

Lecture 18: Wavenet and Waveglow

Mark Hasegawa-Johnson

ECE 537: Speech Processing

1 Review: LPC

2 WaveNet

3 WaveGlow

4 Summary

Outline

1 Review: LPC

2 WaveNet

3 WaveGlow

4 Summary

Review: LPC

- LPC models the resonant structure of speech using a p^{th} -order linear predictor:

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

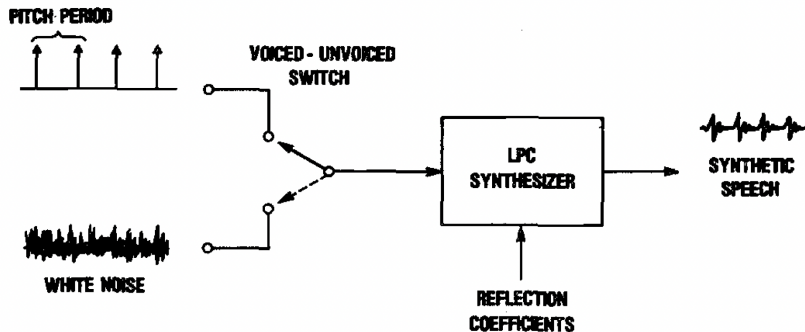
- Minimizing the mean-squared error results in p linear equations in p unknowns, which can be solved as

$$\mathbf{r} = \mathbf{R}\mathbf{a}$$

- In almost all cases, the formant frequencies (and their bandwidths) are among the poles of the LPC synthesis filter:

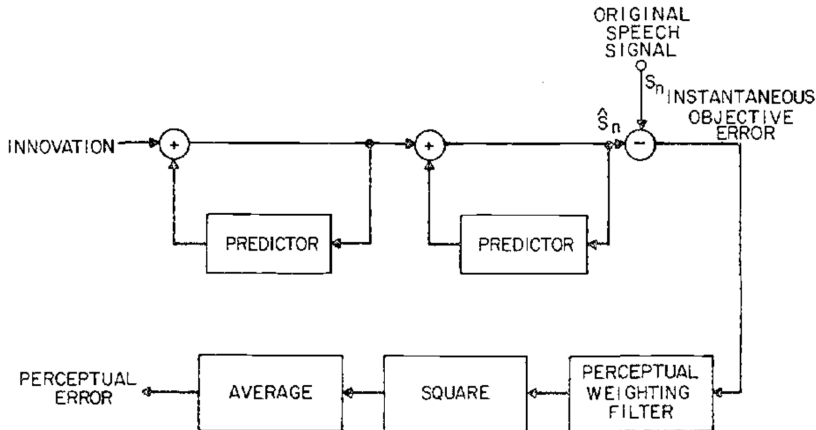
$$H(z) = \frac{G}{A(z)} = \frac{G}{1 - \sum_k a_k z^{-k}} = \frac{Gz^p}{\prod_{i=1}^p (z - z_i)}$$

The LPC-10 Vocoder



Atal and Remde, 1982, Fig. 1

Code-Excited LPC (CELP)



Schroeder and Atal, 1985, Fig. 2

What about text-to-speech (TTS)?

- CELP can produce natural-sounding resynthesis of a known speech waveform from a low-bit-rate representation. If you're trying to generate a waveform that never previously existed (e.g., text-to-speech), CELP doesn't help you.
- Formant-based synthesis (Stephen Hawking's voice) and LPC-10 vocoder synthesis are intelligible, but not natural.
- Concatenating pre-recorded waveform chunks can be extremely natural, intelligible, and low-cost, if you want the provided voice and emotion (e.g., LJSpeech). It was the industry standard until well into the 2020s, replaced by neural synthesis only when commercial vendors provided neural voices that were cheaper.

Outline

1 Review: LPC

2 WaveNet

3 WaveGlow

4 Summary

WaveNet Overview

Wavenet (2016) was the first neural TTS with naturalness above 4.0 on a 5-point scale. It introduced four key innovations, and had one key problem:

- Innovation: Nonlinear prediction
- Innovation: Generates discrete tokens
- Innovation: Dilated causal convolutions
- Problem: Autoregressive synthesis

Innovation: Nonlinear prediction

- LPC performs linear prediction:

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

- WaveNet performs gated nonlinear prediction:

$$\begin{aligned} \mathbf{z}_k[n] &= \tanh \left(\sum_{m=1}^p \mathbf{w}_{f,k}[m] \mathbf{x}_{k-1}[n-m] \right) \sigma \left(\sum_{m=1}^p \mathbf{w}_{g,k}[m] \mathbf{x}_{k-1}[n-m] \right) \\ &\equiv \tanh (\mathbf{W}_{f,k} * \mathbf{x}_{k-1}) \odot \sigma (\mathbf{W}_{g,k} * \mathbf{x}_{k-1}) \end{aligned}$$

... where σ is the logistic sigmoid, f and g denote filter and gate, respectively, k is the layer index, $\odot$ is scalar multiplication, and $*$ is convolution.

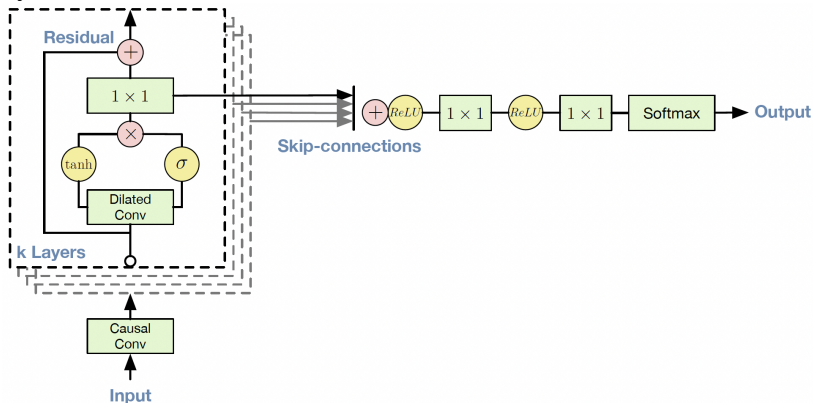
Nonlinear prediction with inputs

For TTS, WaveNet uses a preprocessing network to upsample the text symbol sequence, $\mathbf{h} = [h[1], \dots, h[L]]$, to a time series $\mathbf{Y} = [\mathbf{y}[1], \dots, \mathbf{y}[N]]$ with the same number of samples as the desired speech signal, then uses that input sequence as an auxiliary input of every layer:

$$\mathbf{z}_k = \tanh(\mathbf{W}_{f,k} * \mathbf{x}_{k-1} + \mathbf{V}_{f,k} * \mathbf{y}) \odot \sigma(\mathbf{W}_{g,k} * \mathbf{x}_{k-1} + \mathbf{V}_{g,k} * \mathbf{y})$$

Nonlinear residual prediction with inputs

The most effective TTS used gated nonlinear prediction followed by a 1×1 convolution, then a residual connection:



Innovation: Generates discrete tokens

- All previous neural speech synthesizers, like LPC, generate continuous-valued speech samples $\hat{s}[n]$, and are trained to minimize MSE:

$$\mathcal{L} = \sum_{n=0}^{N-1} (s[n] - \hat{s}[n])^2$$

- WaveNet generated discrete speech codewords $x[n]$, and is trained to minimize cross-entropy loss:

$$\mathcal{L} = - \sum_{n=0}^{N-1} \log p(x[n] | x[1], \dots, x[n-1])$$

Discrete distribution: Quantized speech samples

- WaveNet modeled each speech sample as a discrete random variable, with either 256 quantized levels (8-bit mu-law encoding) or 16384 quantized levels (16-bit linear PCM).
- The output of WaveNet is a softmax over speech sample values:

$$p(x[n] = x) = \frac{\exp \mathbf{w}_x^T \mathbf{v}[n]}{\sum_y \exp \mathbf{w}_y^T \mathbf{v}[n]}$$

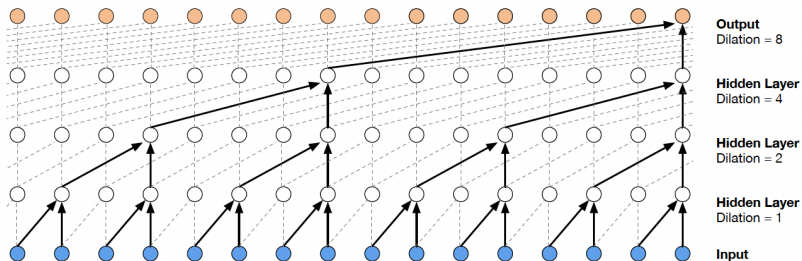
... where $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_{256}]^T$ is WaveNet's output-layer weight matrix, and $\mathbf{v}[n]$ is the penultimate-layer activation at time n .

Innovation: Dilated Causal Convolutions

- LPC predicted each sample using the $p \approx 10$ preceding samples, in a single linear layer.
- WaveNet predicts $\mathbf{z}_k[n]$ using only two inputs: $\mathbf{x}_{k-1}[n]$ and $\mathbf{x}_{k-1}[n - n_k]$.
- The delay is **dilated**:

$$n_k = 2^{k-1}$$

Innovation: Dilated Causal Convolutions



Problem: Autoregressive computation

- Each speech sample is highly correlated with preceding samples
- Taking a cue from LPC, WaveNet leveraged this correlation by predicting each sample from the 2^L preceding samples, where $L = 10$ is the number of layers.

Autoregressive prediction during training: Teacher forcing

- Network weights are trained using a method called “teacher forcing.” The loss function uses natural speech samples $x[n - 1024], \dots, x[n]$ to compute the loss function

$$\mathcal{L} = - \sum_{n=0}^{N-1} \mathcal{L}[n]$$

$$\mathcal{L}[n] = \log p(x[n] | x[n - 1024], \dots, x[n - 1])$$

- $\mathcal{L}$ can be computed efficiently using parallel architectures (GPU) because all of the $\mathcal{L}[n]$ can be computed and differentiated in parallel

Autoregressive prediction during inference

- During inference, however, parallel computation is impossible. Samples are computed one at a time; each sample required as input to generate the next.
- WaveNet could therefore use GPU during training, but must use much slower CPU sequential operations during inference (to generate speech signals).

Outline

- 1 Review: LPC
- 2 WaveNet
- 3 WaveGlow**
- 4 Summary

WaveGlow: Non-Autoregressive Neural Speech Synthesis

- WaveGlow was like WaveNet, but non-autoregressive.
- It first generates a complete length- N sequence of unit-normal Gaussian random samples, $\mathbf{z} = [z_1, \dots, z_N]^T$,

$$z[n] \sim \mathcal{N}(0, 1)$$

- It then generates speech samples $\mathbf{x} = [x_1, \dots, x_N]^T$ all at once using k layers:

$$\mathbf{x} = \mathbf{f}_1(\dots \mathbf{f}_k(\mathbf{z}) \dots)$$

- The real trick: How to train the parameters of those layers?

Functions of random variables: Example

- Suppose $z \sim \mathcal{U}(0, 1)$, i.e.,

$$p_Z(z) = \begin{cases} 1 & 0 \leq z < 1 \\ 0 & \text{otherwise} \end{cases}$$

- Suppose $X = 2Z$.
- Then it follows that

$$p_X(x) = \begin{cases} \frac{1}{2} & 0 \leq x < 2 \\ 0 & \text{otherwise} \end{cases}$$

Functions of random variables: General case

- Suppose $\mathbf{z} \sim p_Z(\mathbf{z})$.
- Suppose $\mathbf{x} = \mathbf{f}(\mathbf{z})$ and $\mathbf{z} = \mathbf{f}^{-1}(\mathbf{x})$, for any invertible $\mathbf{f}$.
- Then it follows that

$$p_X(\mathbf{x}) = p_Z(\mathbf{f}^{-1}(\mathbf{x})) \odot |\det(\mathbf{J}(\mathbf{f}^{-1}(\mathbf{x})))|$$

... where $\det$ is determinant and $\mathbf{J}$ is the Jacobian,

$$\mathbf{J}(\mathbf{f}^{-1}(\mathbf{x})) = \begin{bmatrix} \frac{\partial f_1^{-1}}{\partial x_1} & \cdots & \frac{\partial f_1^{-1}}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N^{-1}}{\partial x_1} & \cdots & \frac{\partial f_N^{-1}}{\partial x_N} \end{bmatrix}$$

Signal generation using flow networks

- We can train a network to generate complete speech sequences, $\mathbf{x}$, from Gaussian random signals, as long as each layer is invertible.
- The parameters of each layer, $\mathbf{f}_k$, are trained to maximize the probability that $\mathbf{x}$ was generated from some $\mathbf{z}$ distributed as $p_Z(\mathbf{z})$ by maximizing:

$$\ln p_X(\mathbf{x}) = \ln p_Z(\mathbf{f}_k^{-1}(\cdots \mathbf{f}_1^{-1}(\mathbf{x}))) + \sum_{i=1}^k \ln |\det(\mathbf{J}(\mathbf{f}_i^{-1}(\mathbf{x})))|$$

How do you make the layers invertible?

- Invertible nonlinearities can be introduced by splitting the vector in two, $\mathbf{x}^T = [\mathbf{x}_a^T, \mathbf{x}_b^T]$, keeping half unchanged, and using the unchanged half to change the other:

$$\mathbf{f}^{-1}(\mathbf{x}) = [\mathbf{x}_a, \mathbf{x}_b \odot \mathbf{s}(\mathbf{x}_a) + \mathbf{t}(\mathbf{x}_a)]$$

$$\mathbf{f}(\mathbf{z}) = [\mathbf{z}_a, (\mathbf{z}_b - \mathbf{t}(\mathbf{z}_a))/\mathbf{s}(\mathbf{z}_a)]$$

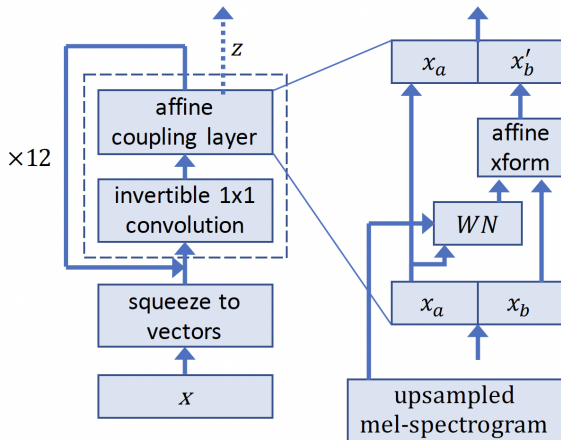
$$\mathbf{J}(\mathbf{f}^{-1}) = \text{diag}(\mathbf{s})$$

- The problem with splitting is that $\mathbf{x}_a$ never gets changed. This problem is solved with a 1×1 convolution (a.k.a. linear transform), which is invertible if the matrix is non-singular:

$$\mathbf{f}^{-1}(\mathbf{x}) = \mathbf{W}\mathbf{x}, \quad \mathbf{f}(\mathbf{z}) = \mathbf{W}^{-1}\mathbf{z}, \quad \mathbf{J}(\mathbf{f}^{-1}) = \mathbf{W}$$

Maximizing $|\det(\mathbf{J})|$ means maximizing $|\det(\mathbf{W})|$, so if we start with nonsingular $\mathbf{W}$, it stays nonsingular.

WaveGlow overview



Historical Summary

- WaveGlow generated waveforms from mel spectrograms
- Modern authors use Flow-based synthesis to generate mel-spectrograms from text
- Generating waveforms from mel spectrograms (the “vocoder” part of TTS) is now often done using GAN-based methods, though Flow methods are still sometimes used

Outline

1 Review: LPC

2 WaveNet

3 WaveGlow

4 Summary

Summary

- WaveNet
 - Innovations: Gated nonlinear residual prediction, Discrete generation of quantized samples, Dilated convolution
 - Problem: Autoregressive, therefore inference is slow
- WaveGlow
 - Innovations: First application of Flow to TTS, which solved the problem of non-autoregressive waveform generation, and is still SoTA or near-SoTA for mel-spectrogram generation
 - Why GAN has mostly replaced Flow for vocoders: GAN can use a wider variety of training criteria, not just log probability