

The resulting objective can be easily optimized since q_0 is still a Gaussian from which we can easily sample. However, the final samples $\mathbf{z}_k$ used by the decoder are drawn from a more complex distribution.

3. OUR PROPOSAL

3.1. Formalizing synthesizer control

Considering a set of audio samples $\mathcal{D} = \{\mathbf{x}_i\}, i \in [1, n]$ where the $\mathbf{x}_i \in \mathbb{R}^d$ follow an unknown distribution $p(\mathbf{x})$, we can define latent factors $\mathbf{z} \in \mathbb{R}^z$ to model the joint distribution $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{x})$ as detailed in Section 2.1. In our case, some $\bar{\mathbf{x}} \in \mathcal{D}_s \subset \mathcal{D}$ inside this set have been generated by a given synthesizer. This synthesizer defines a generative function $f_s(\mathbf{v}; p, i) = \bar{\mathbf{x}}$ where $\mathbf{v} \in \mathbb{R}^s$ is a set of parameters that produce $\bar{\mathbf{x}}$ at a given pitch p and intensity i. However, in the general case, we know that if $\mathbf{x}_j \notin \mathcal{D}_s$, then $\mathbf{x}_j = f_s(\mathbf{v}) + \epsilon$ where ϵ models the error made when trying to reproduce any audio $\mathbf{x}_i$ with a given synthesizer. Finally, we consider that some audio examples are annotated with a set of *categorical semantic tags* $\mathbf{t}_i = \{0, 1\}^t$, which define high-level perceptual properties that separate *unknown* latent factors $\mathbf{z}$ and *target* factors $\mathbf{t}$. Hence, the complete generative story of a synthesizer can be defined as

$$p(\mathbf{x}, \mathbf{v}, \mathbf{t}, \mathbf{z}) = p(\mathbf{x}|\mathbf{v}, \mathbf{t}, \mathbf{z})p(\mathbf{v}|\mathbf{t}, \mathbf{z})p(\mathbf{t}|\mathbf{z})p(\mathbf{z}) \quad (6)$$

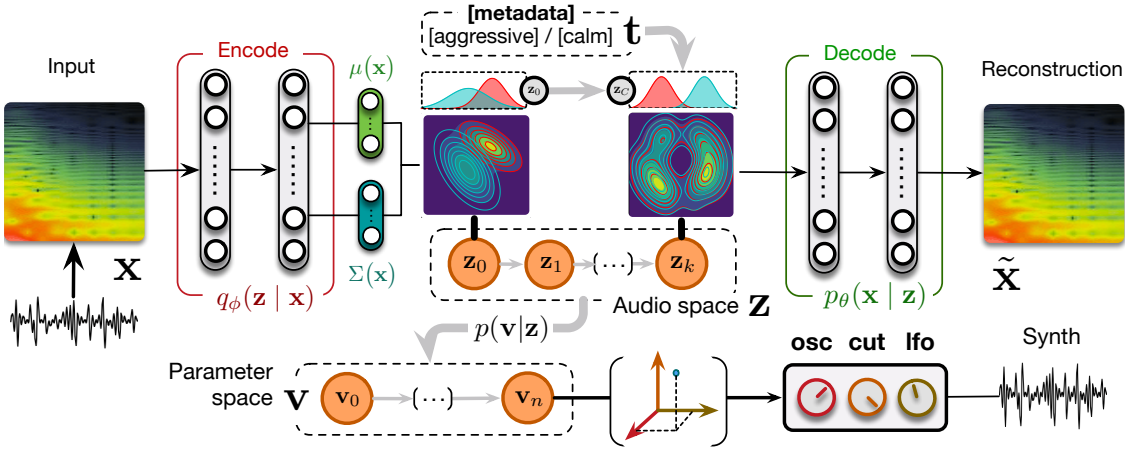

Figure 2: *Universal synthesizer control*. We learn an organized latent audio space $\mathbf{z}$ of a synthesizer capabilities with a VAE parameterized with NF. This space maps to the parameter space $\mathbf{v}$ through our proposed *regression flow* and can be further organized with metadata targets $\mathbf{t}$. This provides sampling and invertible mapping between different spaces.

This very general formulation entails our original idea that we should uncover the relationship between the latent audio $\mathbf{z}$ and parameters $\mathbf{v}$ spaces by modeling $p(\mathbf{v}, \mathbf{z})$. The advantage of this formulation is that the reduced dimensionality $\mathbb{R}^z \ll \mathbb{R}^x$ of the latent $\mathbf{z}$ simplifies the problem of parameters inference, by relying on a more adequate and smaller input space. Furthermore, this formulation also provides a natural way of learning *macro-controls* by inferring $p(\mathbf{v}|\mathbf{z})$ in the general case, where separate dimensions of $\mathbf{z}$ are expected to produce smooth auditory transforms. Although we provide a complete formalization, due to space constraints, we do not detail the use of metadata $\mathbf{t}$ in our model. We provide this information in the supporting webpage and companion article. Here, we consider that tags $\mathbf{t}$ are included in the latent factors $\mathbf{z}$ and define the model as

$$p_\theta(\mathbf{x}, \mathbf{v}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{v}, \mathbf{z})p_\theta(\mathbf{v}|\mathbf{z})p_\theta(\mathbf{z}) \tag{7}$$

3.2. Mapping latent spaces with regression flows

In order to map the latent $\mathbf{z}$ and parameter $\mathbf{v}$ spaces, we first separate our formulation so that

$$\log p_\theta(\mathbf{x}, \mathbf{v}, \mathbf{z}) = \log(p_\theta(\mathbf{x}|\mathbf{v}, \mathbf{z})p_\theta(\mathbf{z})) + \log p_\theta(\mathbf{v}|\mathbf{z}) \tag{8}$$

This allows to separately model the variational approximation detailed in Section 2.1, while solving separately the inference problem $p_\theta(\mathbf{v}|\mathbf{z})$. To address this inference, we need to find the optimal parameters $\boldsymbol{\psi}$ of a transform $f_{\boldsymbol{\psi}}$ so that $\mathbf{v} = f_{\boldsymbol{\psi}}(\mathbf{z}) + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{C}_v)$ models the inference error as a zero-mean additive Gaussian noise with covariance $\mathbf{C}_v$. Here, we assume that the covariance decomposes into $\mathbf{C}_v^{-1} = \sum_i exp(\lambda_i)\mathbf{Q}_i$, where $\mathbf{Q}_i$ are fixed basis functions and λ are hyperparameters. Therefore, the full joint likelihood that we need to optimize is given by

$$\mathcal{L}_{f_{\boldsymbol{\psi}},\lambda} = \log\left[p_\theta(\mathbf{v}|f_{\boldsymbol{\psi}}, \lambda, \mathbf{z})p_\theta(f_{\boldsymbol{\psi}}|\mathbf{z})p_\theta(\lambda|\mathbf{z})\right] \tag{9}$$

If we know the optimal transform $f_{\boldsymbol{\psi}}$ and parameters λ, the likelihood of the data can be easily computed as

$$p_\theta(\mathbf{v} \mid f_{\boldsymbol{\psi}}, \lambda, \mathbf{z}) = \mathcal{N}(\mathbf{v}; f_{\boldsymbol{\psi}}(\mathbf{z}), \mathbf{C}_v) \tag{10}$$

However, the two posteriors $p_\theta(f_{\boldsymbol{\psi}}|\mathbf{z})$ and $p_\theta(\lambda|\mathbf{z})$ remain intractable in the general case. In order to solve this issue, we rely again on variational inference by defining an approximation $q_\phi(f_{\boldsymbol{\psi}}, \lambda|\mathbf{v}, \mathbf{z})$ (see Section 2.1) and assume that it factorizes as $q(f_{\boldsymbol{\psi}}, \lambda|\mathbf{v}, \mathbf{z}) = q(f_{\boldsymbol{\psi}}|\mathbf{v}, \mathbf{z})q(\lambda|\mathbf{v}, \mathbf{z})$. Therefore, our final inference problem is

$$\begin{aligned}\mathcal{L}_{f_{\boldsymbol{\psi}},\lambda} &= \log\left[p_\theta(\mathbf{v}|f_{\boldsymbol{\psi}}, \lambda, \mathbf{z})\right] \\ &+ \mathcal{D}_{\mathrm{KL}}\left[q_\phi(f_{\boldsymbol{\psi}}|\mathbf{z}, \mathbf{v})\|p_\theta(f_{\boldsymbol{\psi}}|\mathbf{z})\right] \qquad (11)\\ &+ \mathcal{D}_{\mathrm{KL}}\left[q_\phi(\lambda|\mathbf{z}, \mathbf{v})\|p_\theta(\lambda|\mathbf{z})\right] \qquad (12)\end{aligned}$$

Hence, we can optimize our approximations through the KL divergence if we find a closed form. To solve for λ, we use a Gaussian distribution for both the prior $p_\theta(\lambda|\mathbf{z}) = \mathcal{N}(\lambda, \mu_\lambda, C_\lambda)$ and posterior $q_\phi(\lambda|\mathbf{z}, \mathbf{v}) = \mathcal{N}(\lambda, \mu_q, C_q)$. To solve this issue, we introduce the idea of *regression flows*. This allows to obtain a simple analytical solution. However, the second part of the objective might be more tedious. Indeed, to perform an accurate inference, we need to rely on a complicated non-linear function, which cannot be assumed to be Gaussian. To address this issue, we introduce the idea of *regression flows*. We consider that the transform $f_\theta(\mathbf{z})$ is a normalizing flow (see Section 2.1) and provides two different way of optimizing the approximation.

Posterior parameterization. First, we follow a reasoning akin to the original formulation of normalizing flows by parameterizing the posterior $q_\phi(f_{\boldsymbol{\psi}}|\mathbf{z}, \mathbf{v})$ with a flow $q_k(\mathbf{v}_k)$.

Hence, by developing the KL expression, we obtain

$$\mathcal{D}_{\mathrm{KL}}\left[q_\phi(f_{\boldsymbol{\psi}}|\mathbf{z},\mathbf{v})\|p(f_{\boldsymbol{\psi}}|\mathbf{z})\right] = \mathbb{E}_{q_0}\left[\log q_0(\mathbf{v}_0)\right] - \mathbb{E}_{q_0}\left[\log p(\mathbf{v}_k)\right] - \mathbb{E}_{q_0}\left[\sum_{i=1}^{k}\log\left|\det\frac{\partial f_i}{\partial \mathbf{v}_{i-1}}\right|\right] \quad (13)$$

Hence, we can now safely rely on Gaussian priors for $q_0(\mathbf{v}_0)$ and $p(\mathbf{v}_k)$. This formulation allows to consider $\mathbf{v}$ as a transformed version of $\mathbf{z}$, while being easily invertible as $\mathbf{z} = f^{-1}_{[k,1]}(\mathbf{v})$. We denote this version as $Flow_{post}$.

Conditional amortization. Here, we consider that the parameters $\boldsymbol{\psi}$ of the flow are random variables that are optimized by decomposing the posterior KL objective as

$$\mathcal{D}_{\mathrm{KL}}\left[q_\phi(f_{\boldsymbol{\psi}}|\mathbf{z},\mathbf{v})\|p(f_{\boldsymbol{\psi}}|\mathbf{z})\right] = \mathcal{D}_{\mathrm{KL}}\left[q_\phi(\boldsymbol{\psi}|\mathbf{z})\|p(\boldsymbol{\psi}|\mathbf{z})\right] + \mathbb{E}_{q_0(\mathbf{v}_0)}\left[\sum_{i=1}^{k}\log\left|\det\frac{\partial f_i}{\partial \mathbf{v}_{i-1}}\right|\right] \quad (14)$$

As we rely on Gaussian priors for the parameters, this additional KL term can be computed easily. In this version, denoted $Flow_{cond}$, parameters of the flow are sampled from their distributions before computing the resulting transform.

4. EXPERIMENTS

4.1. Dataset

Synthesizer. We constructed a dataset of synthesizer sounds and parameters, by using an off-the-shelf commercial synthesizer *Diva* developed by U-He[2]. It should be noted that our model can work for any synthesizer, as long as we obtain couples of (audio, parameters) as input. We selected Diva as (i) almost all its parameters can be MIDI-controlled, (ii) large banks of presets are available and (iii) presets include well-organized semantic tags pairs. The factory presets for Diva and additional presets from the internet were collected, leading to a total of roughly 11k files. We manually established the correspondence between synth and MIDI parameters as well as the parameters values range and distributions. We only kept continuous parameters and normalize all their values to $[0, 1]$. All other parameters are set to their fixed *default* value. Finally, we performed PCA and manual screening to select increasing sets of the most used 16, 32 and 64 parameters. We use *RenderMan*[3] to batch-generate all the audio files by playing the note for 3 sec. and recording for 4 sec. to capture the release of the note. The files are saved in 22050Hz and 16bit floating point format.

Audio processing. For each sample, we compute a 128 bins Mel-spectrogram with a FFT of size 2048 with a hop of 1024 and frequency range of $[30, 11000]$. We only keep the magnitude of the spectrogram and perform a log-amplitude transform. The dataset is randomly split between a training (80%), validation (10%) and test (10%) set before each training. We repeat the training k times to perform k-fold cross-validation. Finally, we perform a corpus-wide zero-mean unit-variance normalization based on the train set.

[2]https://u-he.com/products/diva/

[3]https://github.com/fedden/RenderMan

4.2. Models

Baseline models. In order to perform an objective evaluation of our proposal, we implemented several recent high-capacity models similar to [4]. We implement a 5-layers MLP with 2048 hidden units per layer, Exponential Linear Unit (ELU) activation, batch normalization and dropout with $p = .3$. The final layer is a sigmoid activation. We implement a gated variant of this model, denoted MLP_g. We implement a convolutional model composed of 5 layers with 128 channels of strided dilated convolutions with kernel size 7, stride 2 and an exponential dilation factor of 2^l with batch normalization and ELU activation. The convolutions are followed by a 3-layers MLP identical to the previous model. We also implement the gated variant denoted CNN_g. Finally, we implemented a variant of *Residual Networks*, with parameters settings identical to CNN and denote this model $ResCNN$.

Our models. We implemented various *AE architectures to evaluate different aspects of our proposal. To perform a fair comparison, we rely on the same setup as before, but halve the number of parameters to obtain roughly the same capacity as the baselines. First, we implement a simple deterministic AE without regularization. We implement the VAE by adding a KL regularization to the latent space and the WAE by replacing the KL by the MMD. Finally, we implement a VAE_{flow} by adding a normalizing flow composed of 16 successive IAF transforms to the VAE latent posterior. All AEs map to a latent space of dimensionality equal to the number of synthesis parameters. For probabilistic models, we perform *warmup* [10] by linearly increasing the regularization β from 0 to 1 for 100 epochs. We also apply the same weight annealing for the regression loss. We first evaluate all these models by using a simple 2-layers MLP to predict the parameters based on the latent space. Finally, we evaluate our *regression flows* by adding them to the VAE_{flow}, with an IAF of length 16.

Optimization. We train all models for 500 epochs with the ADAM optimizer, initial learning rate of 0.0002, Xavier initialization and a scheduler that halves the learning rate if the validation loss stalls for 20 epochs. With this setup, the most complex VAE_{flow} with regression flows only needs 5 hours to complete training on a NVIDIA Titan Xp GPU.

5. RESULTS

5.1. Parameters inference

We compare the accuracy of our proposal with all baseline models on the *parameters inference* task. To do so, we evaluate the distance between predicted parameters and their real values in the test dataset, by computing the magnitude-

	Parameter		Audio			
	MSE_n		SC		MSE	
MLP	0.236	± .44	6.226	± .13	9.445	± 3.1
MLP_g	0.195	± .45	1.731	± .31	7.787	±1.9
CNN	0.171	± .43	1.372	± .29	6.329	±1.9
CNN_g	0.174	± .44	1.245	± .28	6.496	±2.0
$ResCNN$	0.191	± .43	1.004	± .35	6.422	±1.9
AE	0.181	±.40	0.893	±.13	5.557	±1.7
VAE	0.182	±.32	0.810	±.03	4.901	±1.4
WAE	**0.159**	**±.37**	0.787	±.05	4.979	±1.5
VAE_{fl}	0.199	±.32	0.838	±.02	4.975	±1.4
$Flow_{post}$	0.197	±.31	**0.752**	**±.05**	**4.409**	**±1.6**
$Flow_{cond}$	0.199	±.31	1.085	±.02	6.303	±2.1

Table 1: Comparison between baseline, *AEs with MLP regression and our proposed *regression flows* on the test set. Parameters accuracy is evaluated with normalized MSE and the audio with Spectral Convergence (SC) and MSE.

normalized *Mean Square Error* (MSE_n). We average these results across k-folds and report across-runs variance. More importantly, we evaluate the distance between the audio synthesized based on these inferred parameters and the original audio through the *Spectral Convergence* (SC) and *MSE* distances, where SC is the Frobenius norm normalized over time and frequency. The results are displayed in Table 1.

As we can see, baseline models are able to perform an accurate approximation of the parameter vectors, with the CNN providing the best inference. Based on this parameter distance criterion solely, the best results are obtained by the deterministic WAE model that outperforms traditional approaches. Although it would seem, at first, that our formulation only provides a marginal improvement on the parameters inference task, and that our proposal is even outperformed by baseline models, the analysis of the corresponding synthesized audio tells an entirely different story. Indeed, all AEs approaches strongly outperform the baseline models when it comes to audio accuracy, with the best results obtained with our probabilistic formulation $Flow_{post}$. These results show that, even though AE models do not provide an exact approximation of parameter vectors, they are able to account for the importance of these different parameters on the corresponding audio result. An even more interesting observation is that our proposed $Flow_{post}$ is outperformed by most baseline models on the parameters accuracy distance. However, it strongly outperforms all other methods on the resulting audio approximation accuracy. This supports our original hypothesis that learning the latent space of the synthesizer audio capabilities is a crucial component to understand its behavior. Furthermore, this might imply that our model can provide closely-sounding parameter settings based on the audio latent space, even though the parameters are quite different. This assumption is evaluated in the following section. Finally, this analysis also seems to be supported by the across-run variance, where probabilistic models obtain a more consistent accuracy, indicating that they provide a better generalization.

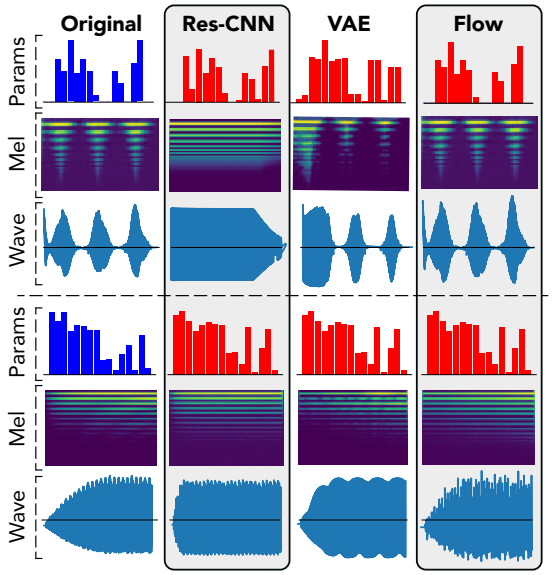

Figure 3: *Reconstruction analysis*. Comparing parameters inference and corresponding synthesized audio on the test dataset between the best performing models.

5.2. Reconstructions and latent space

We provide an in-depth analysis of the relations between inferred parameters and corresponding synthesized audio to support our previous claims. First, we selected two samples from the test set and compare the inferred parameters and synthesized audio in Figure 3.

As we can see, although the CNN provides a close inference of the parameters, the synthesized approximation completely misses important structural aspects, even in simpler instances as the slow ascending attack in the second example. This confirms that direct inference models are unable to assess the *relative impact* of parameters on the audio. Indeed, the errors in all parameters are considered equivalently, even though the same error magnitude on two different parameters can lead to dramatic differences in the synthesized audio. Oppositely, even though the parameters inferred by the VAE are quite far from the original preset, the corresponding audio is largely closer. This indicates that the latent space provides knowledge on the *audio-based neighborhoods* of the synthesizer. Therefore, this allows to understand the impact of different parameters in a given region of the latent audio space.

To evaluate this hypothesis, we encode two distant presets in the latent audio space and perform random sampling around these points to evaluate how local neighborhoods are organized. We also analyze the latent interpolation between those examples. The results are displayed in Figure 4.

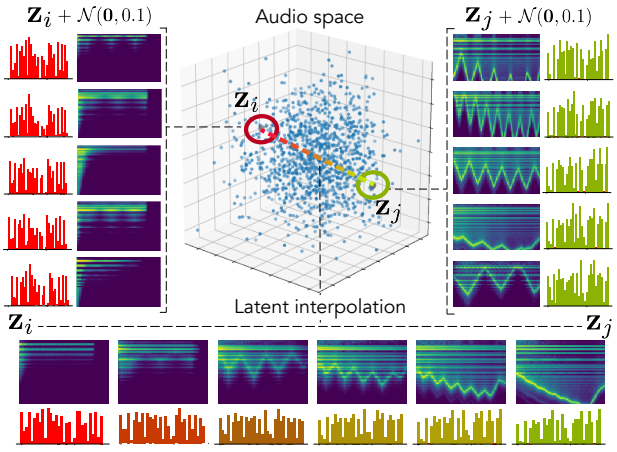

Figure 4: *Latent neighborhoods*. We select two examples from the test set that map to distant locations in the latent space $\mathbf{z}$ and perform random sampling in their local neighborhood to observe the parameters and audio. We also display the latent interpolation between those points.

As we can see, our hypothesis seems to be confirmed by the fact that neighborhoods are highly similar in terms of audio but have a larger variance in terms of parameters. Interestingly, this leads to complex but smooth non-linear dynamics in the parameters control.

5.3. Macro-parameters learning

Our formulation is the first to provide a continuous mapping between the audio $\mathbf{z}$ and parameter $\mathbf{v}$ spaces of a synthesizer. As latent VAE dimensions has been shown to disentangle major data variations, we hypothesized that we could directly use $\mathbf{z}$ as *macro-parameters* defining the most interesting variations in a given synthesizer. Hence, we introduce the new task of *macro-parameters learning* by mapping latent audio dimensions to parameters through $p(\mathbf{v}|\mathbf{z})$, which provides simplified control of the major audio variations for a given synthesizer. This is depicted in Figure 5

We show the two most informative latent dimensions $\mathbf{z}$ based on their variance. We study the traversal of these dimensions by keeping all other fixed at $\mathbf{0}$ to assess how $\mathbf{z}$ defines smooth macro-parameters through the mapping $p(\mathbf{v}|\mathbf{z})$. We report the evolution of the 5 parameters with highest variance (top), the corresponding synthesis (middle) and audio descriptors (bottom).

First, we can see that latent dimension corresponds to very smooth evolutions in terms of synthesized audio and descriptors. This is coherent with previous studies on the disentangling abilities of VAEs [6]. However, a very interesting property appear when we map to the parameter space. Although the parameters evolution is still smooth, it exhibits more non-linear relationships between different parameters. This correlates with the intuition that there are lots of complex interplays in parameters of a synthesizer. Our formulation allows to alleviate this complexity by automatically providing *macro-parameters* that are the most relevant to the audio variations of a given synthesizer. Here, we can see that the $\mathbf{z}_3$ latent dimension (left) seems to provide a *percussivity* parameter, where low values produce a very slow attack, while moving along this dimension, the attack becomes sharper and the amount of noise increases. Similarily, $\mathbf{z}_7$ seems to define an *harmonic densification* parameter, starting from a single peak frequency and increasingly adding harmonics and noise.

5.4. Creative applications

Our proposal allows to perform a direct exploration of presets based on audio similarity. Indeed, as the flow is *invertible*, we can map parameters to the audio space for exploration, and then back to parameters to obtain a new preset. Furthermore, this can be combined with *vocal sketch control* where the user inputs vocal imitations of the sound that he is looking for. This allows to quickly produce an approximation of the intended sound and then exploring the audio neighborhood of the sketch for intuitive refinement. We embedded our model inside a *MaxMSP* external called `flow_synth~` by using the *LibTorch* API and further integrate it into *Ableton Live* by using the *Max4Live* interface.

6. CONCLUSION

In this paper, we introduced several novel ideas including reformulating the problem of synthesizer control as matching the two latent space defined as the *user perception space* and the *synthesizer parameter space*. We showed that our approach outperforms all previous proposals on the seminal problem of *parameters inference*. Our formulation also naturally introduces the original tasks of *macro-control learning*, *audio-based preset exploration* and *semantic parameters discovery*. This proposal is the first to be able to simultaneously address most synthesizer control issues at once.

Altogether, we hope that this work will provide new means of exploring audio synthesis, sparkling the development of new leaps in musical creativity.

7. ACKNOWLEDGEMENTS

This work was supported by MAKIMOno project (ANR:17-CE38-0015-01 and NSERC:STPG 507004-17) and the ACTOR Partnership (SSHRC:895-2018-1023).

8. REFERENCES

[1] Miller Puckette, *The theory and technique of electronic music*, World Scientific Publishing Co., 2007.

[2] Mark Cartwright and Bryan Pardo, "Synthassist: an audio synthesizer programmed with vocal imitation,"

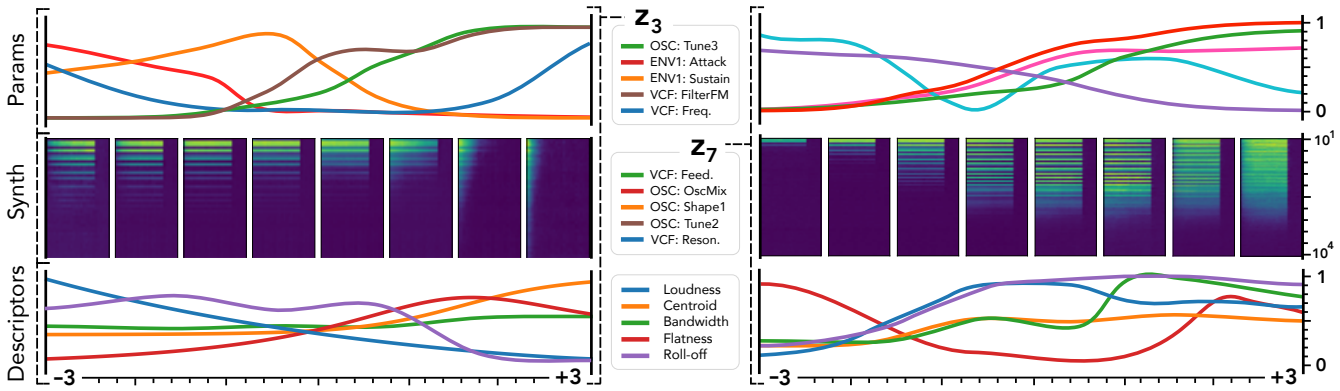

Figure 5: *Macro-parameters learning*. We show two of the learned latent dimensions $\mathbf{z}$ and compute the mapping $p(\mathbf{v}|\mathbf{z})$ when traversing these dimensions, while keeping all other fixed at $\mathbf{0}$ to see how $\mathbf{z}$ define smooth macro-parameters. We plot the evolution of the 5 parameters with highest variance (top), the corresponding synthesis (middle) and audio descriptors (bottom). (Left) $\mathbf{z}_3$ seems to relate to a *percussivity* parameter. (Right) $\mathbf{z}_7$ defines an *harmonic densification* parameter.

in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 741–742.

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

[4] Matthew John Yee-King, Leon Fedden, and Mark d'Inverno, "Automatic programming of vst sound synthesizers using deep networks and other techniques," *IEEE Transactions on ETCI*, vol. 2, no. 2, 2018.

[5] Diederik P. Kingma and Max Welling, "Auto-encoding variational bayes," *arXiv:1312.6114*, 2013.

[6] Irina Higgins, Loic Matthey, Arka Pal, Shakir Mohamed, and Alexander Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," *ICLR*, 2016.

[7] Danilo Rezende and Shakir Mohamed, "Variational inference with normalizing flows," in *International Conference on Machine Learning (ICML)*, 2015.

[8] Philippe Esling, Adrien Bitton, and Axel Chemla-Romeu-Santos, "Generative timbre spaces with variational audio synthesis," *21st International DaFX Conference, arXiv:1805.08501*, 2018.

[9] Christopher M. Bishop and Tom M. Mitchell, "Pattern recognition and machine learning," 2014.

[10] Casper K. Sønderby, Tapani Raiko, Lars Maaløe, Søren K. Sønderby, and Ole Winther, "How to train deep variational autoencoders and probabilistic ladder networks," *arXiv preprint arXiv:1602.02282*, 2016.

[11] Xi Chen, Diederik P Kingma, Tim Salimans, Ilya Sutskever, and Pieter Abbeel, "Variational lossy autoencoder," *International Conference on Learning Representations (ICLR)*, 2016.

[12] Ilya Tolstikhin, Olivier Bousquet, and Bernhard Schölkopf, "Wasserstein auto-encoders," *International Conference on Learning Representations*, 2017.

[13] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling, "Improved variational inference with inverse autoregressive flow," in *Advances in NIPS*, 2016, pp. 4743–4751.

[14] George Papamakarios, Theo Pavlakou, and Iain Murray, "Masked autoregressive flow for density estimation," in *NIPS*, 2017, pp. 2338–2347.