

Lecture 19: Diffusion and Neural Source-Filter

Mark Hasegawa-Johnson

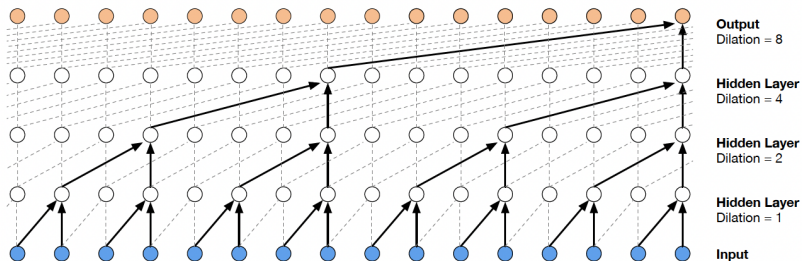
ECE 537: Speech Processing

- 1 Review
- 2 Neural Source-Filter Model
- 3 Diffusion
- 4 Summary

Outline

- 1 Review
- 2 Neural Source-Filter Model
- 3 Diffusion
- 4 Summary

Review: WaveNet



Review: Autoregressive vs Non-Autoregressive Synthesis

- WaveNet is autoregressive. Input can just be upsampled text. Nonlinear sample predictions take care of converting text into waveform samples.
- WaveGlow is non-autoregressive. Start with a white noise signal as long as the desired speech signal. Modify all samples simultaneously with $\mathbf{x} = \mathbf{f}(\mathbf{z})$ optimized to match $\ln p_X(\mathbf{x})$. Repeat for 12 iterations, until it resembles speech.

NSF and Diffusion: Two modifications of the WaveGlow idea, using WaveNet as a backbone

- Neural source-filter: Instead of starting with white noise, start with pitch harmonics.
- Diffusion: Instead of estimating $\mathbf{f}(\mathbf{z})$ directly, estimate $\nabla_{\mathbf{z}} \ln p_t(\mathbf{z})$.

Outline

- 1 Review
- 2 Neural Source-Filter Model
- 3 Diffusion
- 4 Summary

Neural Source-Filter: Input

Assume we have some other neural net that can generate desired f_b =pitch and $\mathbf{s}_b$ =desired spectral features for the b^{th} frame.

$$\mathbf{c}_b = [f_b, \mathbf{s}_b^T]^T$$

$$\mathbf{c}_{1:B} = [\mathbf{c}_1, \dots, \mathbf{c}_B]$$

These are upsampled to the sampling rate, producing per-sample pitch estimates f_t and spectral estimates $\mathbf{s}_t$.

Neural Source-Filter: The source

$$\mathbf{e}_{1:T} = \text{FF} \left(\begin{bmatrix} \mathbf{e}_{1:T}^{(0)} \\ \vdots \\ \mathbf{e}_{1:T}^{(7)} \end{bmatrix} \right)$$

$$\mathbf{e}_t^{(m)} = \begin{cases} 0.01 \sin \left(\sum_{\tau=1}^t \frac{2\pi(m+1)f_{\tau}}{F_s} + \phi \right) + n_t & f_t \neq 0 \\ 111n_t & f_t = 0 \end{cases}$$

$$n_t \sim \mathcal{N}(0, 0.003)$$

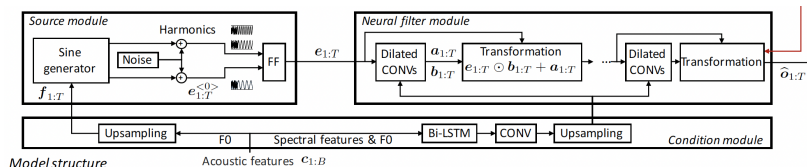
- m is the harmonic index: Wang et al. use the fundamental plus 7 higher harmonics
- The phase, ϕ , is chosen during training to minimize MSE, but is randomized during inference
- The matrix of harmonics is combined using a feedforward network to make $\mathbf{e}_{1:T}$.

Neural Source-Filter: The filter

- Spectral features $\mathbf{s}[t]$ are input to a series of WaveNet-style dilated convolution layers, but these are non-autoregressive: outputs are not fed back to input.
- Output of each layer is scale $\mathbf{b}_{1:T}$ and offset $\mathbf{a}_{1:T}$, applied as

$$\mathbf{b}_{1:T} \odot \mathbf{e}_{1:T} + \mathbf{a}_{1:T}$$

Neural Source-Filter: The inference model



Neural Source-Filter: training criterion

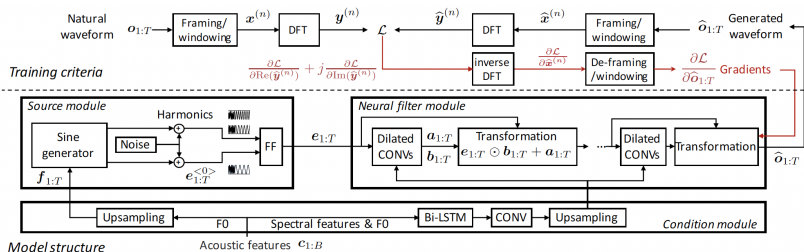
- Take the FFT of both original and synthetic speech
- Compute MSE of the log-magnitude spectrum, and cosine distance of the phase:

$$\mathcal{L}_s = \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K \left[\log \frac{\Re(y_k^{(n)})^2 + \Im(y_k^{(n)})^2}{\Re(\hat{y}_k^{(n)})^2 + \Im(\hat{y}_k^{(n)})^2} \right]$$

$$\mathcal{L}_p = \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K \left[1 - \exp \left(j \left(\hat{\theta}_k^{(n)} - \theta_k^{(n)} \right) \right) \right]$$

- Use inverse-FFT to backpropagate gradients to individual samples

Neural Source-Filter: training method



Neural Source-Filter: Results

- Mean opinion scores > 4 on a 5-point scale, matching WaveNet
- Synthesis is non-autoregressive, so it can be done on a GPU
- Separate manipulation of pitch and spectrum permits explainable speech synthesis (the primary motivation for later uses of NSF, e.g., in the Voice Anonymization challenges).

Outline

- 1 Review
- 2 Neural Source-Filter Model
- 3 Diffusion**
- 4 Summary

Diffusion versus Flow

- Both flow and diffusion start with $\mathbf{z} \sim \mathcal{N}(0, 1)$ (or any other simple source – the source could be a neural source-filter model), then iteratively modify it over several layers until speech is achieved.
- Flow computes $\mathbf{x} = \mathbf{f}(\mathbf{z})$; parameters of $\mathbf{f}$ optimized to minimize $\mathcal{L} = -\ln p_X(\mathbf{x})$.
- Diffusion computes

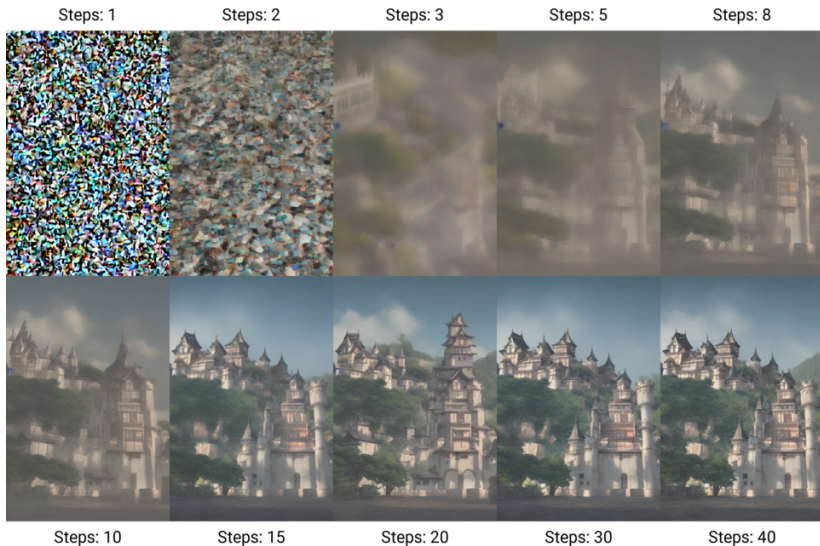
$$-d\mathbf{z} = \frac{\beta}{2} (\mathbf{z} + \nabla_{\mathbf{z}} \ln p(\mathbf{z}))$$

where the network computing $\nabla_{\mathbf{z}} \ln p(\mathbf{z})$ is trained to estimate this quantity with minimum MSE.

Diffusion and Flow: Two different views of the same process

- Flow: Start with Gaussian white noise, and gradually modify the signal so that the pdf better resembles the desired signal
- Diffusion: Start with a signal that has been corrupted by Gaussian white noise, and gradually remove the noise in order to generate the signal

Diffusion example: Image created by Stable Diffusion v1-5



CC-BY-SA 4.0: https://commons.wikimedia.org/wiki/File:X-Y_plot_of_algorithmically-generated_AI_art_of_European-style_castle_in_Japan_demonstrating_DDIM_diffusion_steps.png

Training data = pairs (original, noise)

- Define the input signal (speech, image, etc.) to be $\mathbf{z}_0$.
- Choose an instance of Gaussian noise and call it $\mathbf{z}_1$.
- Generate a “noisy” signal $\mathbf{z}_t, t \in [0, 1]$, by gradually dissolving the original signal and replacing it with Gaussian noise:

$$p(\mathbf{z}_t | \mathbf{z}_0) \sim \mathcal{N}(\rho(\mathbf{z}_0, t), \Sigma_t)$$

$$\rho(\mathbf{z}_0, t) = e^{-\frac{1}{2} \int_0^t \beta_s ds} \mathbf{z}_0$$

$$\Sigma_t = I - e^{-\int_0^t \beta_s ds}$$

where β_t is the noise schedule.

Trained model = score (log pdf gradient) or data (original signal)

- Train a neural network to estimate the score, $\nabla_{\mathbf{z}} \ln p(\mathbf{z}_t)$, or to compute an estimate $\hat{\mathbf{z}}_0$ of the data $\mathbf{z}_0$.
- Given knowledge of $\mathbf{z}_0$ and $\mathbf{z}_1$ during training (or of $\hat{\mathbf{z}}_0$ and $\mathbf{z}_1$ during test), we can find $\mathbf{z}_t$, $\rho(\mathbf{z}_0, t)$ and Σ_t . From this, if the pdf is still Gaussian, then the expected score is

$$E [\nabla \ln p(\mathbf{z}_t)] = \Sigma_t^{-1}(\rho(\mathbf{z}_0, t) - \mathbf{z}_t)$$

so the training criterion is

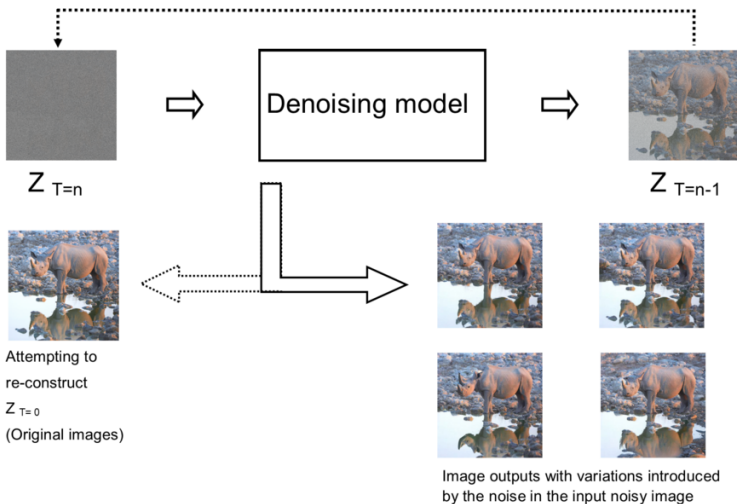
$$\mathcal{L} = \|\mathbf{z}_0 - \hat{\mathbf{z}}_0\|^2 + \|\Sigma_t^{-1}(\rho(\hat{\mathbf{z}}_0, t) - \mathbf{z}_t) - \nabla \ln p(\mathbf{z}_t)\|^2$$

Inference: Compute the signal from the noise

- During inference, a random $\mathbf{z}_1$ is generated, then the signal is generated from it as

$$-d\mathbf{z} = \frac{\beta}{2} (\mathbf{z} + \nabla_{\mathbf{z}} \ln p(\mathbf{z}))$$

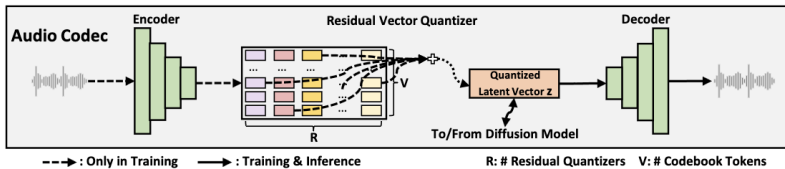
Inference: Compute the signal from the noise



Text-to-speech: Latent synthesis + Codec

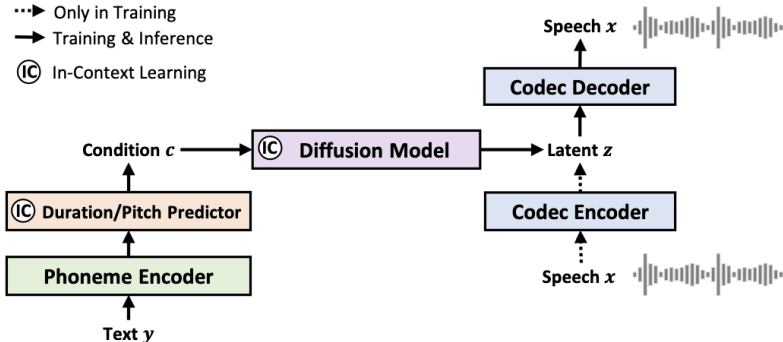
- Some TTS converts text directly to waveforms (e.g., WaveNet)
- Most TTS uses a two-stage process:
 - 1 **Latent synthesis:** Convert text to a sequence of latent feature vectors, e.g., mel-spectrograms or (as in Shen et al., 2023) the latent vectors of a neural audio codec
 - 2 **Codec:** Convert the latents (Shen et al., 2023) or spectrogram (Wang et al., 2019) into a waveform.

How to train a neural audio codec



Natural Speech 2: Overall Training Pipeline

- ⋯→ Only in Training
→ Training & Inference
Ⓢ In-Context Learning



Outline

1 Review

2 Neural Source-Filter Model

3 Diffusion

4 Summary

Summary

- Neural source-filter: Instead of starting with white noise, start with pitch harmonics.
- Diffusion: Instead of estimating $\mathbf{f}(\mathbf{z})$ directly, estimate $\nabla_{\mathbf{z}} \ln p_t(\mathbf{z})$.