

2023 September 24

Deep Learning for Music Analysis and Generation

Music Synthesis

generating audio clips of single notes or loops
 $(x \rightarrow \text{audio})$



Yi-Hsuan Yang Ph.D.
yhyangtw@ntu.edu.tw

Objectives

- We have learned two *conditional* audio generation tasks with **strong, temporally-aligned** conditions
 - **Source separation:** “audio (mixture) → audio (stem)” [*perfect alignment*]
 - **Mel-vocoder:** “mel → audio (stem)” [*output length is a factor of the input length*]
- Let’s consider two *simpler* audio **generation** task with **no** or only **class-level** conditions
 - **Music synthesis:** “MIDI pitch → audio” [**class-level global condition**]
 - **Unconditional generation** (of loops): “x → audio” [**no condition**]
 - **Weaker** condition: **No “ground-truth” target**
 - **Cannot use reconstruction loss**
 - **Diversity** can be important

Outline

- **Non-neural music synthesizers**
- DL-based timbre synthesis (e.g., generation of single notes)
- DL-based generation of loops
- HW2

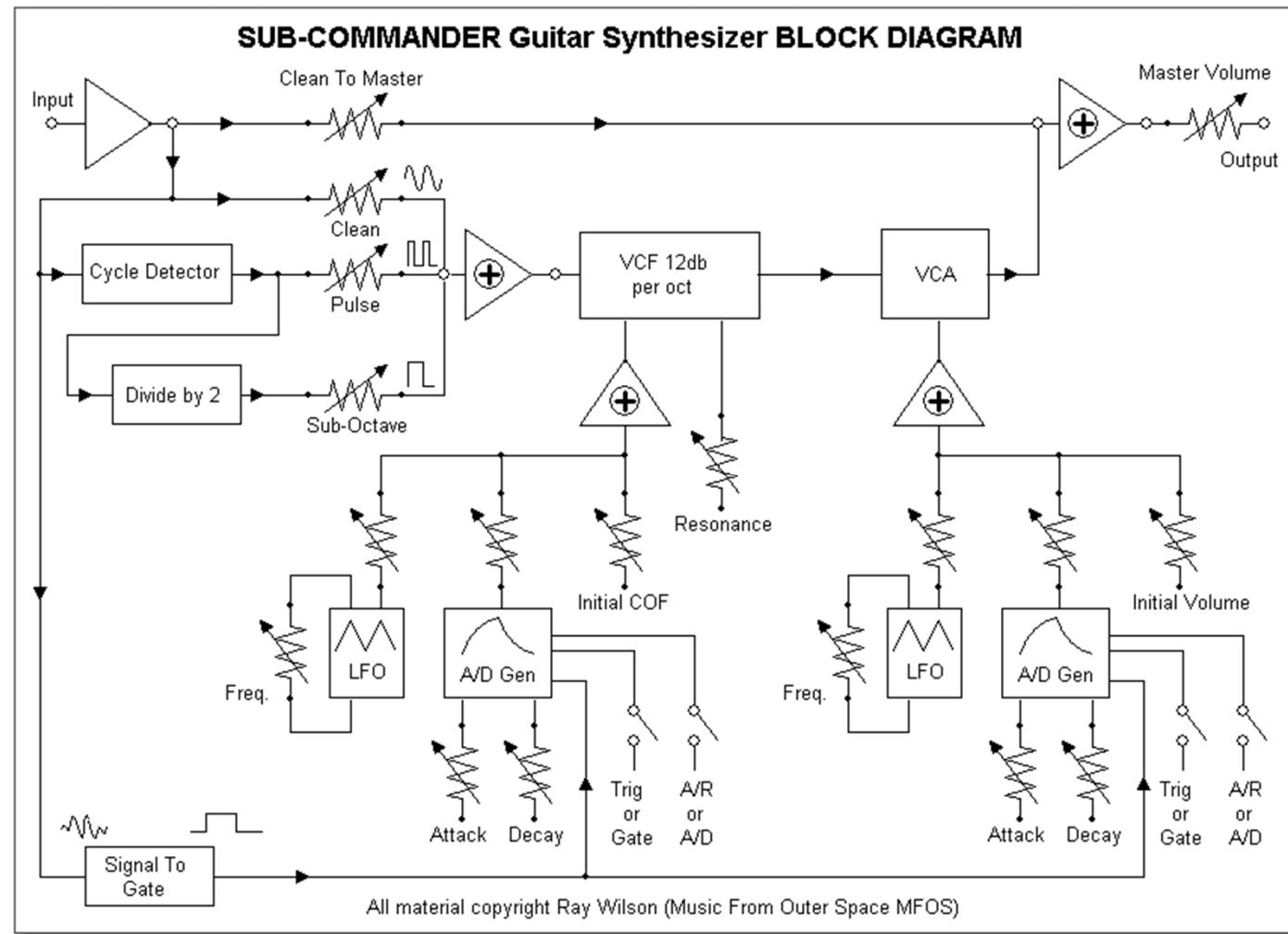
Synthesizers

<https://en.wikipedia.org/wiki/Synthesizer>

- An electronic musical instrument that generates audio signals
 - **Analog:** circuits, voltage-controlled oscillator
 - **Digital:** through DSP
 - Yamaha **DX7:** based on frequency modulation (FM) synthesis developed



Synthesizers via Circuits



Synthesizers for MIDI-to-Audio Rendering

<https://www.fluidsynth.org/>

<https://github.com/bzamecnik/midi2audio>

- Useful when working on symbolic-domain music generation

```
from midi2audio import FluidSynth
```

Play MIDI:

```
FluidSynth().play_midi('input.mid')
```

Synthesize MIDI to audio:

```
# using the default sound font in 44100 Hz sample rate
fs = FluidSynth()
fs.midi_to_audio('input.mid', 'output.wav')
```

FluidSynth

A SoundFont Synthesizer

FluidSynth is a real-time software synthesizer based on the SoundFont 2 specifications and has reached widespread distribution. FluidSynth itself does not have a graphical user interface, but due to its powerful API several applications utilize it and it has even found its way onto embedded systems and is used in some mobile apps.

Synthesizers for Sound Design

- Go beyond simulating the sounds of an acoustic instrument
- Widely used in EDM



Figure from: <https://professionalcomposers.com/synth-pad-quick-guide/>



Figure from: <https://getthatprosound.com/10-top-synths-for-modern-electronic-music-production-plus-programming-tips/>

DSP-based Synthesizers

<https://en.wikipedia.org/wiki/Synthesizer>

- Different types of synthesizers
 - **Subtractive synthesis:** complex waveforms are generated by oscillators and then shaped with filters to remove or boost specific frequencies
 - **Additive synthesis:** a large number of waveforms, usually sine waves, are combined into a composite sound
 - **FM synthesis:** a carrier wave is modulated with the frequency of a modulator wave
 - **Wavetable synthesis:** synthesizers modulate smoothly between digital representations of different waveforms, changing the shape and timbre
 - **Granular synthesis**
 - **Sample-based synthesizer**
 - **Physical modelling synthesis**

DSP-based Synthesizers

<https://www.coursera.org/learn/audio-signal-processing>

<https://github.com/MTG/sms-tools>

Week 1: Introduction; basic mathematics

Week 2: Discrete Fourier transform

Week 3: Fourier transform properties

Week 4: Short-time Fourier transform

Week 5: Sinusoidal model

Week 6: Harmonic model

Week 7: Sinusoidal plus residual modeling

Week 8: Sound transformations

Week 9: Sound/music description

Week 10: Concluding topics; beyond audio signal processing

DSP-based Synthesizers

<https://cs.gmu.edu/~sean/book/synthesis/>

Topics in the Book

1. **Introduction.** Synthesizer basics, usage environments, basic history.
2. **Representations of sound.** Units of measure, issues in digitization of signals.
3. **Additive synthesis.** Additive implementation, monophony and polyphony.
4. **Modulation.** LFOs, envelopes, sequencers and drum machines, arpeggiation, modulation matrices, MIDI modulation.
5. **Subtractive synthesis.** Oscillators, antialiasing and Nyquist, wave shaping, wave folding, phase distortion, combination and amplification.
6. **Digital filters.** Transfer functions, poles and zeros, magnitude and phase response, Laplace domain, Z domain and the Bilinear transform, second-order filters, filter composition, first-order filters, fourth-order ladder filters, formants.
7. **Frequency Modulation synthesis.** Phase modulation, sidebands, Bessel functions and reflection, operators and algorithms.
8. **Sampling.** Pulse code modulation, wavetable synthesis, granular synthesis, resampling techniques, real-time interpolation.
9. **Effects.** Delays, flangers, chorus, reverb, phasers, physical modeling synthesis.
10. **Controllers.** MIDI protocols.
11. **The Fourier Transform.** DFT and FFT, windowing, STFT.

Sample-based Synthesis

- **Vocaloid** by Xavier Serra

https://en.wikipedia.org/wiki/Xavier_Serra

Daisy was a collaboration project with Yamaha. The aim of the project was to develop a singing voice synthesizer in which the user would input the lyrics and the notes of a vocal melody and obtain a synthetic performance of a virtual singer. To synthesize such performance the system concatenates a chain of elemental synthesis units. These units are obtained by transposing and time-scaling samples from singers databases. These databases are created out of recording, analyzing, labeling and storing singers performing in as many different musical and phonetic contexts as possible.

Based on Daisy's research, Yamaha released a product named Vocaloid. Vocaloid was presented at the 114th Audio Engineering Society (AES) Convention in Amsterdam in March 2003 and had a big media impact leaded by The New York Times article "Could I Get That Song in Elvis, Please?". The Vocaloid synthesizer was Nominee for the European IST (Information Society Technologies) Prize 2005 and is the choice of leading artists including Mike Oldfield.

The research carried out for Daisy is mainly described in Bonada and Loscos (2003), Bonada et al. (2003), and Bonada et al (2001).

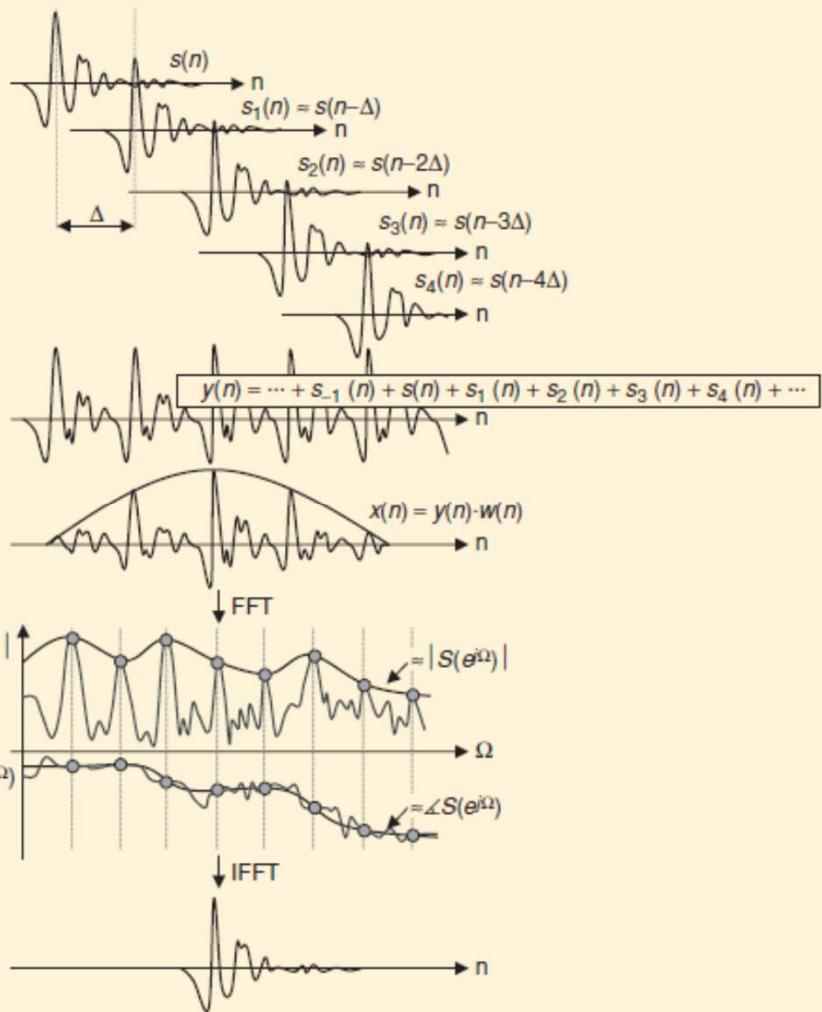


Source:
Alex Loscos,
*Spectral Processing
of the Singing Voice*,
PhD Thesis, UPF,
Spain (supervised
by Xavier Serra)

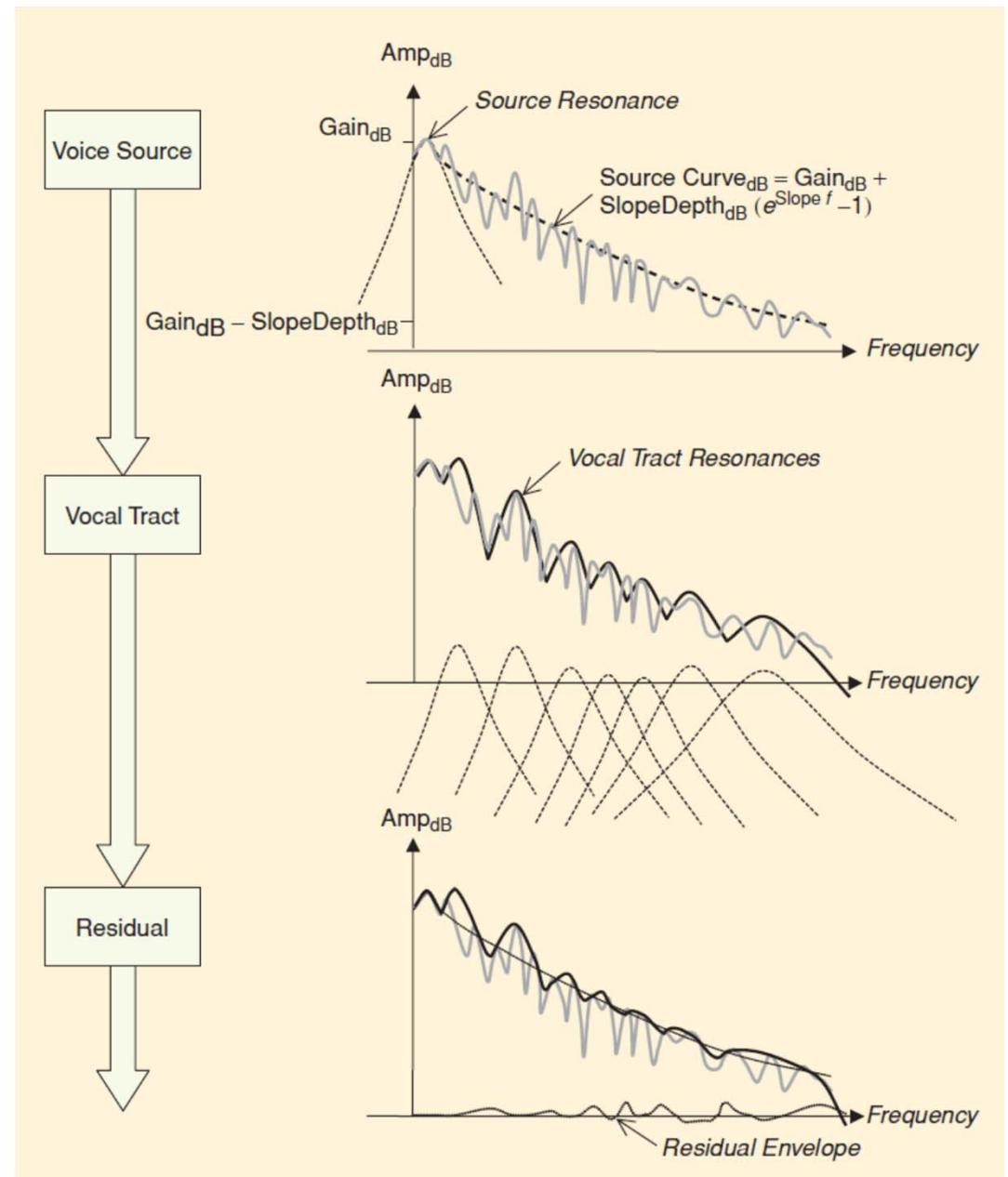
(Bonada and Loscos, 2003) “Sample-based singing voice synthesizer by spectral concatenation”

``The singing synthesis system we present generates a performance of an artificial singer out of the musical score and the phonetic transcription of a song using a frame-based frequency domain technique. This performance mimics the real singing of a singer that has been previously recorded, analyzed and stored in a database, in which we store his voice characteristics (phonetics) and his low-level expressivity (attacks, releases, note transitions and vibratos). **To synthesize such performance the systems concatenates a set of elemental synthesis units (phonetic articulations and stationeries). These units are obtained by transposing and time-scaling the database samples.** The concatenation of these transformed samples is performed by spreading out the spectral shape and phase discontinuities of the boundaries along a set of transition frames that surround the joint frames. [...]``

Source: <http://mtg.upf.edu/node/322>



<https://ieeexplore.ieee.org/document/4117930>



Sample-based Synthesis: Ample Sound

<http://www.amplesound.net/en/index.asp>

- Challenges:
 - Great manual effort in recording
 - Every note, velocity, playing technique, etc
 - Note-note transition



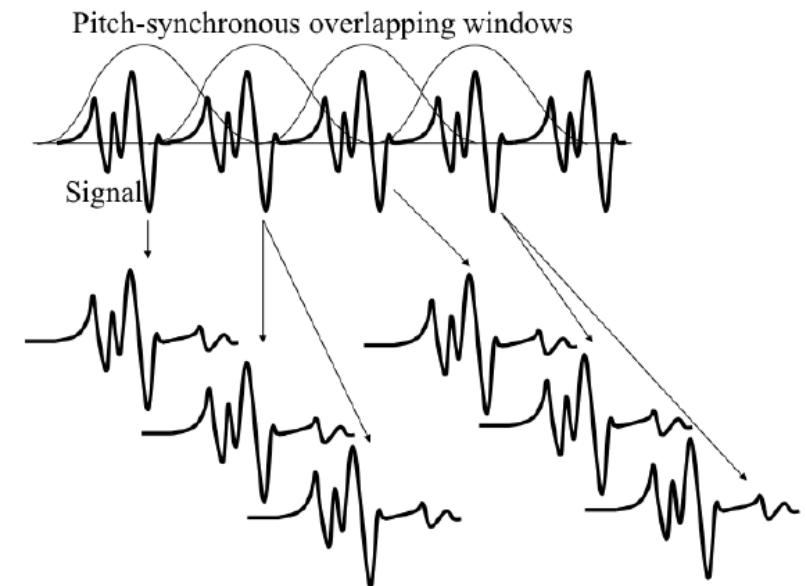
Overlap-and-Add for Audio Time Stretching and Pitch Scaling

https://en.wikipedia.org/wiki/Audio_time_stretching_and_pitch_scaling

<https://en.wikipedia.org/wiki/PSOLA>

- **PSOLA:** Pitch synchronous overlap-and-add

- Divide the waveform into small overlapping segments
- To **change the pitch**, the segments are *moved further apart* (to decrease the pitch) or *closer together* (to increase the pitch)
- To **change the duration**, the segments are then *repeated* multiple times (to increase the duration) or some are *eliminated* (to decrease the duration)



Audio Signal Processing by PyTSM

(Audio Time Stretching and Pitch Scaling)

<https://seyong92.github.io/PyTSM-ISMIR2020LBD/>

Available TSM algorithms

- Overlap-Add (OLA)
 - OLA is the most simple TSM algorithm that changes the length of the signal through modifying the hop size between analysis frame and synthesis frame.
- Pitch Synchronous Overlap-Add (TD-PSOLA)
 - TD-PSOLA is the algorithm that analyzes the original waveforms to create pitch-synchronous analysis windows and synthesize the output signal both for modifying time-scale and pitch-scale.
- Waveform Similarity Overlap-Add (WSOLA)
 - WSOLA maximizes the waveform similarity by allowing timing tolerance to analysis frame to find the most similar position through cross correlation.
- Phase Vocoder (PV)
 - Phase vocoder estimates instantaneous frequency, and it is used to update phases of input signal's frequency components in short-time Fourier transform. Although TSM results with phase vocoder has high phase continuity, it causes transient smearing for percussive audio sources and a coloring artifact called phasiness.
- TSM based on harmonic-percussive source separation (HPTSM)
 - A novel TSM algorithm that applies phase vocoder to only harmonic sources and applying OLA to only percussive sources.

Neural Generation of Single Notes

- “MIDI pitch → audio”
 - Synthesis of **timbre**
 - Can be used in **MIDI-to-audio** rendering
-
- Can we learn such a mapping in a *data-driven* way via DL?
 - Can we learn a *latent embedding* space of sounds that allow for creative applications such as interpolation or extrapolation of sounds?

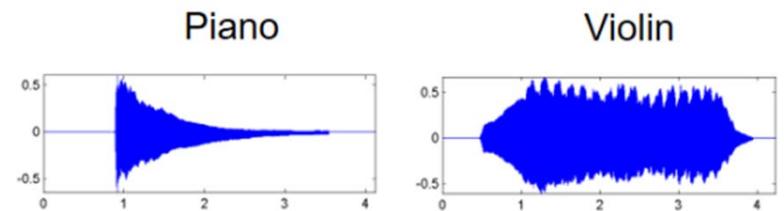


Figure from: <https://crazyoscarchang.github.io/2018/11/13/agent-embeddings/>

Neural Generation of Loops

- “ $x \rightarrow \text{audio}$ ”
- Loops
 - Repeating section of sound material
 - Widely used in many genres (e.g., hip hop)
- More challenging than note synthesis
 - Longer
 - May not have associated MIDI
- Less challenging than *general unconditional music generation*
 - Fixed number of bars (e.g., 4-bar)
 - Basic building block of music



Figure from: <https://www.magix.com/us/music-editing/music-production/effects/loops/>

Outline

- Non-neural music synthesizers
- **DL-based timbre synthesis (e.g., generation of single notes)**
 - WaveNet Autoencoder
 - GANSynth
 - RAVE
- DL-based generation of loops
- HW2

Autoregressive Models for Unconditional Audio Generation

- **WaveNet** (Van Den Oord et al., 2016)
- **SampleRNN** (Mehri et al., 2016)
- **WaveRNN** (Kalchbrenner et al., 2018)
- Inference with these models is inherently slow and inefficient because audio samples must be generated sequentially

WaveNet (arXiv'16)

In this paper we introduce a new generative model operating directly on the raw audio waveform. The joint probability of a waveform $\mathbf{x} = \{x_1, \dots, x_T\}$ is factorised as a product of conditional probabilities as follows:

$$p(\mathbf{x}) = \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1}) \quad (1)$$

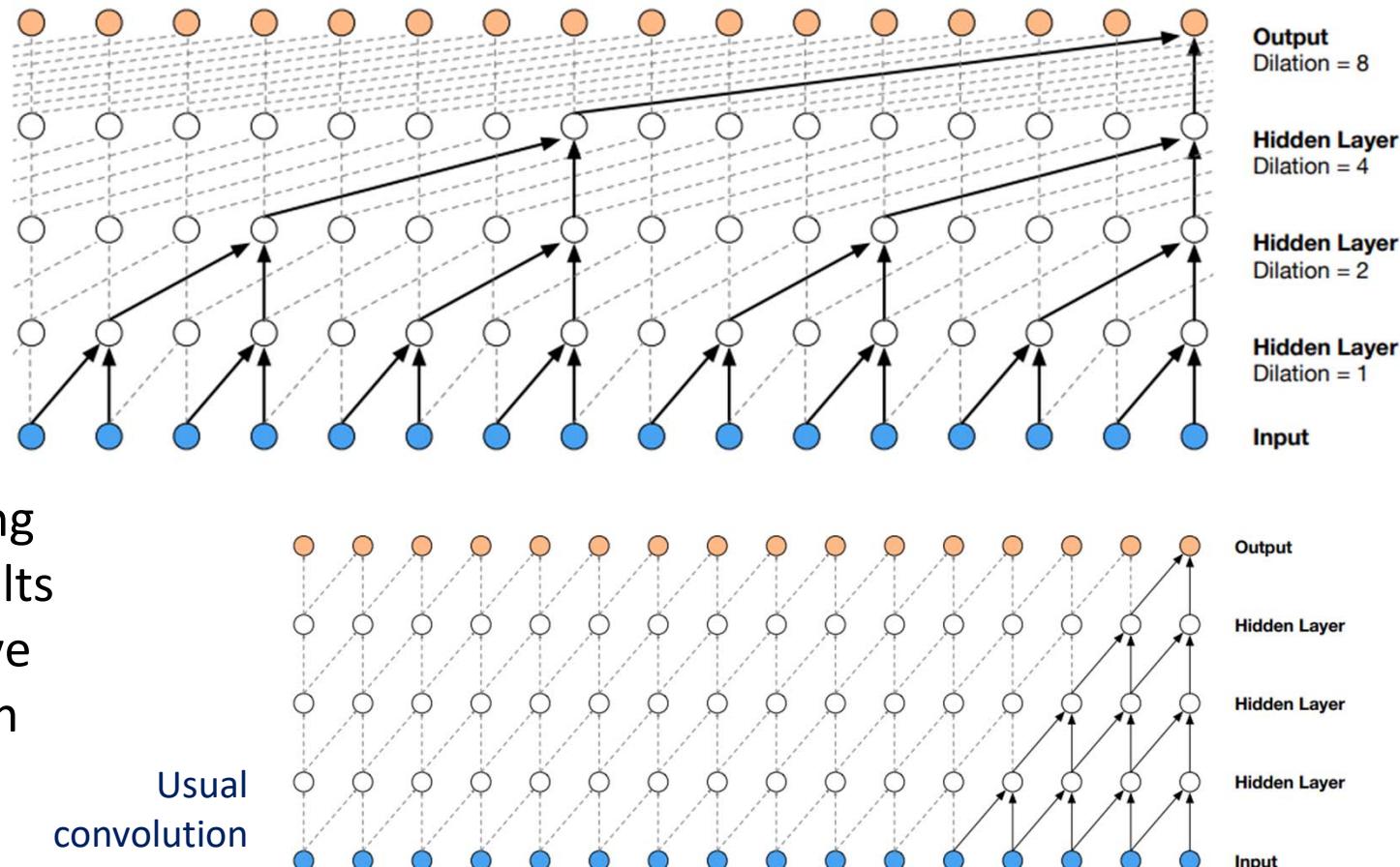
Each audio sample x_t is therefore conditioned on the samples at all previous timesteps.

- **Autoregressive; causal convolutions**
 - The prediction at timestep t cannot depend on any of the future timesteps
 - “At training time, the conditional predictions for all timesteps can be made in **parallel** because all timesteps of ground truth \mathbf{x} are known”
 - “When generating with the model, the predictions are **sequential**: after each sample is predicted, it is fed back into the network to predict the next sample”

WaveNet: Dilated Convolutions

- *Convolution with holes*

- “A convolution where the filter is applied over an area larger than its length by skipping input values with a certain step”



WaveNet: Output Design

Because raw audio is typically stored as a sequence of 16-bit integer values (one per timestep), a softmax layer would need to output 65,536 probabilities per timestep to model all possible values. To make this more tractable, we first apply a μ -law companding transformation (ITU-T, 1988) to the data, and then quantize it to 256 possible values:

$$f(x_t) = \text{sign}(x_t) \frac{\ln(1 + \mu |x_t|)}{\ln(1 + \mu)},$$

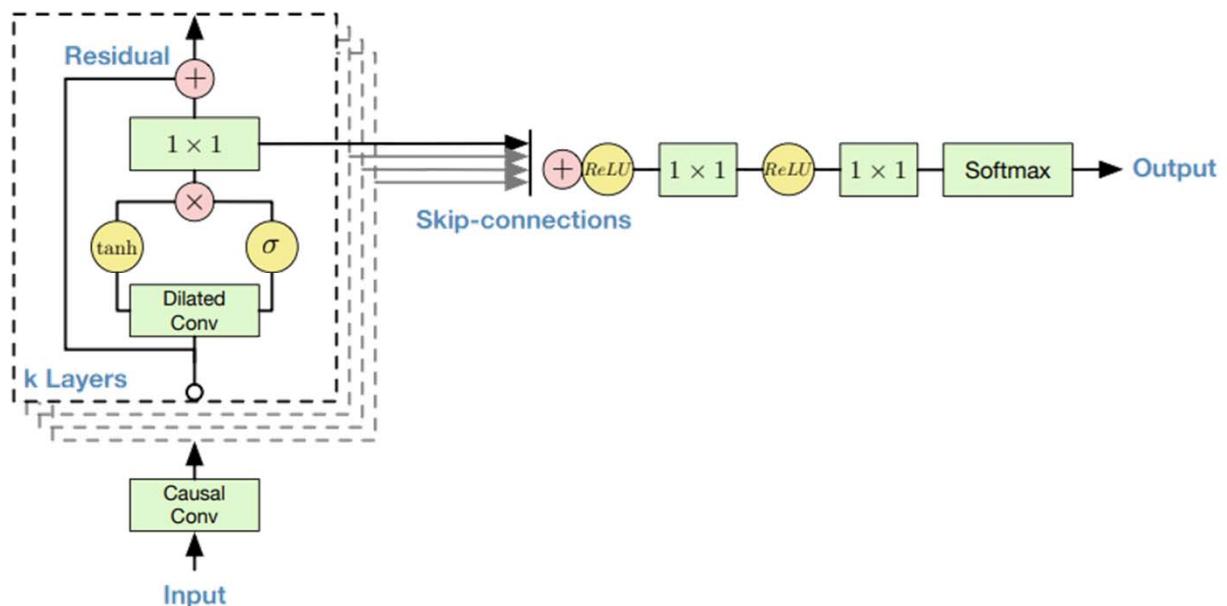
where $-1 < x_t < 1$ and $\mu = 255$. This non-linear quantization produces a significantly better reconstruction than a simple linear quantization scheme. Especially for speech, we found that the reconstructed signal after quantization sounded very similar to the original.

WaveNet: Gating

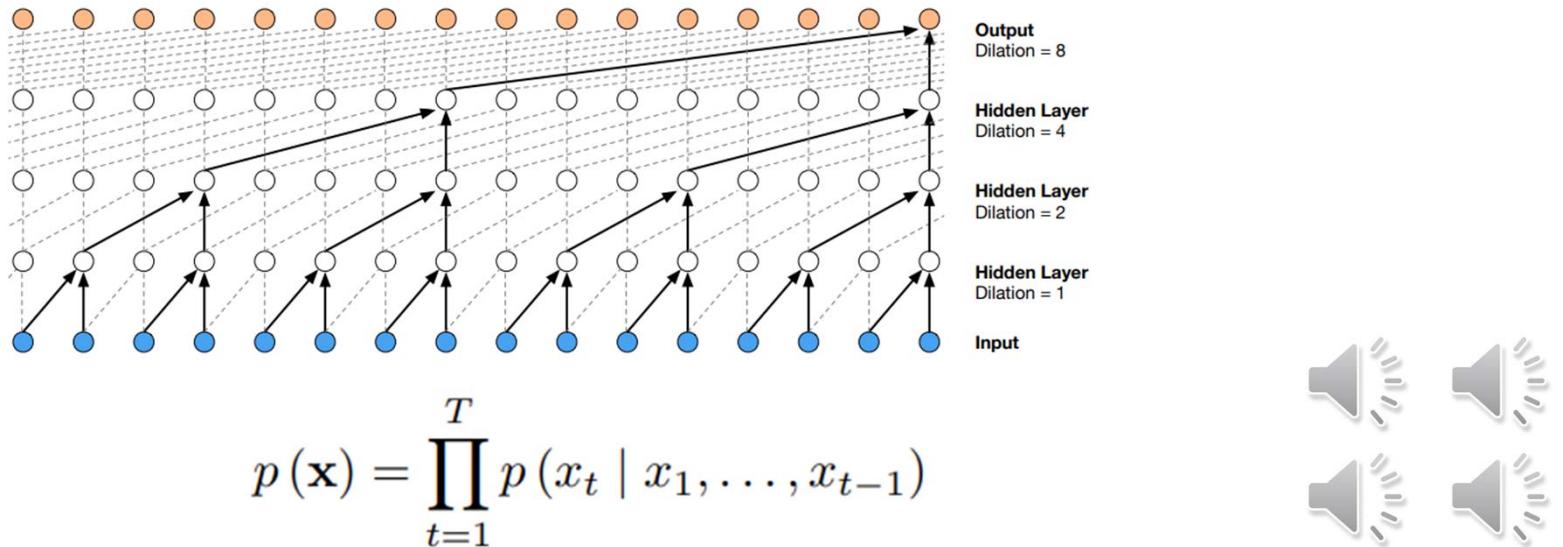
We use the same gated activation unit as used in the gated PixelCNN (van den Oord et al., 2016b):

$$\mathbf{z} = \tanh(W_{f,k} * \mathbf{x}) \odot \sigma(W_{g,k} * \mathbf{x}), \quad (2)$$

where $*$ denotes a convolution operator, \odot denotes an element-wise multiplication operator, $\sigma(\cdot)$ is a sigmoid function, k is the layer index, f and g denote filter and gate, respectively, and W is a learnable convolution filter. In our initial experiments, we observed that this non-linearity worked significantly better than the rectified linear activation function (Nair & Hinton, 2010) for modeling audio signals.



WaveNet: Generated Samples



- Unconditional generation from this model manifests as “babbling” due to the lack of longer term structure

- https://download.magenta.tensorflow.org/audio_examples/nsynth/UnconditionalWaveNetGeneration_WARNING_HIGH_VOLUME/0.mp3
- https://download.magenta.tensorflow.org/audio_examples/nsynth/UnconditionalWaveNetGeneration_WARNING_HIGH_VOLUME/1.mp3
- https://download.magenta.tensorflow.org/audio_examples/nsynth/UnconditionalWaveNetGeneration_WARNING_HIGH_VOLUME/2.mp3
- https://download.magenta.tensorflow.org/audio_examples/nsynth/UnconditionalWaveNetGeneration_WARNING_HIGH_VOLUME/3.mp3

WaveNet AutoEncoder (ICML'17)

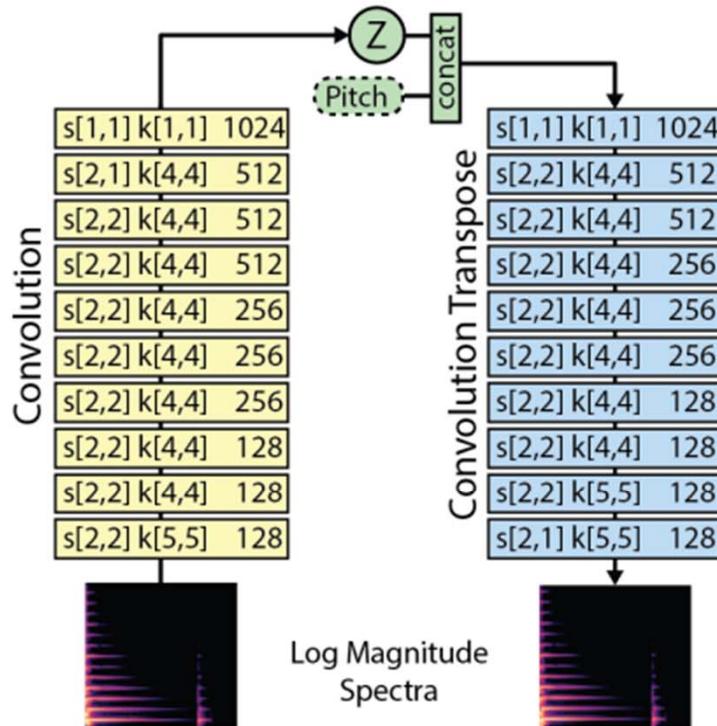
- Extension of WaveNet for “MIDI pitch → audio”
 - *WaveNet*: decoder-only; unconditional generation
 - *WaveNet Autoencoder*: encoder-decoder; conditional generation
- **WaveNet Autoencoder**
 - WaveNet-like **encoder** that infers hidden embeddings $Z = f(x)$ distributed in time
 - WaveNet **decoder** that uses Z to reconstruct the original audio autoregressively

$$p(x) = \prod_{i=1}^N p(x_i|x_1, \dots, x_{N-1}, f(x))$$

- At inference time, the embedding can be inferred deterministically from audio or drawn from other points in the embedding space, e.g., through interpolation

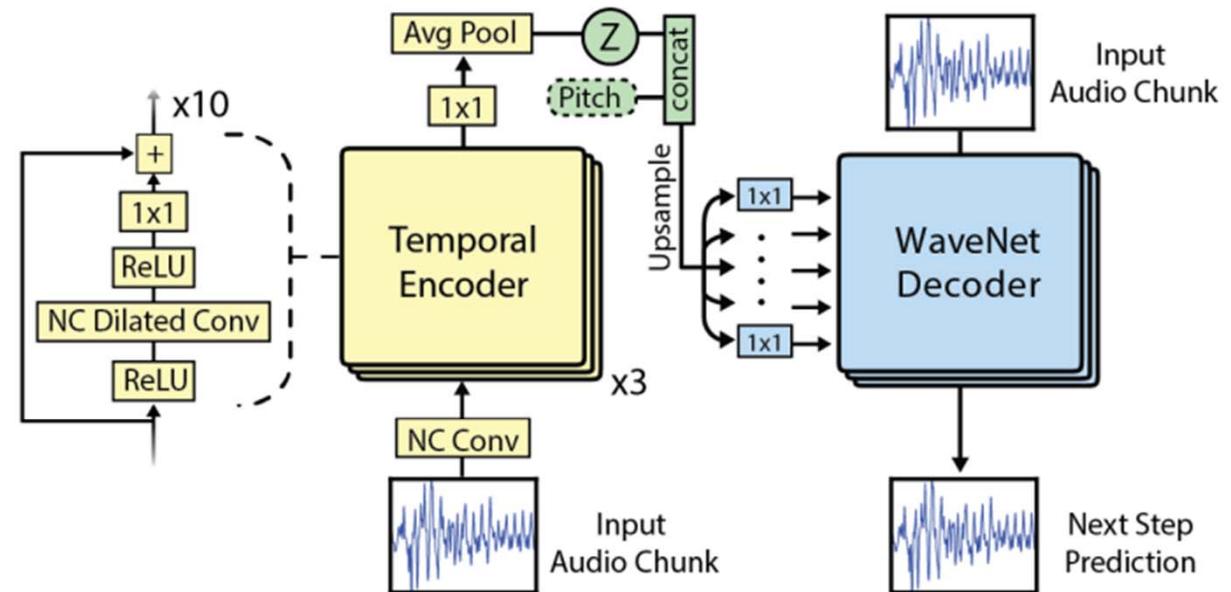
WaveNet AutoEncoder

a) Baseline Spectral Autoencoder



b)

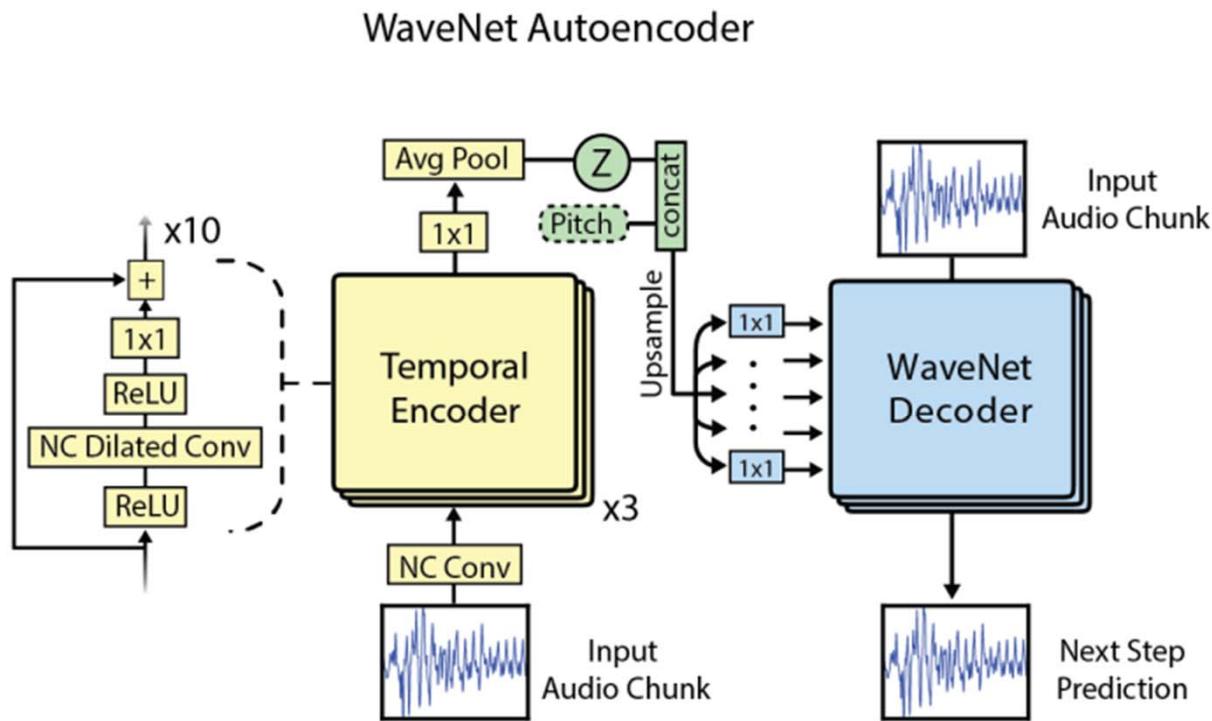
WaveNet Autoencoder



- **Pitch:** one-hot pitch representation
- For the spectral autoencoder, Griffin-Lim is used to reconstruct phase

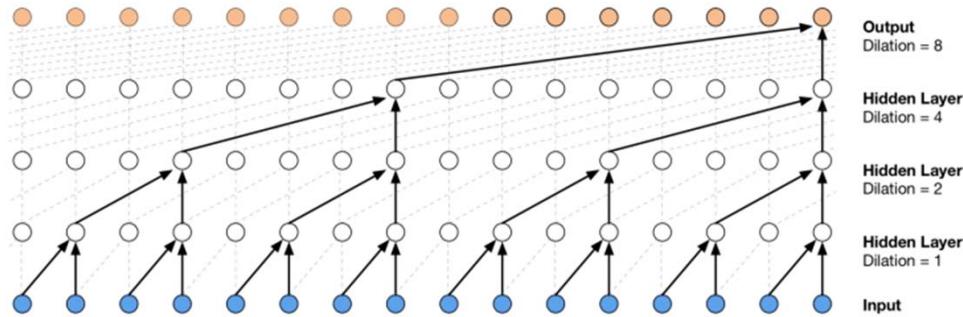
WaveNet AutoEncoder

- *WaveNet-like encoder: non-causal (NC), non-autoregressive*
- *WaveNet decoder: causal, autoregressive*

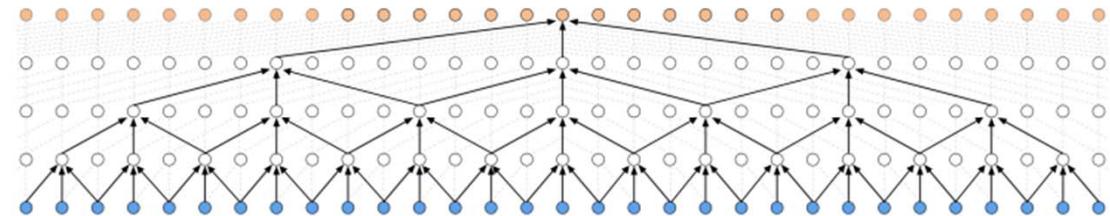


WaveNet AutoEncoder

- *WaveNet-like encoder: non-causal (NC), non-autoregressive*
- *WaveNet decoder: causal, autoregressive*



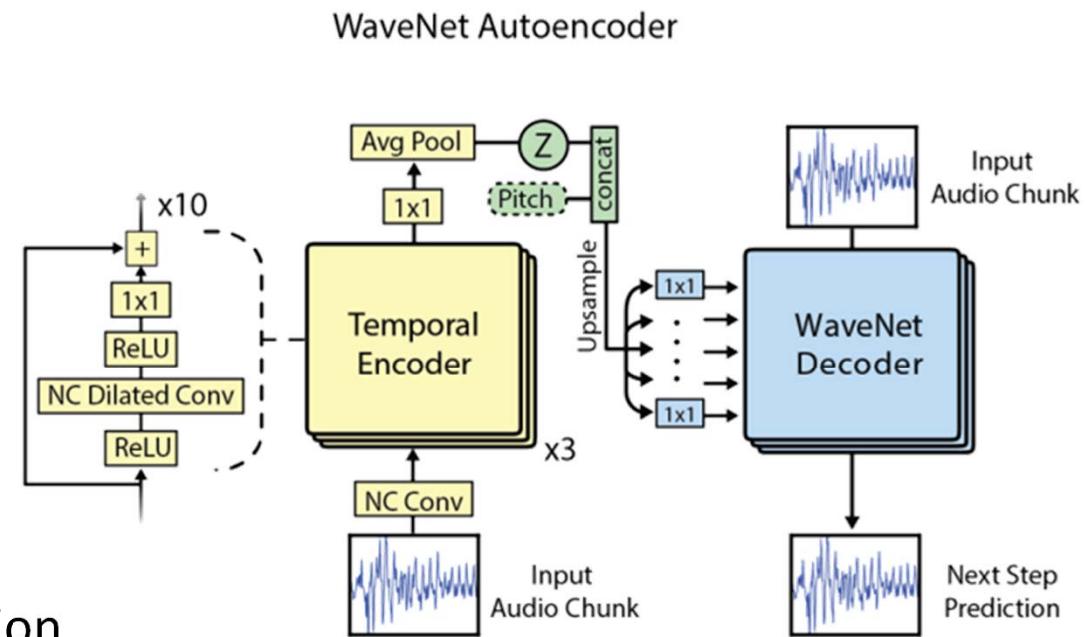
a stack of dilated causal convolutional layers



a stack of dilated non-causal convolutional layers

WaveNet AutoEncoder: Details

- Encoder
 - 30-layer nonlinear residual network
 - The result Z is a sequence of hidden codes with separate dimensions for time and channel
 - A stride of 512 (32ms) with 16 dimensions per timestep



- Decoder
 - Upsample Z to the original audio rate with nearest neighbor interpolation
 - Use that as a condition to bias every layer with a different 1×1 linear projection

Dataset: NSynth

<https://magenta.tensorflow.org/nsynth>

- Across a range of pitches, timbres, and volumes

Family	Source			Total
	ACOUST	ELECTR	SYNTH	
BASS	200	8387	60 368	68 955
BRASS	13 760	70	0	13 830
FLUTE	6572	70	2816	9458
GUITAR	13 343	16 805	5275	35 423
KEYBOARD	8508	42 709	3838	55 055
MALLET	27 722	5581	1763	35 066
ORGAN	176	36 401	0	36 577
REED	14 262	76	528	14 866
STRING	20 510	84	0	20 594
SYNTH LEAD	0	0	5501	5501
VOCAL	3925	140	6688	10 753
Total	108 978	110 224	86 777	306 043

Performance Evaluation on NSynth

- WaveNet is slow (autoregressive generation)
- And is outperformed by a later model called **GANSynth** in perceptual quality

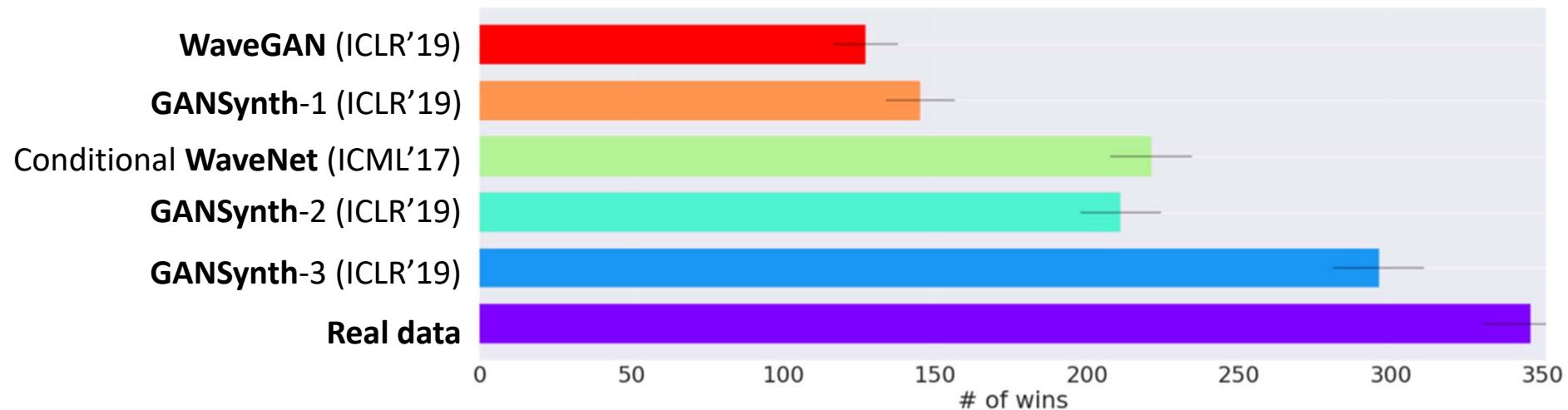


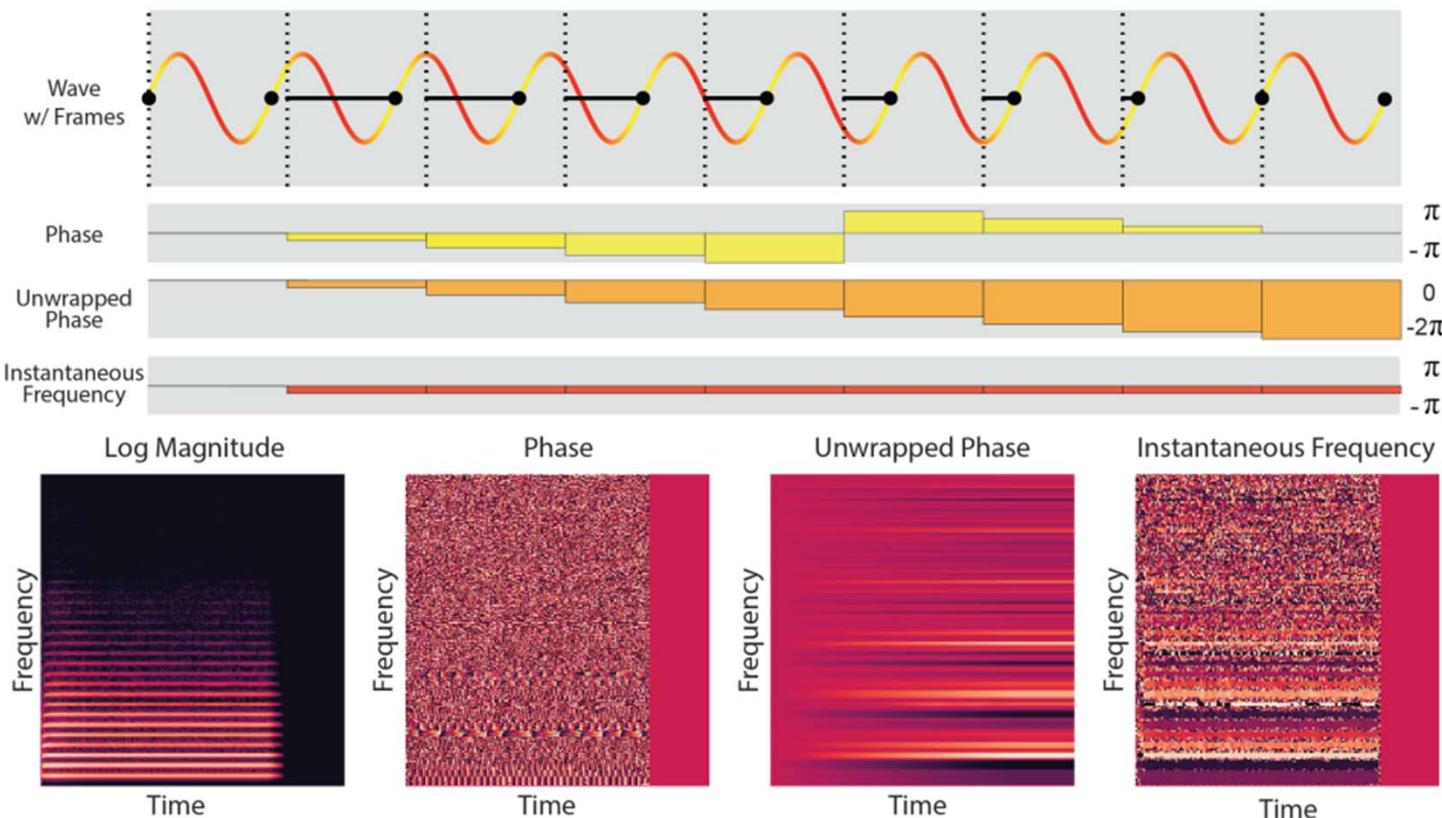
Figure 2: Number of wins on pair-wise comparison across different output representations and baselines. Ablation comparing highest performing models of each type. Higher scores represent better

GANSynth (ICLR'19)

- **Non-autoregressive** generation using GAN
- Generate **magnitude spectrogram** and **phase**, instead of waveforms
- Use *Progressive growing GAN* (ICLR'18)
- “On the NSynth dataset, GANs can outperform a strong WaveNet baseline in automatic and human evaluations, and generate examples **~54,000 times faster**”
- **Q:** WaveNet is slow because it’s autoregressive. Why don’t we use the non-autoregressive version?
- **A:** Sure, if the task is **audio-to-audio**; that’s called **TCN** (see lecture 3)

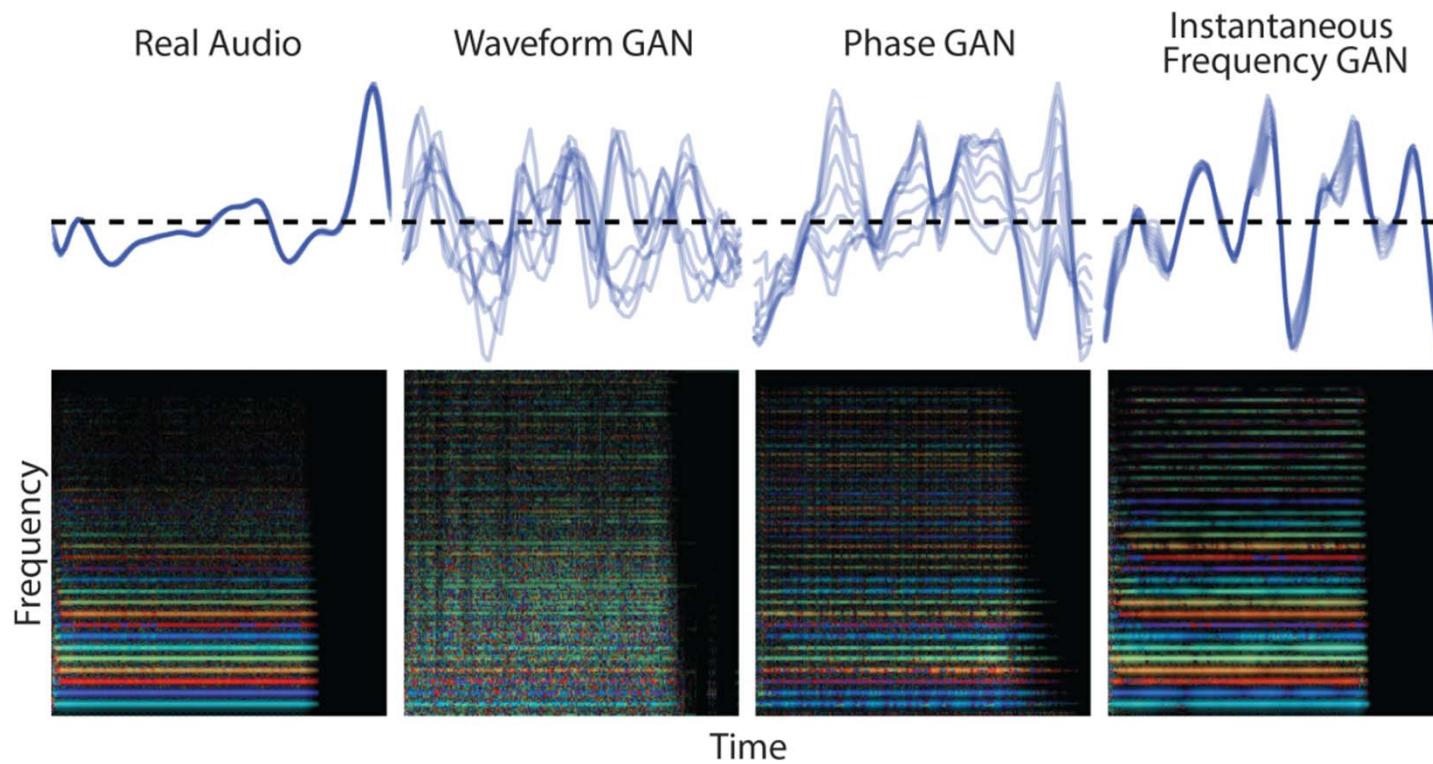
GANSynth (ICLR'19)

- Actually model the **instantaneous radial frequency (IF)**, not the phase
 - IF: unwrap the phase over the 2π boundary (orange boxes) and take its derivative



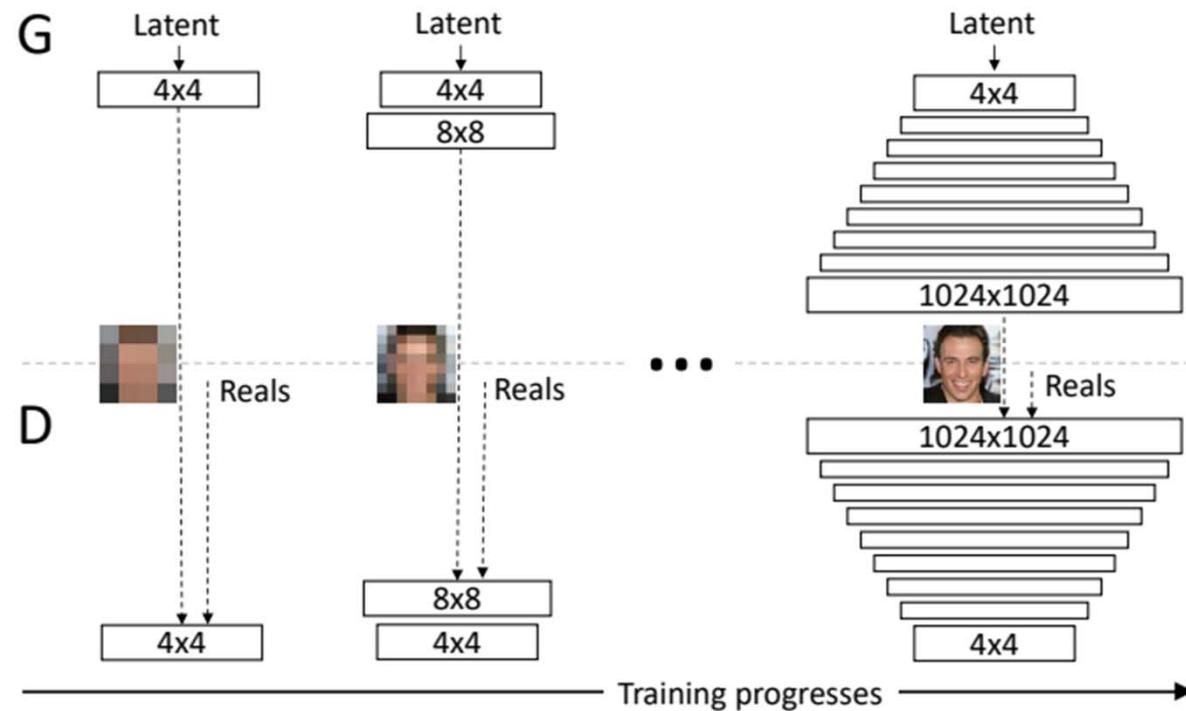
GANSynth (ICLR'19)

- Actually model the **instantaneous radial frequency (IF)**, not the phase
 - IF: unwrap the phase over the 2π boundary (orange boxes) and take its derivative



GANSynth (ICLR'19)

- Use *Progressive growing GAN* (PGGAN; ICLR'18)



Ref: Karras et al, “Progressive growing of GANs for improved quality, stability, and variation,” ICLR 2018

36

Performance Evaluation on NSynth

- Without “+H”
 - 1,024 window size, 75% overlaps
 - “images” of size (256, 512, 2)
- With “+H”
 - 2,048 window size, 75% overlaps
 - “images” of size (128, 1024, 2)

Examples	Human Eval					
	(wins)	NDB	FID	IS	PA	PE
Real Data	549	2.2	13	47.1	98.2	0.22
IF-Mel + H	485	29.3	167	38.1	97.9	0.40
IF + H	308	36.0	104	41.6	98.3	0.32
Phase + H	225	37.6	592	36.2	97.6	0.44
IF-Mel	479	37.0	600	29.6	94.1	0.63
IF	283	37.0	708	36.3	96.8	0.44
Phase	203	41.4	687	24.4	94.4	0.77
WaveNet	359	45.9	320	29.1	92.7	0.70
WaveGAN	216	43.0	461	13.7	82.7	1.40

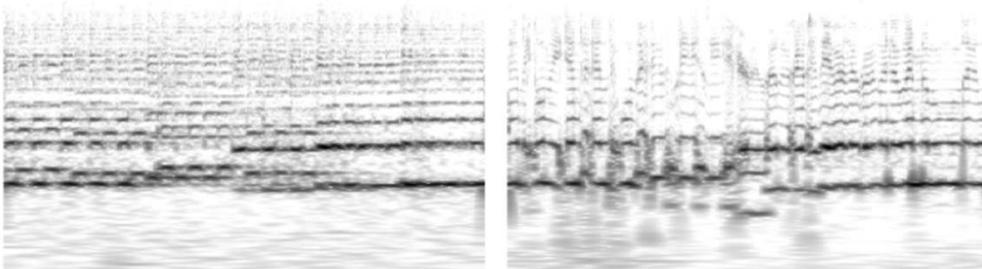
Recap: Generation of *single notes*

- “MIDI pitch → audio”
 - WaveNet autoencoder (ICML’17)
 - GANSynth (ICLR’19)
 - Many others; for example, DrumGAN (ISMIR’20)
 - Mainly about *timbre synthesis*
 - Studies on the generation of single notes seem no longer popular due to the advance of more powerful models, e.g., text-to-music generation
- However, the idea of *timbre synthesis* can **go beyond single notes**
 - **DDSP** (ICLR’20) and lots of DDSP-based approaches since then
 - **DDSP-piano**: <https://github.com/lrenault/ddsp-piano>
 - **RAVE** (arXiv’21)

RAVE (arXiv'21)

https://anonymous84654.github.io/RAVE_anonymous/

Strings to speech transfer



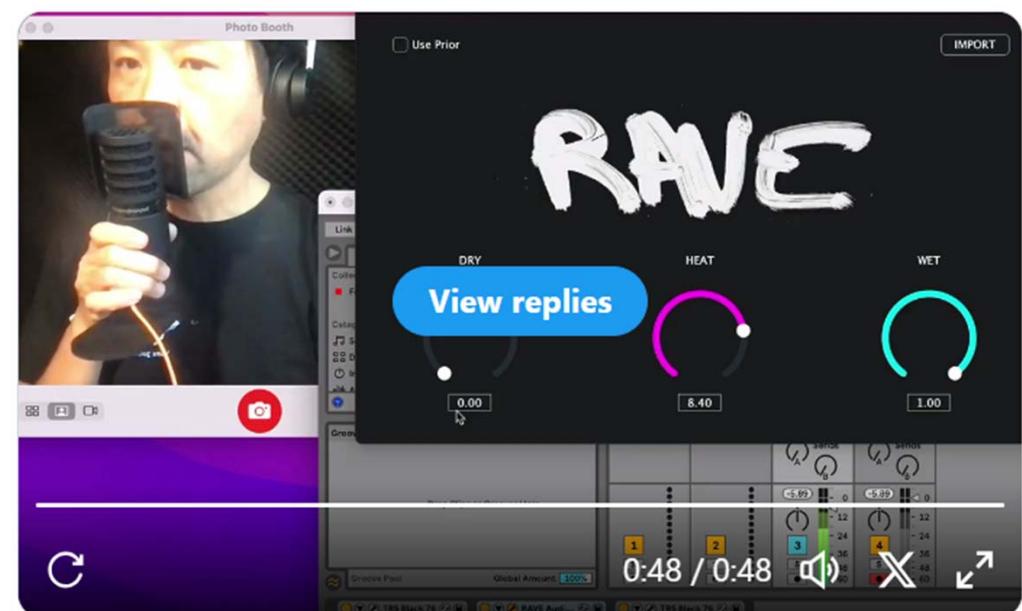
original

reconstructed

▶ 0:00 / 0:04 - 🔍 ⏰

▶ 0:00 / 0:04 - 🔍 ⏰

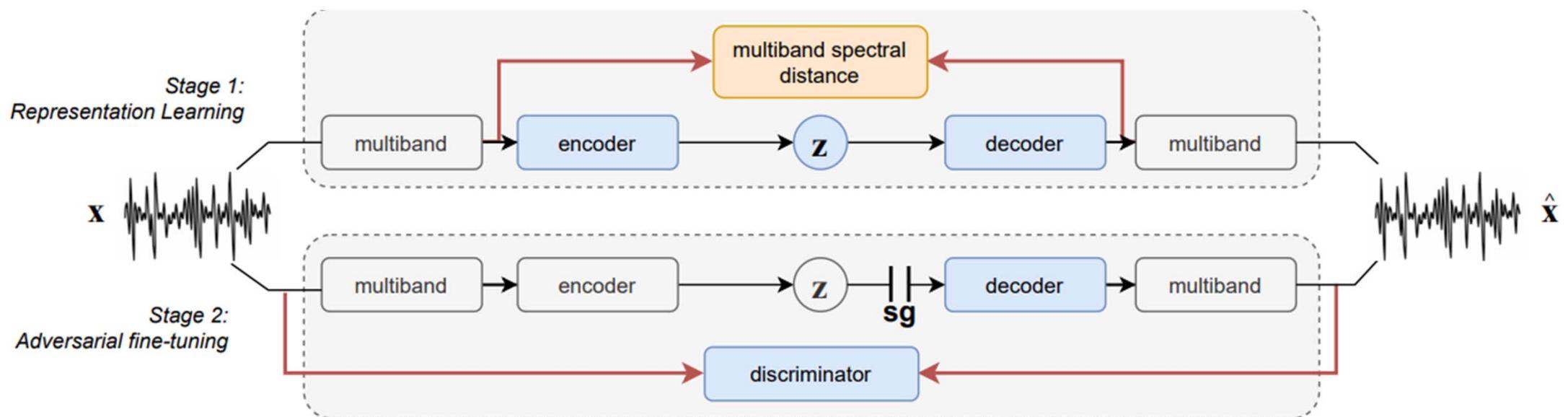
Realtime Voice to Darbuka transfer



Ref: Caillon and Esling, "RAVE: A variational autoencoder for fast and high-quality neural audio synthesis," arXiv 2021

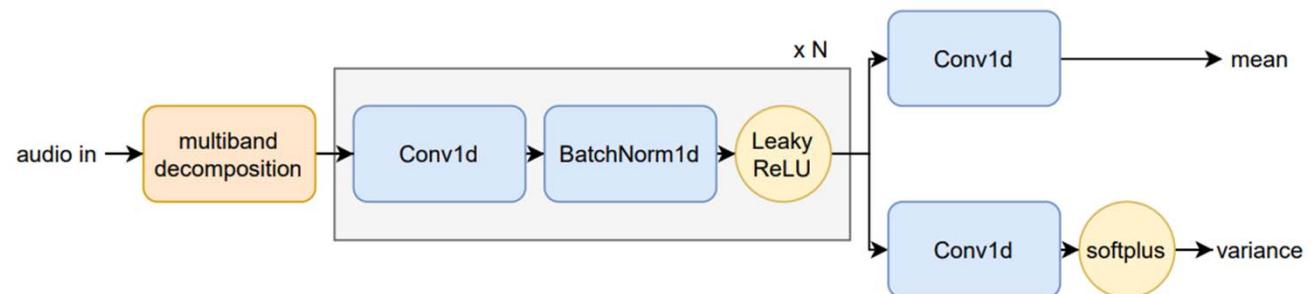
RAVE: VAE+GAN

- Two-stage training procedure
 - First trained as a **regular VAE** for representation learning
 - Then fine-tuned with an **adversarial** generation objective in order to achieve high quality audio synthesis

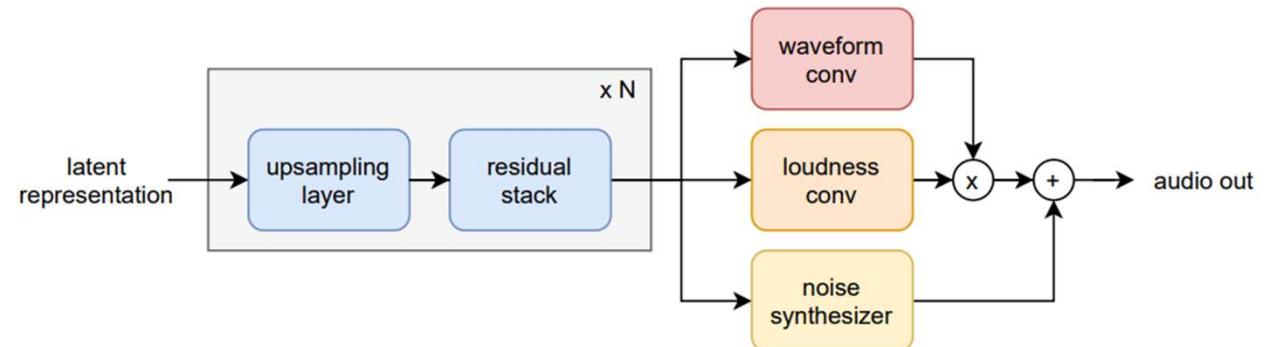


RAVE: VAE+GAN

- **Encoder:** multiband decomposition and outputs a 128-dim latent



- **Decoder** is a modified version of the generator of **MelGAN**
- **Discriminator**: Exactly the same discriminator as in **MelGAN**



RAVE: Real-time Generation

- Good for creative applications

Table 2: Comparison of the synthesis speed for several models

Model	CPU synthesis	GPU synthesis
NSynth	18 Hz	57 Hz
SING	304 kHz	9.8 MHz
RAVE (Ours) w/o multiband	38 kHz	3.7 MHz
RAVE (Ours)	985 kHz	11.7 MHz

Realtime usage ↗

This section presents how RAVE can be loaded inside `nn~` in order to be used live with Max/MSP or PureData.

RAVE: Being Actively Updated at IRCAM

(Institut de recherche et coordination acoustique/musique)

<https://github.com/acids-ircam/RAVE>

Type	Name	Description
Architecture	v1	Original continuous model
	v2	Improved continuous model (faster, higher quality)
	v3	v2 with Snake activation, descript discriminator and Adaptive Instance Normalization for real style transfer
	discrete	Discrete model (similar to SoundStream or EnCodec)
	onnx	Noiseless v1 configuration for onnx usage
	raspberry	Lightweight configuration compatible with realtime RaspberryPi 4 inference
Regularization (v2 only)	default	Variational Auto Encoder objective (ELBO)
	wasserstein	Wasserstein Auto Encoder objective (MMD)
	spherical	Spherical Auto Encoder objective
Discriminator	spectral_discriminator	Use the MultiScale discriminator from EnCodec.
Others	causal	Use causal convolutions
	noise	Enable noise synthesizer V2

Outline

- Non-neural music synthesizers
- DL-based timbre synthesis (e.g., generation of single notes)
- **DL-based generation of loops**
- HW2

Progress in Image Generation

- “**x → image**”
 - PGGAN (ICLR’18)
 - StyleGAN (CVPR’19)
 - StyleGAN 2 (CVPR’20)
 - StyleGAN 3 (NeurIPS’21)
 - VQGAN (CVPR’21)
- “**text → image**”
 - DALL·E (ICML’21)
 - DALL·E 2 (arXiv’22)
 - Imagen (NeurIPS’22)
 - Stable Diffusion (CVPR’22)
 - MidJourney
- “**class → image**”
 - BigGAN (ICLR’19)
 - ADM (diffusion; NeurIPS’21)

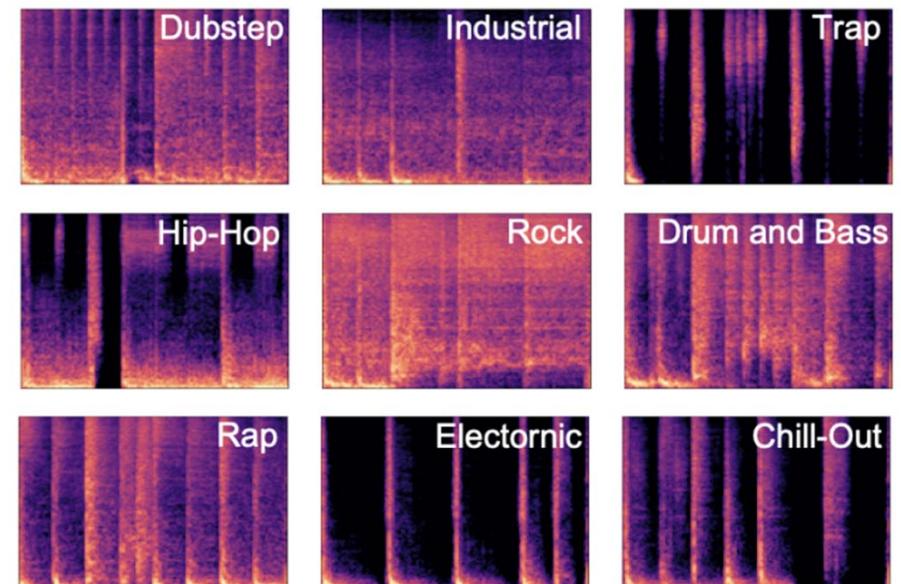
Why GAN?

- GANs are advantageous when...
 1. Hand-crafted loss functions are not good enough
 - For example, reconstruction loss functions such as MSE can easily lead to over-smoothing result
 - Let the model (i.e., the discriminator) **learn** the loss function
 - This is the case for source separation and Mel-vocoders, for we don't want to rely on reconstruction loss functions only
 2. There is no paired data; or there is no ground truth target
 - This is the case for generation problems with weaker conditions
 - Such as “ $x \rightarrow$ image”, “class \rightarrow image”, “text \rightarrow image”, and “ $x \rightarrow$ audio”
- Downside of GANs
 - Work well for **fixed-length data** (e.g., images) but not that good for **sequential data**
 - Need to build a **language model** (e.g., via **Transformers**) instead

Loop Generation by StyleGAN2

<https://loopgen.github.io/>

- “ $x \rightarrow \text{audio}$ ”
- Use StyleGAN2 (a non-autoregressive model) to generate **fixed-length** Mel-spectrograms
- Then use a Mel-vocoder (e.g., MelGAN) to get waveforms
- Here the Mel-spectrograms are not from real data, but are *generated*

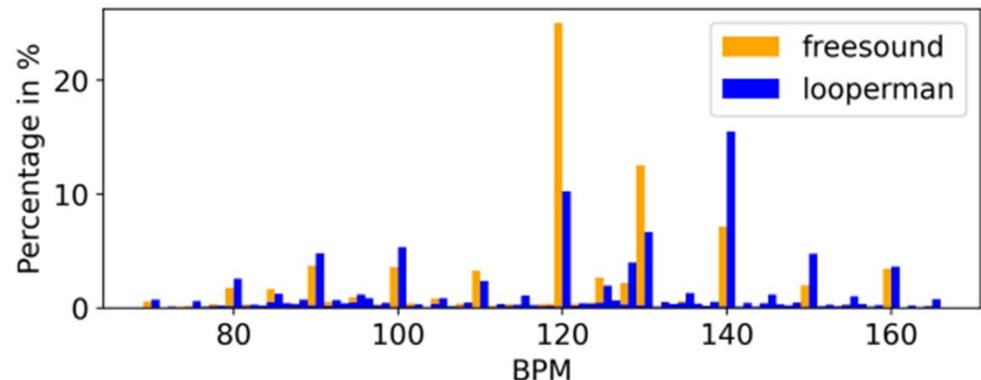


Ref: Hung et al, “A benchmarking initiative for audio-domain music generation using the FreeSound loop dataset,” ISMIR 2021

Dataset: FreeSound Loop Dataset (FSLD)

<https://zenodo.org/record/3967852>

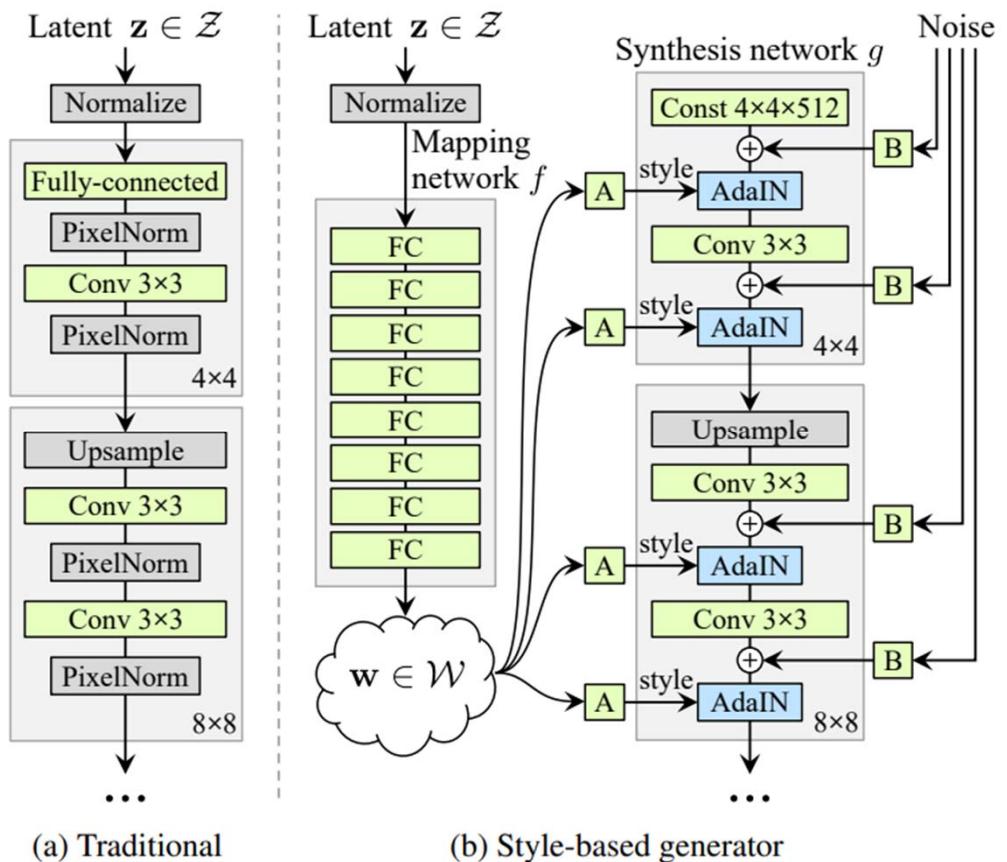
- 9,455 production-ready, public-domain CC-licensed loops
- 2,608 drum loops
- 13,666 **one-bar** loops
- All time-scaled to **120 BPM**
 - “We use pyrubberband to temporally stretch all the loops to **2-second** long [...] most sounded plausible with little perceptible artifacts”
 - “This, however, may not be the case if the loops are not drum loops. Some data filtering might be needed then, e.g., to remove those whose tempo are much away from 120 BPM.”



Ref: Hung et al, “A benchmarking initiative for audio-domain music generation using the FreeSound loop dataset,” ISMIR 2021

StyleGAN & StyleGAN2

- Traditionally the latent code is provided to the generator through an input layer
- In StyleGAN, the latent code \mathbf{z} is transformed into another latent \mathbf{w} thru fully-connected layers and then used to control **AdaIN** operations after each convolution layer



Ref 1: Karras et al, "A style-based generator architecture for generative adversarial networks," CVPR 2019

Ref 2: Karras et al, "Analyzing and improving the image quality of StyleGAN," CVPR 2020

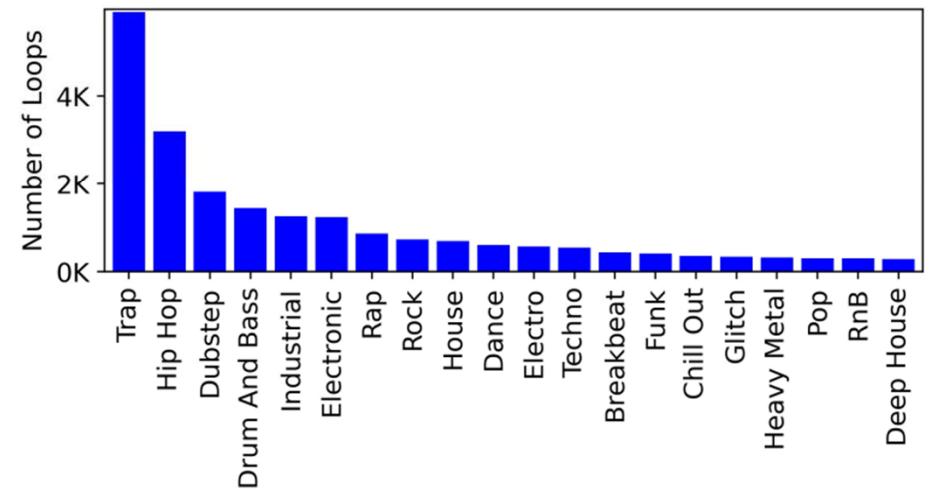
Objective Evaluation Metric: Inception Score

- A **reference-free** evaluation metric for audio generation models
- Measures the **quality + diversity** of the generated data and detects **mode collapse** by using a pre-trained *domain-specific* classifier
 - Computed as the KL divergence between the conditional probability $p(y|x)$ and marginal probability $p(y)$,

$$\text{IS} = \exp \left(E_x [\text{KL}(p(y|x) \| p(y))] \right),$$

- To compute the IS, we use short-chunk CNN to train a 66-class classifier for **loop genre** classification

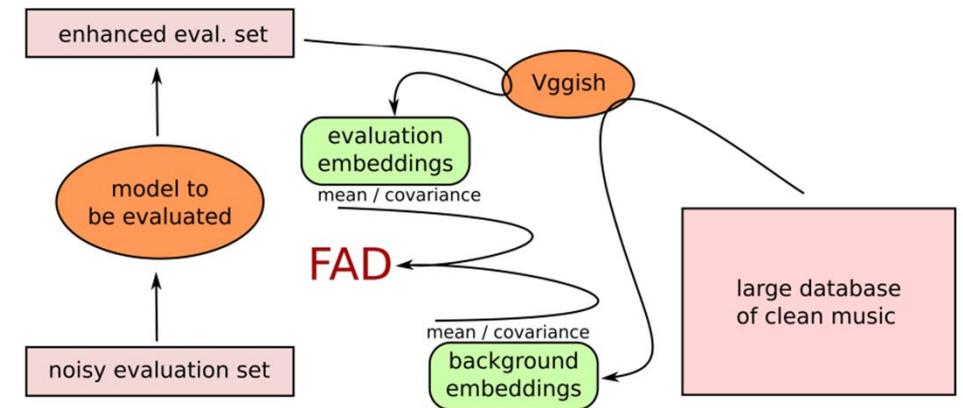
Ref: Salimans et al, “Improved techniques for training GANs,” NeurIPS 2016



Objective Evaluation Metric: Fréchet Audio Distance (FAD)

- Also *reference-free*
- The discrepancy of the ***multivariate Gaussians distributions*** of real and generated data in an embedding space computed by a pretrained VGGish-based audio classifier

$$\mathbf{F}(\mathcal{N}_b, \mathcal{N}_e) = \|\mu_b - \mu_e\|^2 + \text{tr}(\Sigma_b + \Sigma_e - 2\sqrt{\Sigma_b \Sigma_e})$$



- The VGGish is an audio classifier over 3,000 classes trained on a large dataset of YouTube videos (i.e., *not domain-specific*). The activations from the 128 dimensional layer prior to the final classification layer are used as the embedding.
- Measures **quality + diversity**

Loop Generation by StyleGAN2

<https://github.com/allenhung1025/LoopTest>

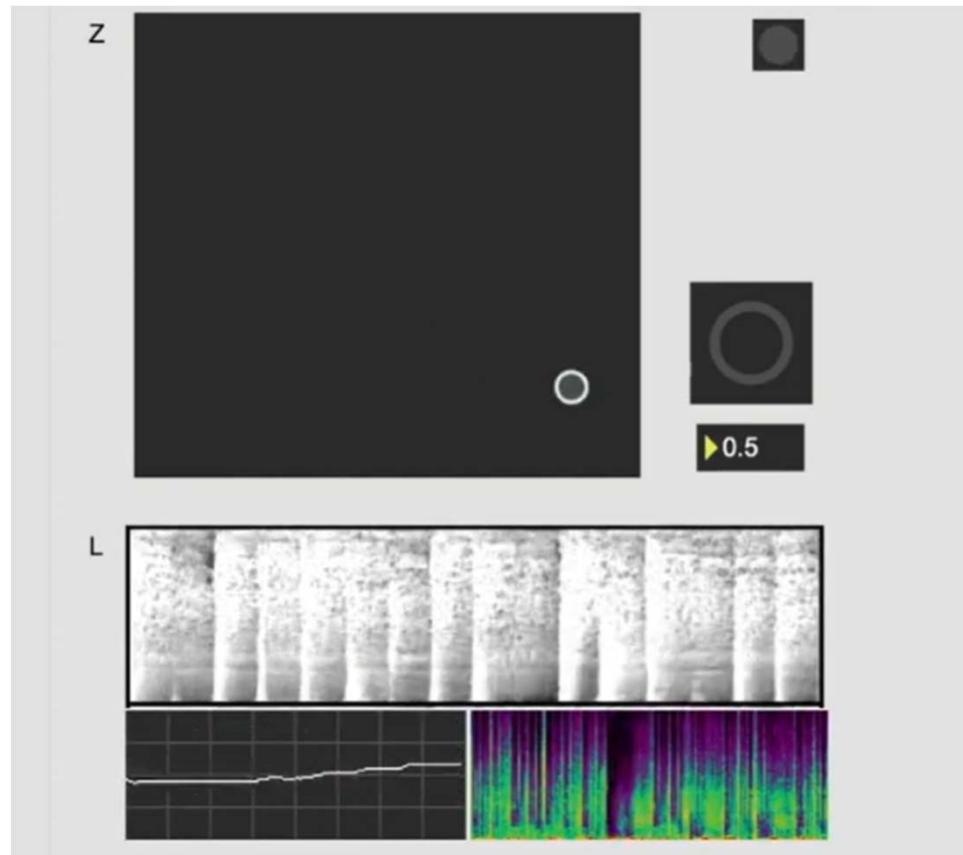
- *Easy to try (recommended!)*
 - “The total training time is 100 hr on an **NVIDIA GTX1080 GPU** with 8GB memory.”
- We also provide code for objective evaluation metrics
 - **Inception score**
 - **FAD**
 - Two other **diversity metrics**
 - Number of statistically-different bins (**NDB**)
 - Jensen-Shannon divergence (**JSD**)

	IS ↑	FAD ↓	JS ↓	NDB/K ↓
Looperman	11.9 ± 3.21	0.11	0.01	0.01
Freesound	6.30 ± 1.82	0.72	0.08	0.46
StyleGAN	1.31 ± 1.95	13.78	0.43	0.94
StyleGAN2	5.24 ± 1.84	7.91	0.09	0.59
UNAGAN	3.33 ± 1.65	4.32	0.16	0.73

Ref: Hung et al, “A benchmarking initiative for audio-domain music generation using the FreeSound loop dataset,” ISMIR 2021

Loop Generation: Exploring the Latent Space

<https://www.youtube.com/watch?app=desktop&v=Sg3tlwPjPBI>

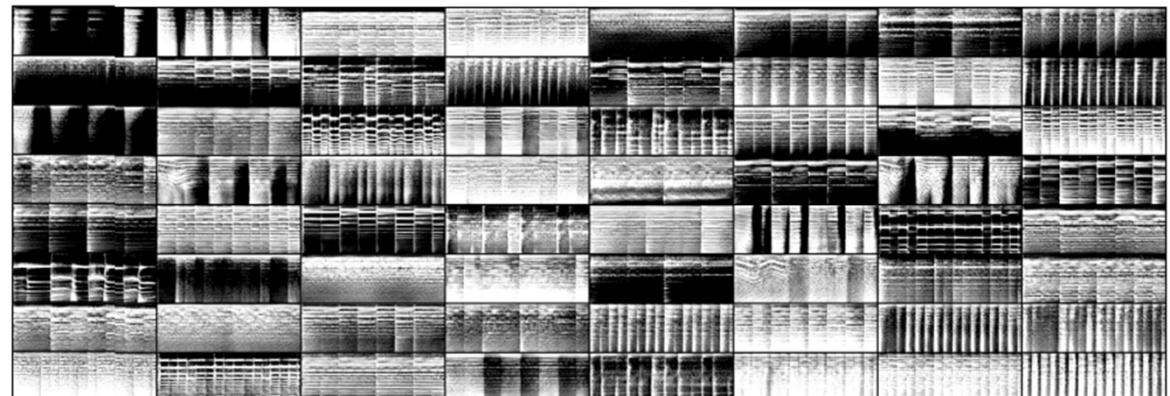


By Nao Tokui (2023)

Loop Generation by Projected GAN

<https://arthurddd.github.io/PjLoopGAN/>

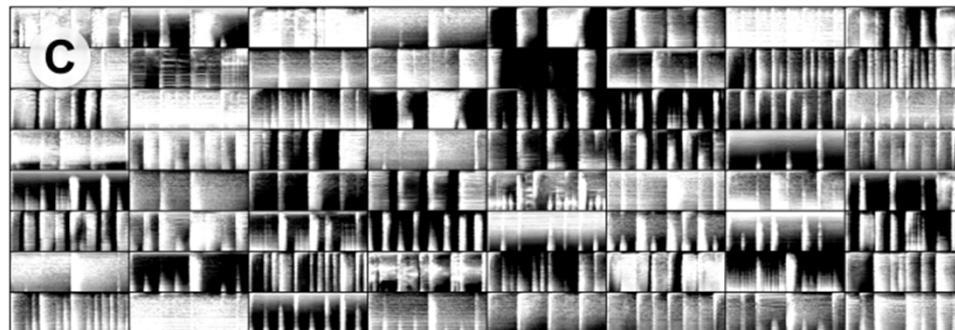
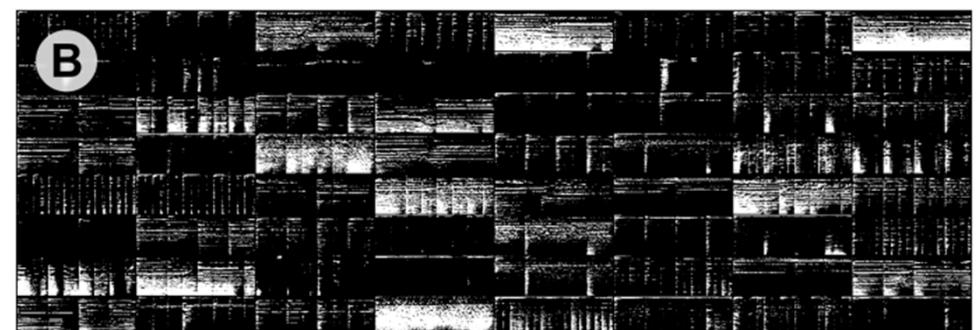
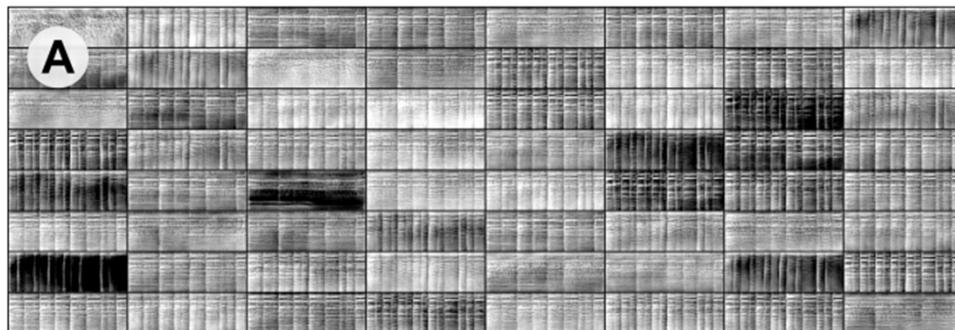
- Use **projected GAN** to speed up and stabilize GAN training
 - With Projected GAN, the loop generation models can converge around 5 times faster without performance degradation
- Consider the generation of not only drum loops but also **synth loops**



Ref: Yeh et al, “Exploiting pre-trained feature networks for generative adversarial networks in audio-domain loop generation,” ISMIR 2022

Different Cases of GAN Training

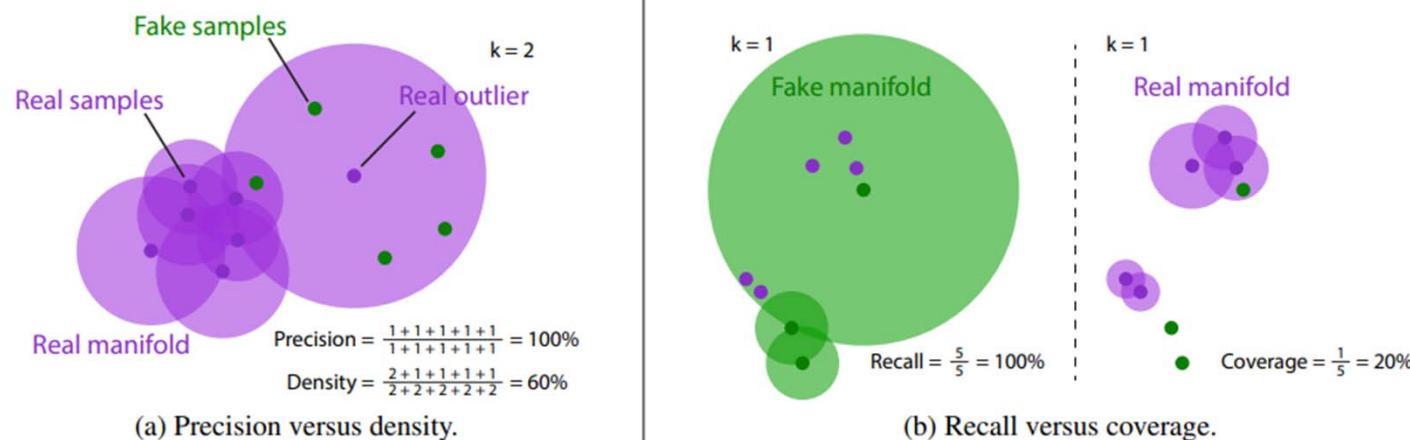
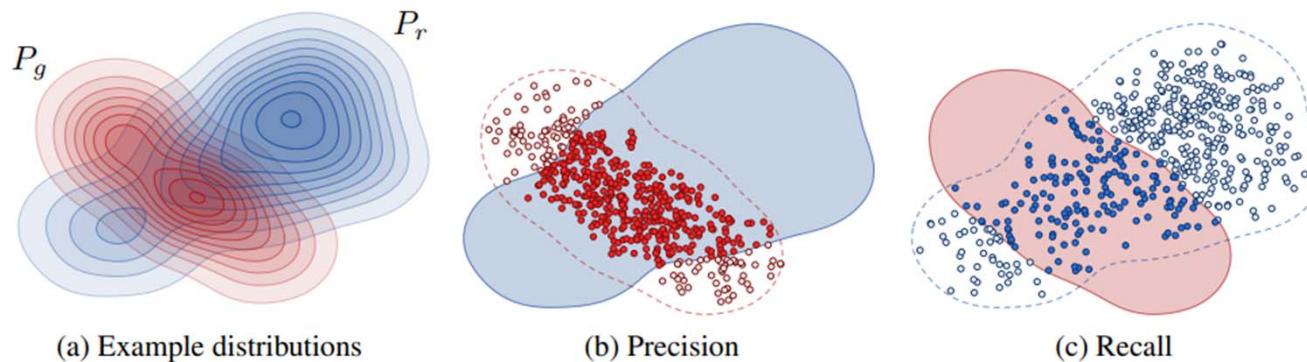
- Also studied in the Projected GAN paper for loop generation



- Examples from Mel-spectrogram generation (not Mel-vocoder)
 - (A) Mode collapse
 - (B) Gradient vanishing
 - (C) Successful training

Ref: Yeh et al, “Exploiting pre-trained feature networks for generative adversarial networks in audio-domain loop generation,” ISMIR 2022

More Evaluation Metrics for Unconditional Generation



Ref 1: Kynkänniemi et al., “Improved precision and recall metric for assessing generative models,” NeurIPS 2019
Ref 2: Naeem et al., “Reliable fidelity and diversity metrics for generative models,” ICML 2020.

Advanced Topics for Loop Generation

- We have only considered loop generation as a fixed-length, unconditional audio generation problem
 - “**x → audio of loop**”
- Loop generation can also be ***class-conditioned***
 - “**MIDI pitch → audio of single note**”
 - “**genre class (e.g., rap, trap, dubstep) → audio of loop**”
 - “**loop role (e.g., percussive, melody, bass, chord, FX) [1] → audio of loop**”
- Loop generation can also be of ***variable-length*** (various BPM, or #bars)
 - May need a **language model (LM)**
 - See forthcoming lectures

[1] Ching et al, “Instrument role classification: Auto-tagging for loop based music,” CSMC 2020

Outline

- Non-neural music synthesizers
- DL-based timbre synthesis (e.g., generation of single notes)
- DL-based generation of loops
- **HW2**