Yandex

SCHOOL OF DATA ANALYSIS

TTS: Vocoders (vol. I)

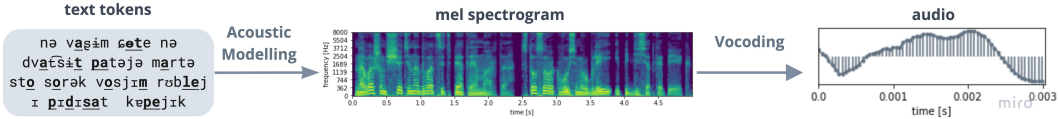
Sergey Dukanov

April 25th 2022

Lecture Plan

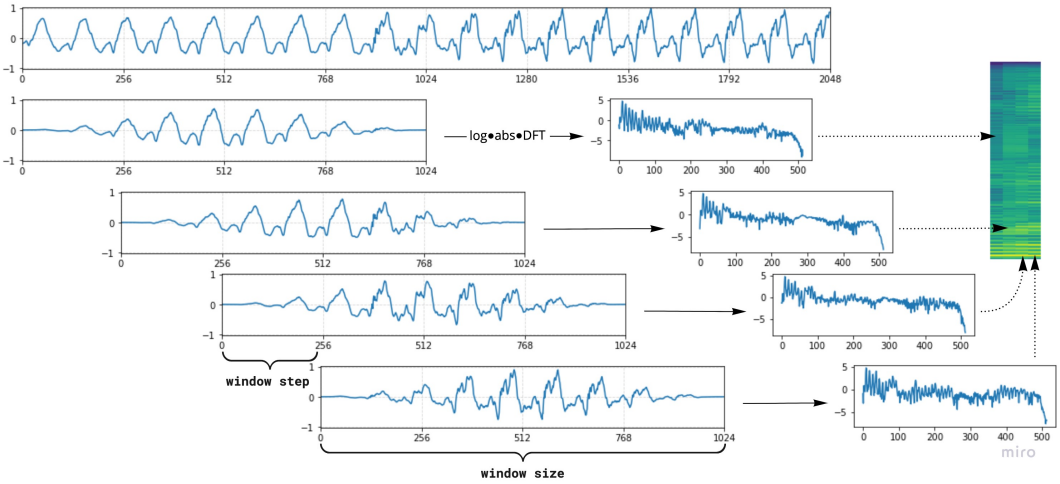
1. recap TTS and DSP
2. what is vocoder?
3. modern vocoder types
4. WaveNet, WaveRNN
5. a bit of DSP
6. LPCNet

TTS Pipeline



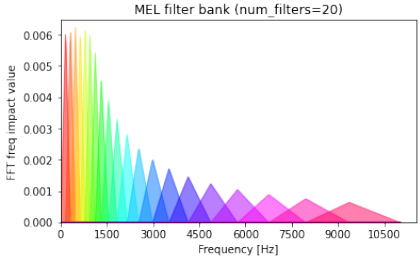
- common TTS pipeline: NLP frontend $\rightarrow$ acoustic modelling $\rightarrow$ **vocoding**
- **vocoders** converts low resolution linguistic features into speech signal
- in the beginning of Statistical Parametric Speech Synthesis $\mathbf{F_0}$ and **MFCC** features were used
- but with the advent of **end-to-end** approaches, the use of **mel** features has become common

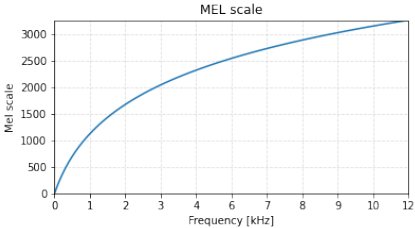
Short-Time Fourier Transform

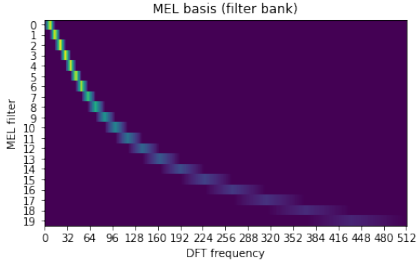


Mel-scale and mel filter bank

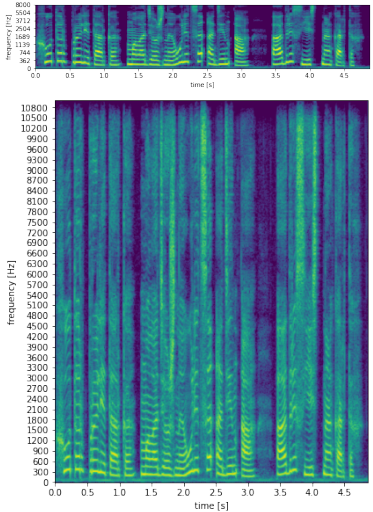
- dynamic frequency squeezing
- empirically estimated
- filter bank – basis in frequency domain
- mel-scale vs bark-scale

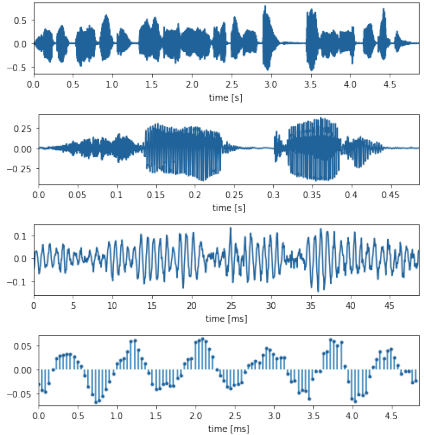






Audio Representations





Question: what "vocoder" we already know?

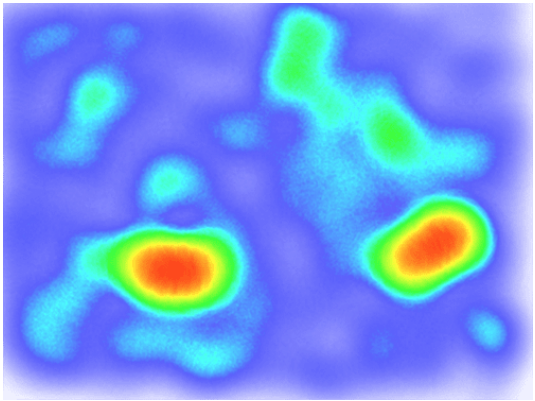
Probability

Let's consider probability **distribution** p on $\mathbb{R}^N$: the more probability $p(x)$ – the more likely that $x \in \mathbb{R}^N$ is **speech** signal.

Speech **synthesis** $\Leftrightarrow$ **sampling** from p.

We can condition on **linguistic features** y (e.g. mel spectrogram): $p(x|y)$ to add more control.

We can **parametrize** our distribution: $p(x|\theta)$ and use the power of DL.



Categorize Vocoders

We can group vocoder architectures by the way of $p(x|\theta)$ is represented and optimized:

- autoregressive models (*WaveNet*, *WaveRNN*, *LPCNet*)
- inverse autoregressive flows (*Parallel WaveNet*, *ClariNet Vocoder*)

- non autoregressive normalizing flows (*WaveGlow*, *FloWaveNet*)
- generative adversarial networks (*MelGAN*, *HiFiGan*)

- log-density gradient estimation (*WaveGrad*)

Autoregressive

Consider **signal** $x_{[1:k]} = (x_1, \ldots, x_k)$ and its **probability** $p(x_{[1:k]})$.

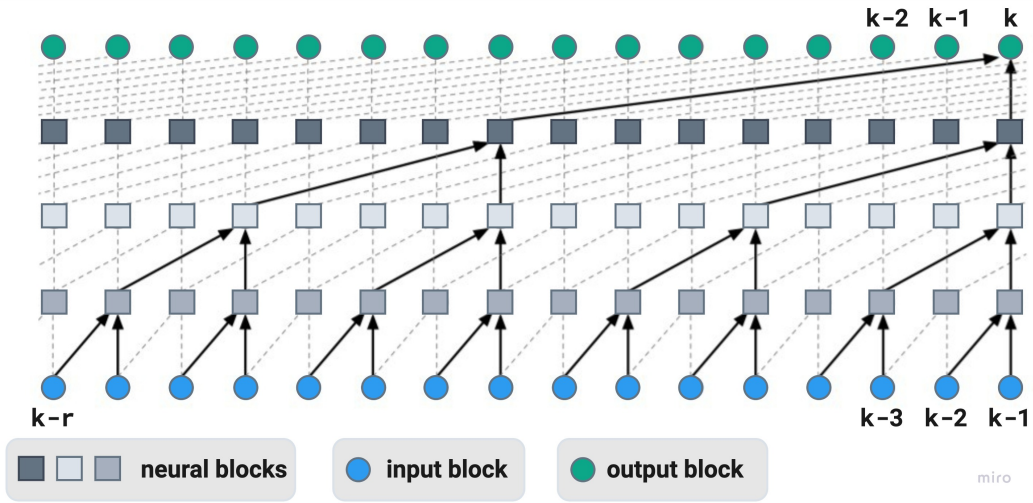
Following the **conditional probability** decomposition rule (y and θ omitted for clarity):

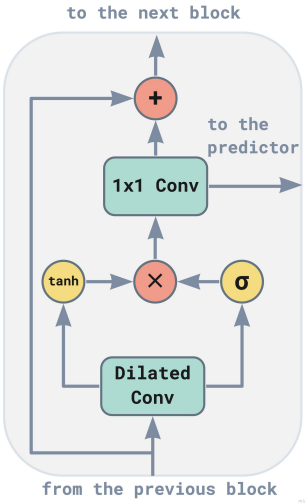
$$p(x_{[1:k]}) = p(x_k | x_{[1:k-1]}) \ldots p(x_1 | x_0) p(x_0)$$

We can assume that $p(x_k | x_{[1:k-1]}) \sim p(x_k | x_{[\mathbf{k-r}:k-1]})$, where r is the approximate number of steps that x_k really depends on.

Question: what r would you choose?

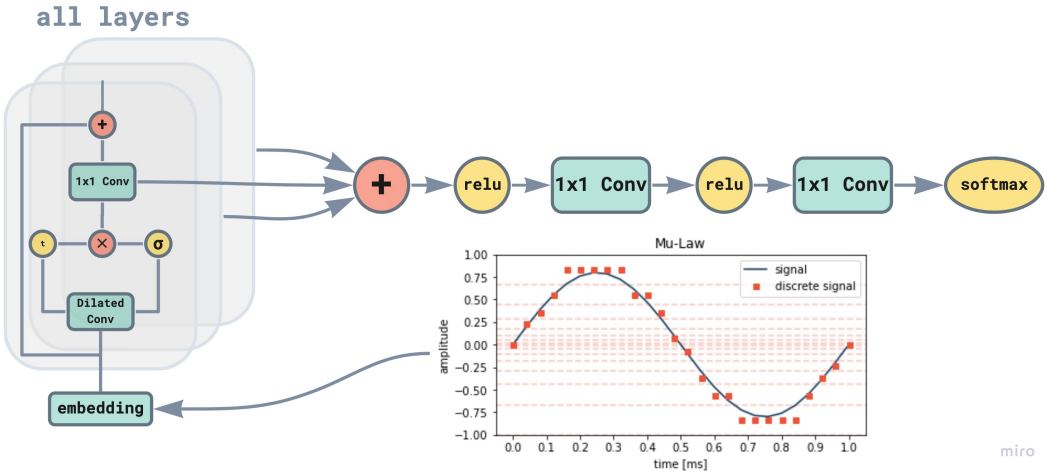
Let's model $p(x_k | x_{[k-r:k-1]}; \theta)$ via neural net!

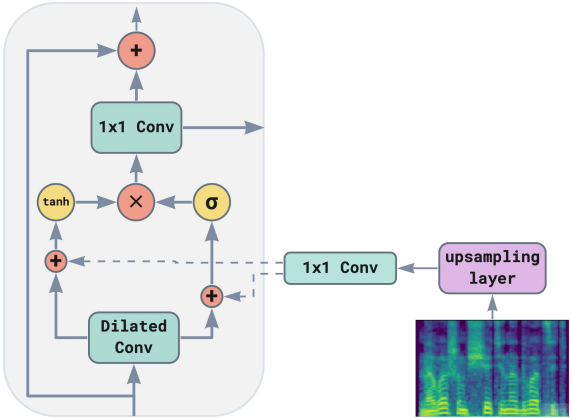

k-2
k-1
k
k-r
k-3
k-2
k-1
neural blocks
input block
output block
miro



WaveNet dilated convolutions:

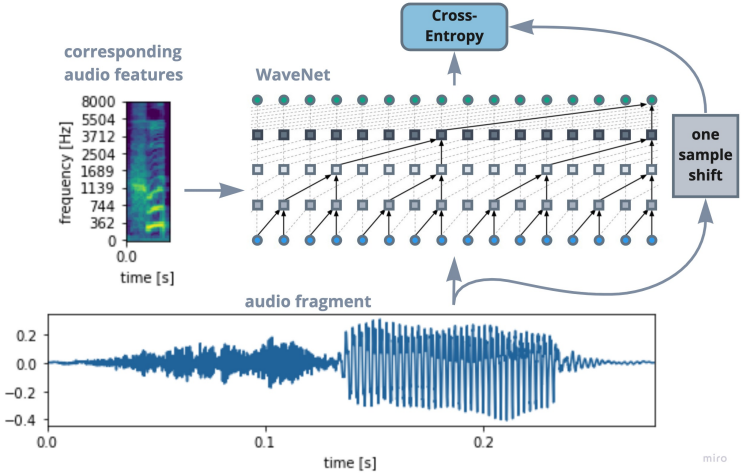
- causal (does not look into the future)
- dilation increases exponentially: $1, 2, 4, 8, \ldots$
- kernel size $= 2$ (but 3 is also possible)


all layers
+
1x1 Conv
t
σ
Dilated Conv
embedding
relu
1x1 Conv
relu
1x1 Conv
softmax
Mu-Law
signal
discrete signal
amplitude
time [ms]
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
miro

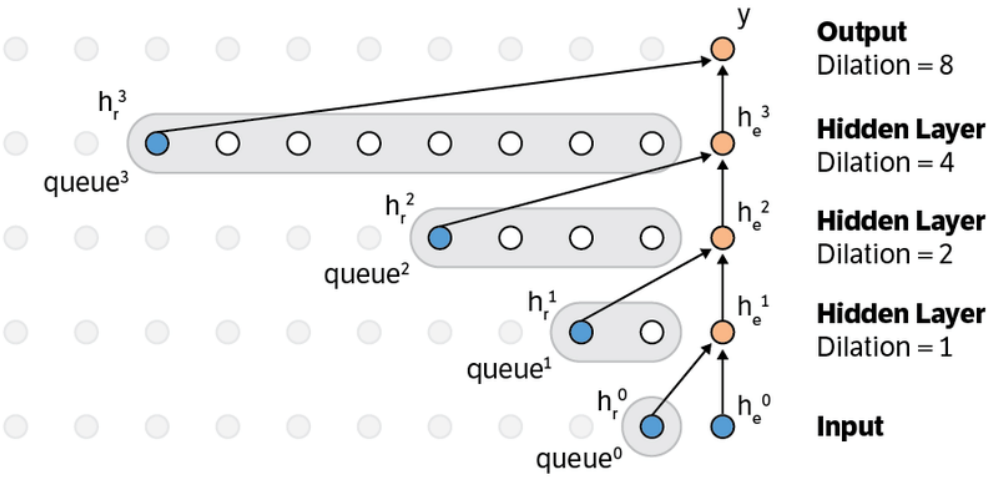


- signal and spectrogram have difference time resolution
- **upsampling layer**: transposed convolutions or just nearest neighbour upsampling

Question: what upsampling factor must be?



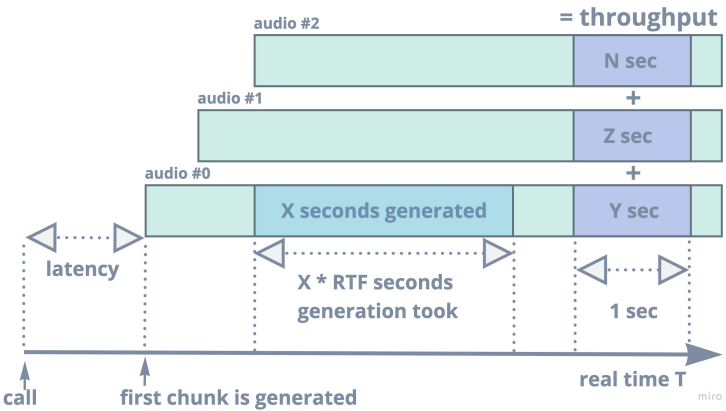
- training on pairs $(\text{signal}, \text{features})$ (**no text** is needed)
- **cross-entropy** minimization
- WaveNet inputs **original** samples
- requires precisely aligned conditioning


y
h_r^3
h_e^3
queue^3
h_r^2
h_e^2
queue^2
h_r^1
h_e^1
queue^1
h_r^0
h_e^0
queue^0
Output
Dilation = 8
Hidden Layer
Dilation = 4
Hidden Layer
Dilation = 2
Hidden Layer
Dilation = 1
Input

- almost **no prior knowledge** about audio signals
  (except the choice of receptive field and signal quantization)
- can be viewed as a **non-linear causal filter**
- can represent complicated signals,
  hand-crafted engineering is not required

- sampling from output distribution for inference
  (greedy decoding leads to worsen results, beam search is impossible)

Performance Metrics



- latency $\to 0$
- real-time factor (RTF) ≤ 1
- throughput (SPS) $\to$ max

WaveNet architecture details:

- $\text{num\_blocks} = N = 40$
- $\text{kernel\_size} = K = 2$
- $\text{hidden\_size} = R = 64$
- $\text{skip\_size} = S = 128$
- $\text{output\_size} = A = 256$
- $\text{sample\_rate} = T = 22050$

The needed number of floating point operations per second:

$$((2 \cdot R \cdot 2R + R \cdot (S + R)) \cdot N + S \cdot A + A \cdot A) \cdot T \cdot 2$$

$$\sim 0.055 \text{ TFLOPs}$$

Server CPU core has ~ 0.077 TFLOPs !

But! We ignore memory bandwidth.
(L2 $\rightarrow$ L1 bandwidth ~ 100 GB/s)

GPU has much higher peak ~ 5.3TFLOPs

But it requires specific programming skills!

CPU

- matrix multiplications using AVX instructions (requires very accurate at caching)
- replace non-liniarities with approximations
- int16 or float16 arithmetic

Numpy implementation works with $\mathrm{RTF} \sim 10$ (slow).

GPU

- GPU is not suitable for autoregressive models
- only **persistent kernels** provide $\mathrm{RTF} < 1$
- RTF-throughput trade-off

CUDA implementation works with $RTF \sim 0.95$ and $\mathrm{batch_size} = 1$ on 1080ti (ineffective). ▸ nVidia blog post

WaveNet Pros and Cons

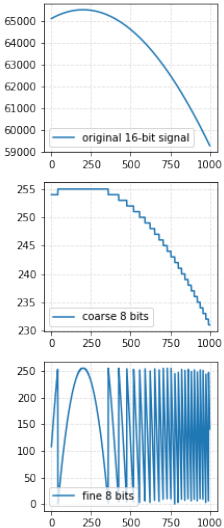
Pros

- the ease of implementation
- fast and stable convergence
- generated audio is almost indistinguishable from the original one
  (if using its spectrogram)

Cons

- inference is hard to parallelize
- sometimes domain shift occurs (need to train or generated spectrograms)

  Question: what generated spectrograms to use?



WaveNet models 8-bit quantized signal
(continuous distribution is possible, but harder)
$\Rightarrow$ let's split 16-bit signal into coarse and fine parts (8-bit both)

Sequential operations in WaveNet hard to parallelize
$\Rightarrow$ let's minimize number of matmuls and use the power of RNNs

$$
\begin{aligned}
\mathbf{x}_k &= [c_{k-1}, f_{k-1}, c_k] && (1)\\
\mathbf{u}_k &= \sigma(\mathcal{R}_u \mathbf{h}_{k-1} + \mathcal{I}_u^* \mathbf{x}_k) && (2)\\
\mathbf{r}_k &= \sigma(\mathcal{R}_r \mathbf{h}_{k-1} + \mathcal{I}_r^* \mathbf{x}_k) && (3)\\
\mathbf{e}_k &= \tau(\mathbf{r}_k \circ (\mathcal{R}_e \mathbf{h}_{k-1}) + \mathcal{I}_e^* \mathbf{x}_k) && (4)\\
\mathbf{h}_k &= \mathbf{u}_k \circ \mathbf{h}_{k-1} + (1 - \mathbf{u}_k) \circ \mathbf{e}_k && (5)\\
\mathbf{h}_k^c, \mathbf{h}_k^f &= \mathrm{split}(\mathbf{h}_k) && (6)\\
p(c_k) &= \mathrm{softmax}(\mathcal{O}_2 \mathrm{relu}(\mathcal{O}_1 \mathbf{h}_k^c)) && (7)\\
p(f_k) &= \mathrm{softmax}(\mathcal{O}_4 \mathrm{relu}(\mathcal{O}_3 \mathbf{h}_k^f)) && (8)
\end{aligned}
$$

WaveRNN:

- GRU-like architecture with two linear layers on top
- $\mathcal{I}^*$ denotes masked matrix

  Question: what is the propose of the masking?
- only 6 matrix-vector multiplications per sample (but larger matrices)
- **0.22 RTF** on GPU (P100)
  - custom CUDA kernel
  - weights are stored on registers

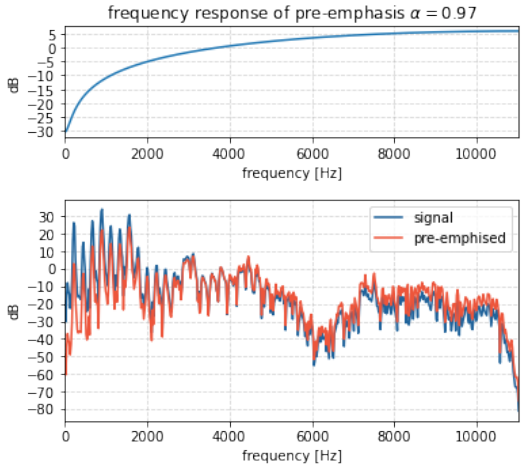
During synthesis high-frequency noise occurs due to models' imperfection.

To increase **signal-to-noise** ratio we can add more energy to higher frequencies.

Pre-emphasis filter emphasises higher frequencies:

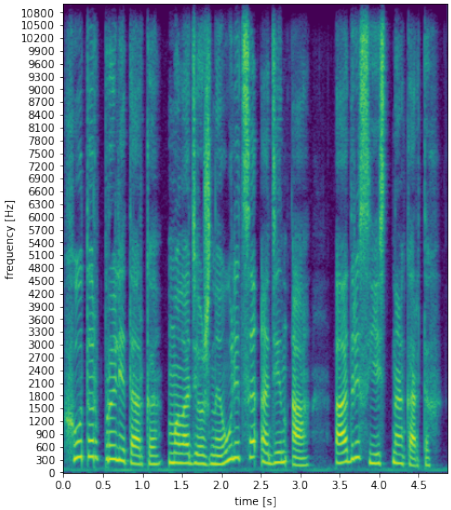
$$\hat{x}_k = x_k - \alpha x_{k-1}, \text{ where } \alpha \sim 0.97$$

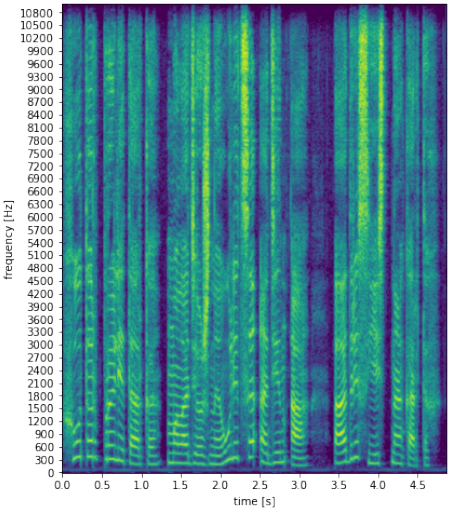
May help not only vocoders but acoustic models also.



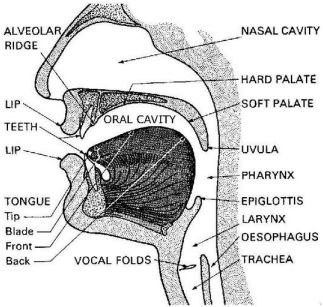
Pre-emphasis

Spectrograms

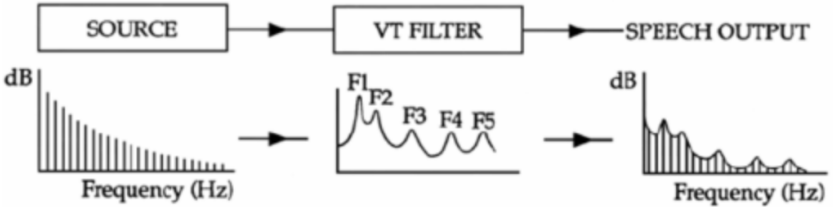




Human Vocal Tract and Source-Filter Model



- the **vocal folds** – source for voiced speech (vowels and some consonants)
- unvoiced sounds (e.g. fricatives) are produced via random noise via turbulence near the **lips and teeth**
- the **oral and nasal cavities** produce resonances – playing the role of a filter
- **Question**: how to mathematically represent this filter?



Linear Prediction Coding

Predicting signal sample $s[n]$ as linear combination of previous ones:

$$s[n] = \sum_{k=1}^{p} a_k s[n-k] + e[n]$$

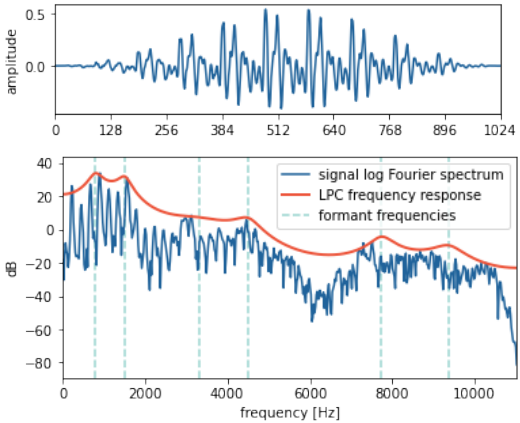
- $a_1, a_2, \ldots, a_p$ – coefficients of linear predictor
- $e[n]$ – prediction error (additional information)

To find LPC coefficients a_k we minimize energy of residual $e[n]$ for $n = 1, \ldots, N$:

$$\sum_n e^2[n] = \sum_n (s[n] - \sum_k^p a_k s[n-k])^2$$

Question: how to find solution?

Formants Detection



$$\frac{S(z)}{E(z)} = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}} = \frac{1}{A(z)}$$

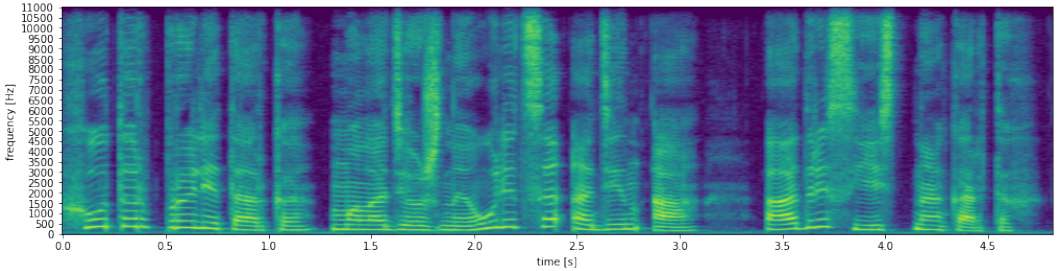
Frequency response of $\frac{1}{A(z)}$ filter is **spectral envelope**. $\Rightarrow$

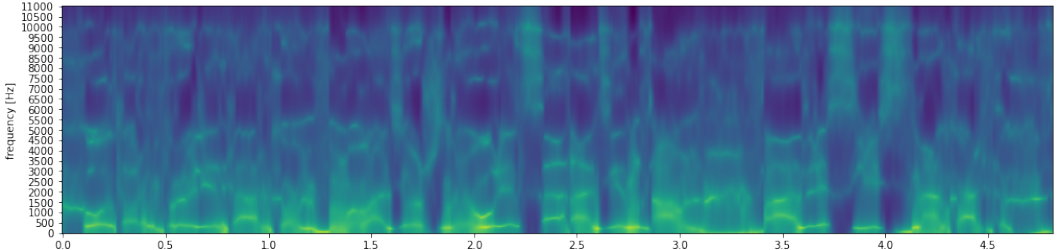
LPC analysis separates the **resonant characteristics** of a speech sound from the **source characteristics**.

Formants occur around frequencies that correspond vocal tract resonances.

Poles of $\frac{1}{A(z)}$ correspond to **formants**.

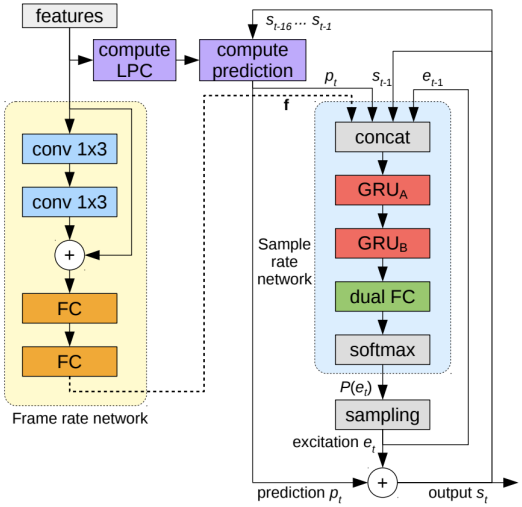
LPC Spectrogram





- WaveRNN has **no prior knowledge** about speech domain
- model capacity is partially spent on modelling **simple periodic patterns**

- LPC can give **approximation** $\hat{s}[n] = \sum_{k=1}^{p} a_k s[n-k]$ on each step (in autoregressive manner)
- so we need to predict only $e[n]$ **residual**
- $\Leftrightarrow$ taking **spectral envelope** modeling away from vocoder

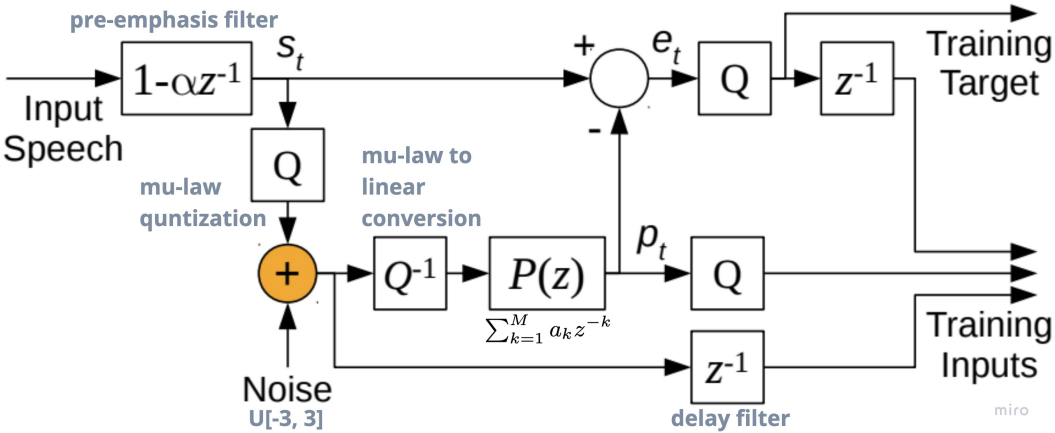


Features for signal synthesis:

- 18 Bark-scale cepstral coefficients
- 2 pitch parameters (period, correlation)

To obtain LPC coefficients from features:

- Bark-scale cepstral $\xrightarrow{\text{projection}}$
- Power Spectral Density $\xrightarrow{\text{inverse FFT}}$
- Autocorrelation function $\xrightarrow[\text{equations}]{\text{solve liniear}}$


pre-emphasis filter
$1-\alpha z^{-1}$
s_t
Input Speech
Q
mu-law quntization
mu-law to linear conversion
+
Q^{-1}
$P(z)$
$\sum_{k=1}^{M} a_k z^{-k}$
Noise
U[-3, 3]
+
-
e_t
Q
z^{-1}
Training Target
p_t
Q
z^{-1}
delay filter
Training Inputs
miro

Lecture summary

- vocoder is an essential part of modern TTS pipeline
- usually vocoder is trained separately from acoustic model
- there is a trade-off between **sampling efficiency** and **ease of training**
- examples: Wavenet, WaveRNN, LPCNet

Yandex

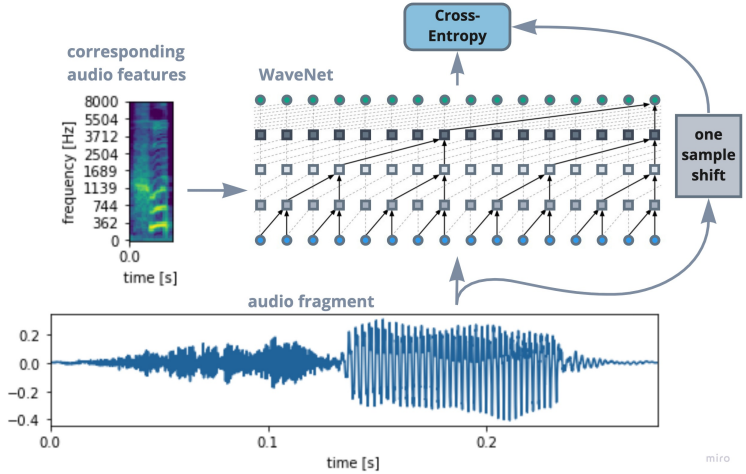
SCHOOL OF DATA ANALYSIS

TTS: Vocoders (vol. II)

Sergey Dukanov

May 2nd 2022

Recap WaveNet



In **original WaveNet** we sample from **categorical** distribution ($2^8 = 256$ classes).

But it **doesn't relate values** (e.g. $128 > 127$) and also restricts the range.

Parallel training, autoregressive inference.

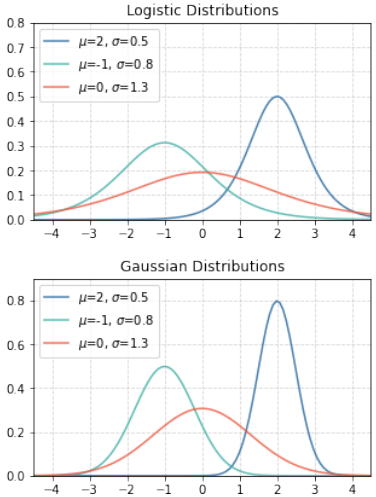
WaveNet Continuous Output

We can try continuous distribution:

- **MoL** (Mixture of Logistic Distributions)
- **MoG** (Mixture of Gaussian Distributions)

Parametrize them with $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^d$.

We will train network which predicts $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma}$.
Question: what loss to use?

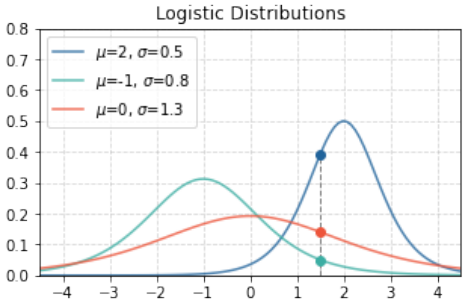


Likelihood of MoL

Consider audio sample $\hat{\boldsymbol{x}}$ and MoL **PDF** with parameters $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^d$.

$$p(\hat{\boldsymbol{x}}; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \sum_{k=1}^{d} \boldsymbol{\pi}_k \frac{e^{-(\hat{\boldsymbol{x}}-\boldsymbol{\mu}_k)/\boldsymbol{\sigma}_k}}{(1 + e^{-(\hat{\boldsymbol{x}}-\boldsymbol{\mu}_k)/(2\boldsymbol{\sigma}_k)})^2 \boldsymbol{\sigma}_k}$$

Mximizing $\log p(\hat{\boldsymbol{x}}; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma})$ (log-likelihood) is **prone to numerical issues**.

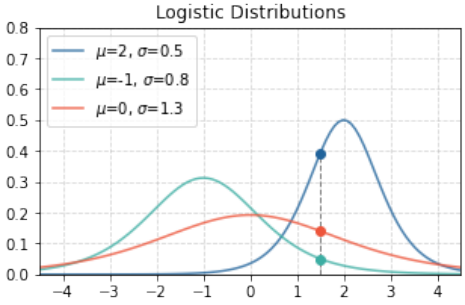


CDF instead PDF

We can optimize quantized surrogate loss using **CDF** instead of **PDF**:

$$P(\hat{\mathbf{x}}; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma}) =$$

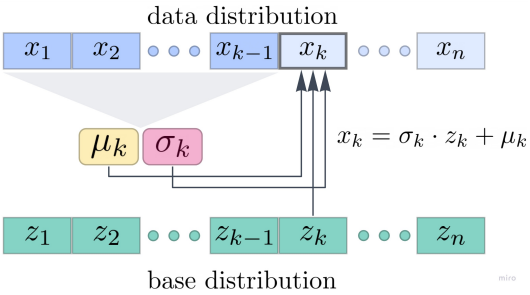
$$\sum_{k=1}^{d} \boldsymbol{\pi}_k [\mathrm{sigm}(\frac{\hat{\mathbf{x}} + \delta - \boldsymbol{\mu}_k}{\boldsymbol{\sigma}_k}) - \mathrm{sigm}(\frac{\hat{\mathbf{x}} - \delta - \boldsymbol{\mu}_k}{\boldsymbol{\sigma}_k})]$$



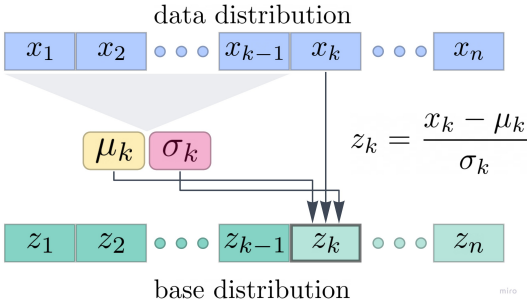
Maximizing **Single Gaussian** log-like via **PDF** does not seem to have same issues.

WaveNet as Masked Autoregressive Flow

Adding corresponding **noise** we can sample from autoregressive model, which predicts μ_i and $\log \sigma_i$ on i-th step: $x_i = u_i \cdot \sigma_i + \mu_i$, where $i \sim \mathcal{N}(0,1)$ or $\mathcal{L}(0,1)$



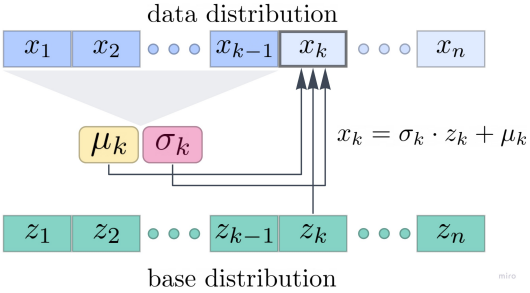
Masked Autoregressive Flow
(**sampling**)



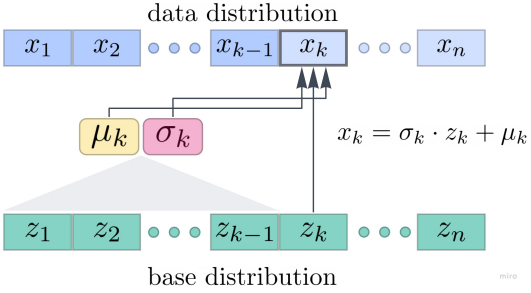
Masked Autoregressive Flow
(**likelihood estimation**)

MAF and IAF

MAF restricts us to auto-regressive inference (by design). But **IAF** does not!



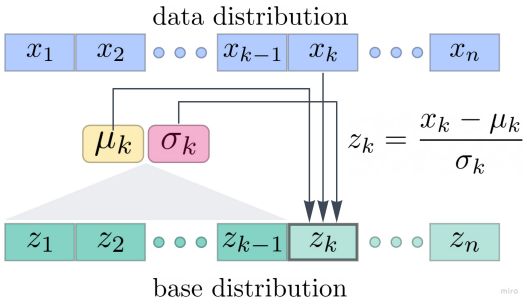
Masked Autoregressive Flow (**sampling**)



Inverse Autoregressive Flow (**sampling**)

Question: what is the problem with **IAF**?

Inverse Autoregressive Flow Training



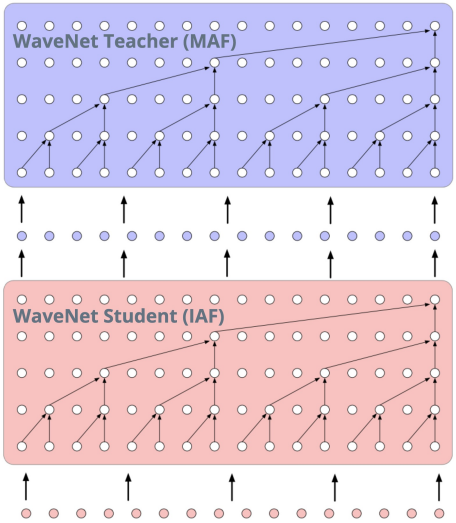
Inverse Autoregressive Flow
(**likelihood estimation**)

IAF training involves **autoregressive** $\Rightarrow$

Learning an IAF directly through maximum likelihood is **impractical**.

What about training IAF with **probability density distillation**?

MAF Distills IAF



Teacher Output
$P(x_i|x_{<i})$

Generated Samples
$x_i = g(z_i|z_{<i})$

Student Output
$P(x_i|z_{<i})$

Input noise
z_i

We minimize Kullback-Leibler divergence:

$$D_{KL}(P_S||P_T) = H(P_S, P_T) - H(P_S)$$

Additional losses:

- Power loss (STFT MSE)
- Perceptual loss (. . .)
- Contrastive loss (. . .)

This original paper results are hard to reproduce.

- Perceptual and contrastive losses
- Monte-Carlo sampling

ClariNet is **text-to-wave** architecture training truly end-to-end.

ClariNet Vocoder is simplified version of Parallel WaveNet having same quality (relying on authors' experiments).

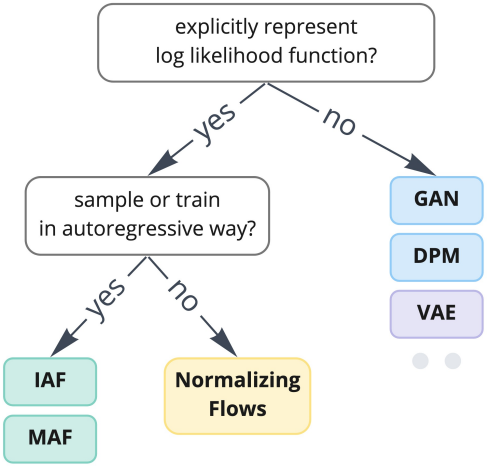
Both student (IAF) and teacher (MAF) use **single** Gaussian distribution, so Kullback-Leibler divergence can be derived analytically:

$$D_{KL}(q||p) = \log \frac{\sigma_p}{\sigma_q} + \frac{\sigma_q^2 - \sigma_p^2 + (\mu_p - \mu_q)^2}{2\sigma_p^2}$$

Auxiliary losses:

- D_{KL} regularization: $\lambda |\log \sigma_p - \log \sigma_q|^2$ for stable start
- STFT loss: $\frac{1}{B}|\psi(x) - \psi(x)|_2^2$, where $\psi(x) = |\mathrm{STFT}(x)|$ for faster convergence

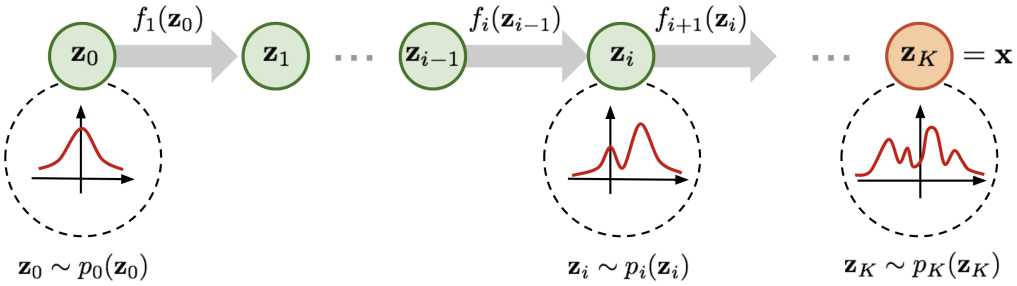
Generative Models



- we want to estimate data distribution to be able to sample from it
- considered autoregressive models:
  - **MAF** easy to estimate likelihood, hard to sample
  - **MAF** easy to sample, hard to estimate likelihood (but can distill)
- further we will figure out how to build models which:
  - allow to provide MLE directly
  - do not use autoregression

Normalizing Flow

Consider transformations f_i, for $i = 1, \dots, K$ applied sequentially to random variable $z_0 \sim \mathcal{N}(0, 1)$ – the result of this transformation is supposed to be sample from data distribution x.



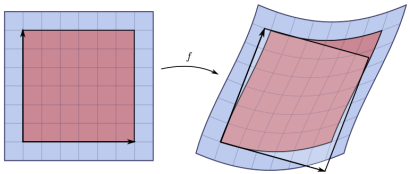
Question: what properties should $f_i(z_i)$ have for possibility of MLE?

Theorem

Let $\mathbf{z} \in \mathbb{R}^d$ *be a random variable and* $f : \mathbb{R}^d \to \mathbb{R}^d$ *– an* ***invertible*** *smooth mapping. The resulting random variable* $x = f(z)$ *has the following probability distribution:*

$$p(x) = p(z) \cdot |\det \frac{\partial f^{-1}}{\partial x}|$$

$$\mathbb{J}(f(x)) = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_d}{\partial x_1} & \cdots & \frac{\partial f_d}{\partial x_d} \end{bmatrix}$$



Jacobian – best linear approximation of the distorted parallelogram near the point.
Jacobian determinant is relation of approximated volume to the original one.

Maximum Likelihood Estimation

Consider $z^{(K)} = f_K \circ \cdots \circ f_1(z^{(0)})$, $z^{(0)} \sim p(z^{(0)})$ and f_k is invertible, than

$$p(z^{(K)}) = p(z^{(0)}) \cdot \prod_{k=1}^{K} |\det \mathbb{J}(f_k^{-1}(z^{(k)}))|$$

Log-likelihood takes the form:

$$\log p(z^{(K)}) = \log p(z^{(0)}) + \sum_{k=1}^{K} \log |\det \mathbb{J}(f_k^{-1}(z^{(k)}))|$$

Further we will take a look on some transformations.

Affine Layer

- Consider $f^{-1}(x) = Wx$ transformation of $x \in \mathbb{R}^d$.

  The reverse function is $f(z) = W^{-1}z$.

$$\det \mathbb{J}(f^{-1}(x)) = \det W$$

- Consider $f^{-1}(x) = x \cdot \sigma + \mu$ where $\mu, \sigma \in \mathbb{R}^d$.

  The reverse function is $f(z) = \frac{z-\mu}{\sigma}$

$$\det \mathbb{J}(f^{-1}(x)) = \det \operatorname{diag}(\sigma) = \prod_{i=1}^{d} \sigma_i$$

Affine Coupling Layer

- Let $\mathbf{x} = (\mathbf{x}_{\leq d}, \mathbf{x}_{>d}) \in \mathbb{R}^d$, $\mu : \mathbb{R}^{\frac{d}{2}} \to \mathbb{R}^{\frac{d}{2}}$ and $\sigma : \mathbb{R}^{\frac{d}{2}} \to \mathbb{R}^{\frac{d}{2}}$
- Affine coupling layer:

$$f^{-1}(\mathbf{x}) = (\mathbf{x}_{\leq d}, \sigma(\mathbf{x}_{\leq d}) \cdot \mathbf{x}_{>d} + \mu(\mathbf{x}_{\leq d}))$$

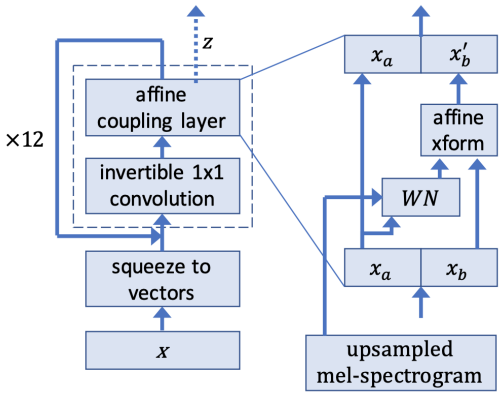
- The inverse transformation is:

$$f(\mathbf{z}) = (\mathbf{z}_{\leq d}, \frac{\mathbf{z}_{>d} - \mu(\mathbf{z}_{\leq d})}{\sigma(\mathbf{z}_{\leq d})})$$

- Jacobian determinant of $f^{-1}(\mathbf{x})$:

$$\det \mathbb{J}(f^{-1}(x)) = \begin{bmatrix} I & 0 \\ \Omega & \operatorname{diag}(\sigma(\mathbf{z}_{\leq d})) \end{bmatrix} = \prod \sigma(\mathbf{z}_{\leq d})$$

WaveGlow



$$x = f_1 \circ f_2 \circ \cdots \circ f_K(z),\ z \sim \mathcal{N}(0, I)$$

$$z = f_K^{-1} \circ \cdots \circ f_2^{-1} \circ f_1^{-1}(x),\ x \sim \text{data}$$

Audio signal x is "squeezed" before passing through network: group of 8 audio samples as vectors ($T^{\text{squeeze}} = \frac{T^{\text{audio}}}{8}$).

Two types of f:

1. matrix multiplication (1x1 convolution): $f_{\text{conv}}(x)$
2. affine coupling layer: $f_{\text{coupling}}(x)$

WaveGlow Loss

$$\log p_\theta(x) = \log p(z) + \sum_{k=1}^{K} \log |\det \mathbb{J}(f_k^{-1}(z_k))|$$

$$\log |\det \mathbb{J}(f_{\text{coupling}}^{-1}(x))| = \log |\sigma|$$

$$\log |\det \mathbb{J}(f_{\text{conv}}^{-1}(x))| = \log |\det W|$$

Final log-likelihood becomes:

$$\log p_\theta(x) = -\frac{z^T z}{2} + \sum_{k=1}^{\#\text{coupling}} \log \sigma_k(x, \text{mel}) + \sum_{k=1}^{\#\text{conv}} \log |\det W_k|$$

WaveGlow Details

- 12 blocks of coupling layer and 1x1 invertible convolution
- outputs 2 of the channels to the loss function after every 4 coupling layers
- WaveNets in f_{coupling}: $L = 8,\ R = 512,\ S = 256,\ K = 3$, not causal
  Question: how much z samples do we need to generate one sample of audio?

WaveGlow Pros and Cons

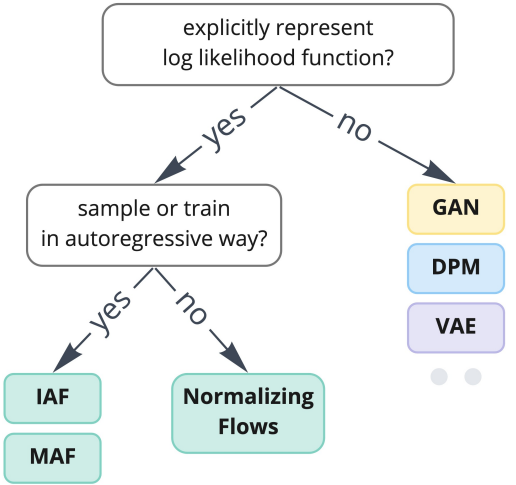
Pros

- tractable likelihood
- good convergence
- efficient inference
  unoptimized version:
  0.04 RTF on V100

Cons

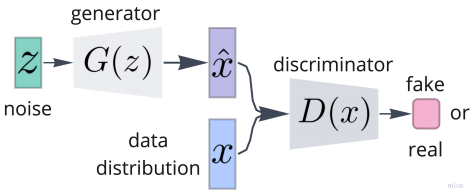
- requires more data than WaveNet
- hard to achieve WaveNet's quality
- additional effort is required to stabilize training gradients
- requires many GPUs for significantly large batch size

Generative Models



- we studied two types of models which optimize explicitly represented log likelihood;
- let's take a look on another class: Generative Adversarial Networks

Generative Adversarial Networks



- the **generator** G learns to generate plausible data
- the **discriminator** D learns to distinguish the generator's fake data from real data

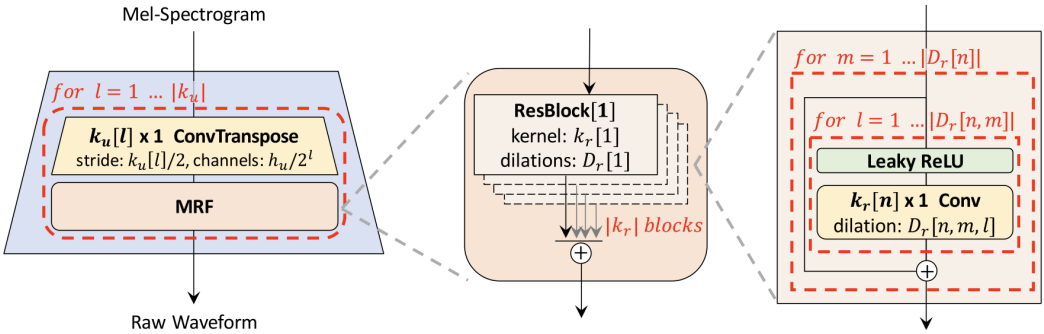
Min-max training objective:

$$\min_G \max_D E_x[\log D(x)] + E_z[\log(1 - D(G(z)))]$$

Difficulties:

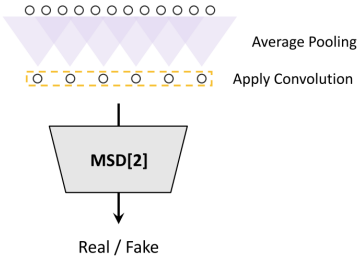
- vanishing gradients with optimal discriminator:
  - minimax loss midifications
  - Wasserstein loss
- mode collapse
- failure to converge

HiFi-GAN

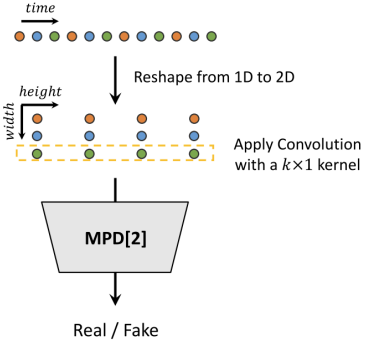


- **HiFi-GAN** – Fully-convolutional network converts mel-spectrogram to raw audio via **transposed convolutions**
- **Milti-Receptive field Fusion** observes patterns of various lengths in parallel

HiFi-GAN Losses



Multi-Scale Discriminator a mixture of sub-discriminators $\mathrm{MSD}[s]$ operating on $\times s$ averaged pooled audio



Multi-Period Discriminator a mixture of sub-discriminators $\mathrm{MPD}[p]$ operating on p-spaced samples for $p = 2, 3, 5, 7, 11$

HiFi-GAN Total Loss

Consider x – ground truth output, $s = \phi(x)$ – condition (mel-spectrogram), losses:

$$\mathcal{L}_{\text{adv}}^{D} = \mathbb{E}_{(x,c)}[(D(x) - 1)^2 + D(G(c))^2]$$

$$\mathcal{L}_{\text{adv}}^{G} = \mathbb{E}_{c}[(D(G(c)) - 1)^2]$$

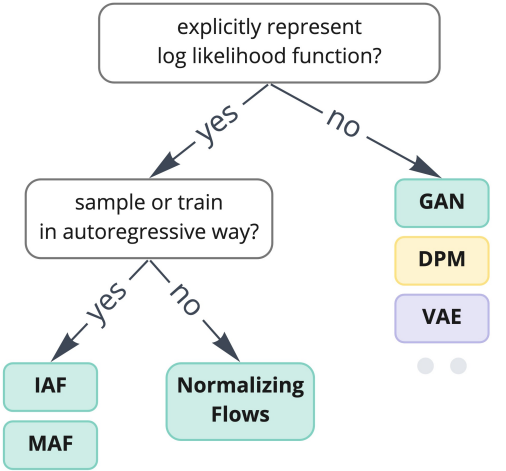
$$\mathcal{L}_{\text{mel}} = \mathbb{E}_{(x,c)}[||\phi(x) - \phi(G(c))||_1]$$

$$\mathcal{L}_{\text{feat}} = \mathbb{E}_{(x,c)}[\sum_{l=1}^{L} ||D^{(l)}(x) - D^{(l)}(G(c))||_1]$$

Final generator and discriminator losses:

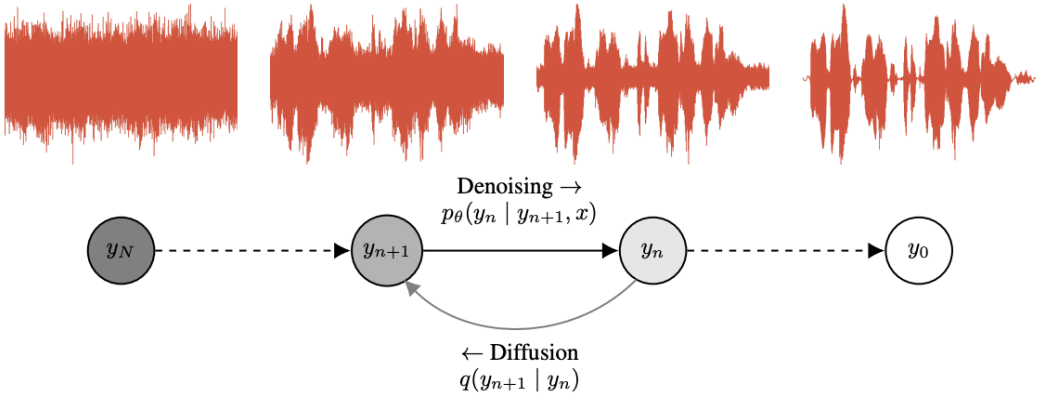
$$\mathcal{L}_G = \mathcal{L}_{\text{adv}}^{G} + \lambda_{\text{feat}}\mathcal{L}_{\text{feat}} + \lambda_{\text{mel}}\mathcal{L}_{\text{me}}, \quad \mathcal{L}_D = \mathcal{L}_{\text{adv}}^{D}$$

Generative Models

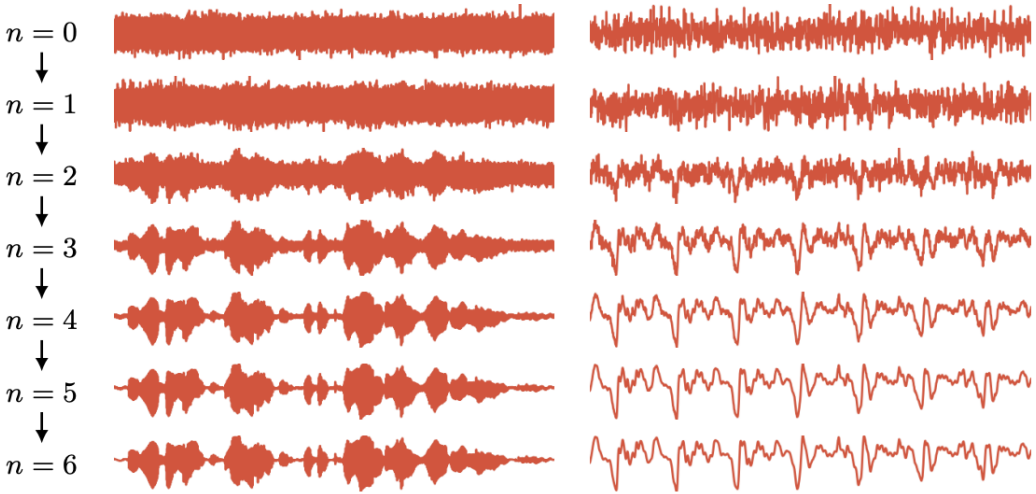


- we discussed how to approximate the data distribution with GANs
- now we'll try to optimize not the distribution but the gradient of it

Diffusion and Denoising Process



WaveGrad Inference Process



Estimation of Diffusion Process

On each iteration we add Gaussian noise:

$$q(y_n|y_{n-1}) := \mathcal{N}(y_n; \sqrt{1-\beta_n}y_{n-1}, \beta_n\mathcal{I})$$

where $\beta_1, \ldots, \beta_N$ – variance schedule.

The diffusion process can be computed for any step n in a closed form:

$$y_n = \sqrt{\overline{\alpha}_n}y_0 + \sqrt{1-\overline{\alpha}_n}\epsilon$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, $\alpha_n := 1 - \beta_n$ and $\overline{\alpha}_n := \prod_{s=1}^{n} \alpha_s$.

Train on pairs (y_0, y_n) and reparameterize the neural network to model ϵ_θ:

$$\mathbb{E}_{\overline{\alpha},\epsilon}[||\epsilon_\theta(\sqrt{\overline{\alpha}}y_0 + \sqrt{1-\overline{\alpha}}\epsilon, x, \sqrt{\overline{\alpha}}) - \epsilon||_1]$$

Algorithm 1 Training. WaveGrad directly conditions on the continuous noise level $\sqrt{\bar{\alpha}}$. l is from a predefined noise schedule.

repeat
  $y_0 \sim q(y_0)$
  $s \sim \text{Uniform}(\{1, \dots, S\})$
  $\sqrt{\bar{\alpha}} \sim \text{Uniform}(l_{s-1}, l_s)$
  $\epsilon \sim \mathcal{N}(0, I)$
  Take gradient descent step on
    $\nabla_\theta \left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}}\, y_0 + \sqrt{1-\bar{\alpha}}\, \epsilon, x, \sqrt{\bar{\alpha}}) \right\|_1$
until converged

Algorithm 2 Sampling. WaveGrad generates samples following a gradient-based sampler similar to Langevin dynamics.

$y_N \sim \mathcal{N}(0, I)$
for $n = N, \dots, 1$ **do**
  $z \sim \mathcal{N}(0, I)$
  $y_{n-1} = \frac{\left(y_n - \frac{1-\alpha_n}{\sqrt{1-\bar{\alpha}_n}}\, \epsilon_\theta(y_n, x, \sqrt{\bar{\alpha}_n})\right)}{\sqrt{\alpha_n}}$
  if $n > 1$, $y_{n-1} = y_{n-1} + \sigma_n z$
end for
return y_0

Lecture Summary