SING: Symbol-to-Instrument Neural Generator

Alexandre Défossez
Facebook AI Research
INRIA / ENS
PSL Research University
Paris, France
defossez@fb.com

Neil Zeghidour
Facebook AI Research
LSCP / ENS / EHESS / CNRS
INRIA / PSL Research University
Paris, France
neilz@fb.com

Nicolas Usunier
Facebook AI Research
Paris, France
usunier@fb.com

Léon Bottou
Facebook AI Research
New York, USA
leonb@fb.com

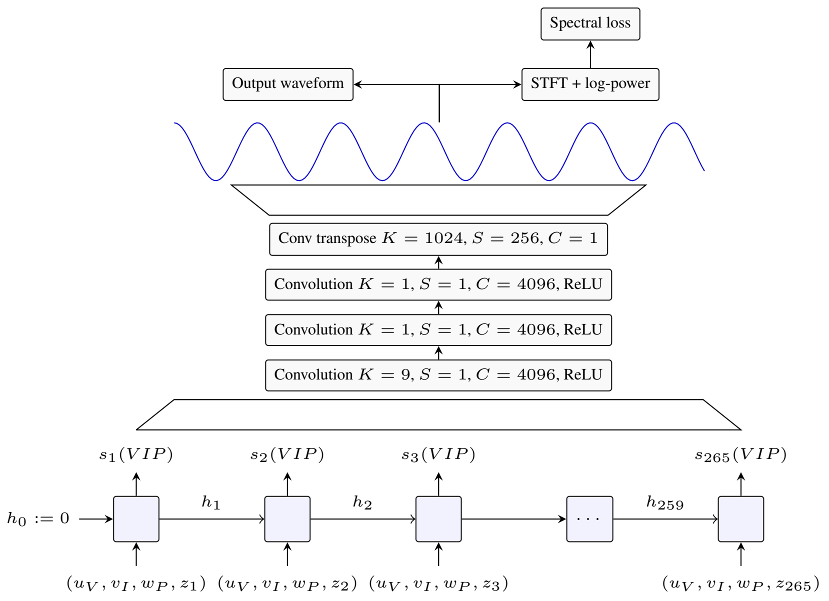
Francis Bach
INRIA
École Normale Supérieure
PSL Research University
francis.bach@ens.fr

Abstract

Recent progress in deep learning for audio synthesis opens the way to models that directly produce the waveform, shifting away from the traditional paradigm of relying on vocoders or MIDI synthesizers for speech or music generation. Despite their successes, current state-of-the-art neural audio synthesizers such as WaveNet and SampleRNN [24, 17] suffer from prohibitive training and inference times because they are based on autoregressive models that generate audio samples one at a time at a rate of 16kHz. In this work, we study the more computationally efficient alternative of generating the waveform frame-by-frame with large strides. We present SING, a lightweight neural audio synthesizer for the original task of generating musical notes given desired instrument, pitch and velocity. Our model is trained end-to-end to generate notes from nearly 1000 instruments with a single decoder, thanks to a new loss function that minimizes the distances between the log spectrograms of the generated and target waveforms. On the generalization task of synthesizing notes for pairs of pitch and instrument not seen during training, SING produces audio with significantly improved perceptual quality compared to a state-of-the-art autoencoder based on WaveNet [4] as measured by a Mean Opinion Score (MOS), and is about 32 times faster for training and $2,500$ times faster for inference.

1 Introduction

The recent progress in deep learning for sequence generation has led to the emergence of audio synthesis systems that directly generate the waveform, reaching state-of-the-art perceptual quality in speech synthesis, and promising results for music generation. This represents a shift of paradigm with respect to approaches that generate sequences of parameters to vocoders in text-to-speech systems [21, 23, 19], or MIDI partition in music generation [8, 3, 10]. A commonality between the state-of-the-art neural audio synthesis models is the use of discretized sample values, so that an audio sample is predicted by a categorical distribution trained with a classification loss [24, 17, 18, 14]. Another significant commonality is the use of autoregressive models that generate samples one-by-one, which leads to prohibitive training and inference times [24, 17], or requires specialized implementations and low-level code optimizations to run in real time [14]. An exception is parallel WaveNet [18] which generates a sequence with a fully convolutional network for faster inference. However, the parallel



Spectral loss
Output waveform
STFT + log-power
Conv transpose $K = 1024, S = 256, C = 1$
Convolution $K = 1, S = 1, C = 4096$, ReLU
Convolution $K = 1, S = 1, C = 4096$, ReLU
Convolution $K = 9, S = 1, C = 4096$, ReLU
$s_1(VIP)$
$s_2(VIP)$
$s_3(VIP)$
$s_{265}(VIP)$
$h_0 := 0$
h_1
h_2
h_{259}
(u_V, v_I, w_P, z_1)
(u_V, v_I, w_P, z_2)
(u_V, v_I, w_P, z_3)
(u_V, v_I, w_P, z_{265})

Figure 1: Summary of the entire architecture of SING. u_V, v_I, w_P, z_* represent the look-up tables respectively for the velocity, instrument, pitch and time. h_* represent the hidden state of the LSTM and s_* its output. For convolutional layers, K represents the kernel size, S the stride and C the number of channels.

generates a sequence $\forall 1 \leq T \leq N, s(V, I, P)_T \in \mathbb{R}^D$ with a linear layer on top of the last hidden state. In our experiments, we have $D = 128$ and $N = 265$.

4.2 Convolutional decoder

The sequence $s(V, I, P)$ is decoded into a waveform by a convolutional network. The first layer is a convolution with a kernel size of 9 and a stride of 1 over the sequence s with 4096 channels followed by a ReLU. The second and third layers are both convolutions with a kernel size of 1 (a.k.a. 1x1 convolution [4]) also followed by a ReLU. The number of channels is kept at 4096. Finally the last layer is a transposed convolution with a stride of 256 and a kernel size of 1024 that directly outputs the final waveform corresponding to an audio frame of size 1024. In order to reduce artifacts generated by the high stride value, we smooth the deconvolution filters by multiplying them with a squared Hann window. As the stride is one fourth of the kernel size, the squared Hann window has the property that the sum of its values for a given output position is always equal to 1 [7]. Thus the final deconvolution can also be seen as an overlap-add method. We pad the examples so that the final generated audio signal has the right length. Given our parameters, we need $s(V, I, P)$ to be of length $N = 265$ to recover a 4 seconds signal $d(s(V, I, P)) \in \mathbb{R}^{64{,}000}$.

4.3 Training details

All the models are trained on 4 P100 GPUs using Adam [15] with a learning rate of 0.0003 and a batch size of 256.

Initialization with an autoencoder. We introduce an encoder turning a waveform x into a sequence $e(x) \in \mathbb{R}^{N \times D}$. This encoder is almost the mirror of the decoder. It starts with a convolution layer with a kernel size of 1024, a stride of 256 and 4096 channels followed by a ReLU. Similarly to the decoder, we smooth its filters using a squared Hann window. Next are two 1x1 convolutions with 4096 channels and ReLU as an activation function. A final 1x1 convolution with no non linearity turns those 4096 channels into the desired sequence with D channels. We first train the encoder and decoder together as an auto-encoder on a reconstruction task. We train the auto-encoder for 50 epochs which takes about 12 hours on 4 GPUs.