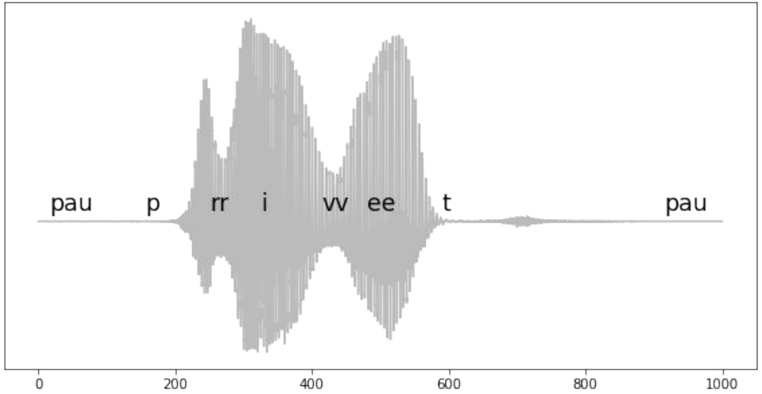
Text-to-speech synthesis

Vladimir Kirichenko, Yandex

Problem Definition

“Привет!”



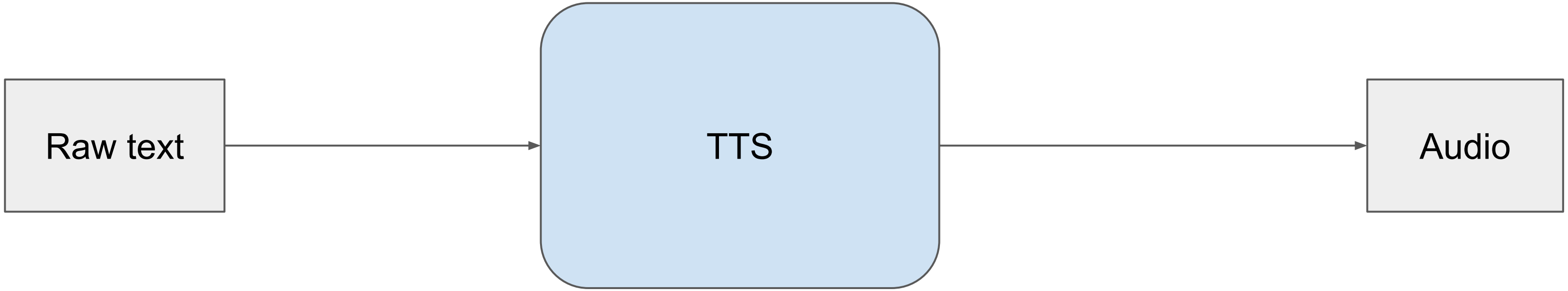
Problem Definition

Input:

Raw text (with digits, strange symbols, abbreviations etc.)

Output:

PCM audio (wav)



Quality Assessment

- No “right” or “wrong” output.
- Subjective perception of the quality.
- Different kinds of errors

Mean Opinion Score

1. Remove “hard” errors (words mispronunced, wrong intonation)

2. Ask assessors to rate each example on a scale of 1 to 5

3. Average the results

Mean Opinion Score

Table 1. MOS (ACR) scores

Rating	Quality	Distortion
5	Excellent	Imperceptible
4	Good	Just perceptible, but not annoying
3	Fair	Perceptible and slightly annoying
2	Poor	Annoying, but not objectionable
1	Bad	Very annoying and objectionable

Mean Opinion Score - CrowdMOS

- Scores determination and screening
- CI-s estimation
- Crowdsourcing MOS estimation with Amazon MTurk

https://www.microsoft.com/en-us/research/wp-content/uploads/2011/05/0002416.pdf

Mean Opinion Score

Pros:

- Absolute scale
- Good for sound quality estimation

Cons:

- Scores depends a lot on the crowdsource platform
- “Hard” errors are not considered
- Bad for intonation/pronunciation assessment

MUSHRA (MUltiple Stimuli with Hidden Reference and Anchor)

1. For each estimated example provide “reference” with natural speech

2. Ask assessors how similar are reference and example sentence
   (on a scale of 1 to 5)

3. Average the results

MUSHRA

Pros:

- Absolute scale
- More stable than MOS
- Good for intonation and speaker similarity assessment

Cons:

- Depends a lot on a test set
- Expensive to change test set
- Pronunciation issues aren’t considered

(pronunciation) Sentence Error Rate

1. Ask assessors to mark sentences with any “hard” errors

2. Average the results.

(pronunciation) Sentence Error Rate

Pros:

- Considers all “hard” errors
- Cheap collection of test set
- Good to estimate rate of pronunciation errors

Cons:

- Intonation errors rate is very noisy
- Not sensitive to sound quality issues

Side by Side Test

1. Get pairs of sentences (with the same phrase)
   from two different syntheses

2. Ask assessors to choose the most preferable audio for each pair

3. Average the votes for each synthesis

Side by Side Test

Pros:

- Sensitive to all kinds of issues
- Binomial test could help to estimate confidence of the SbS measurement
- Good to compare syntheses quality

Cons:

- Relative scale
- Noisy in the case of almost-equal syntheses

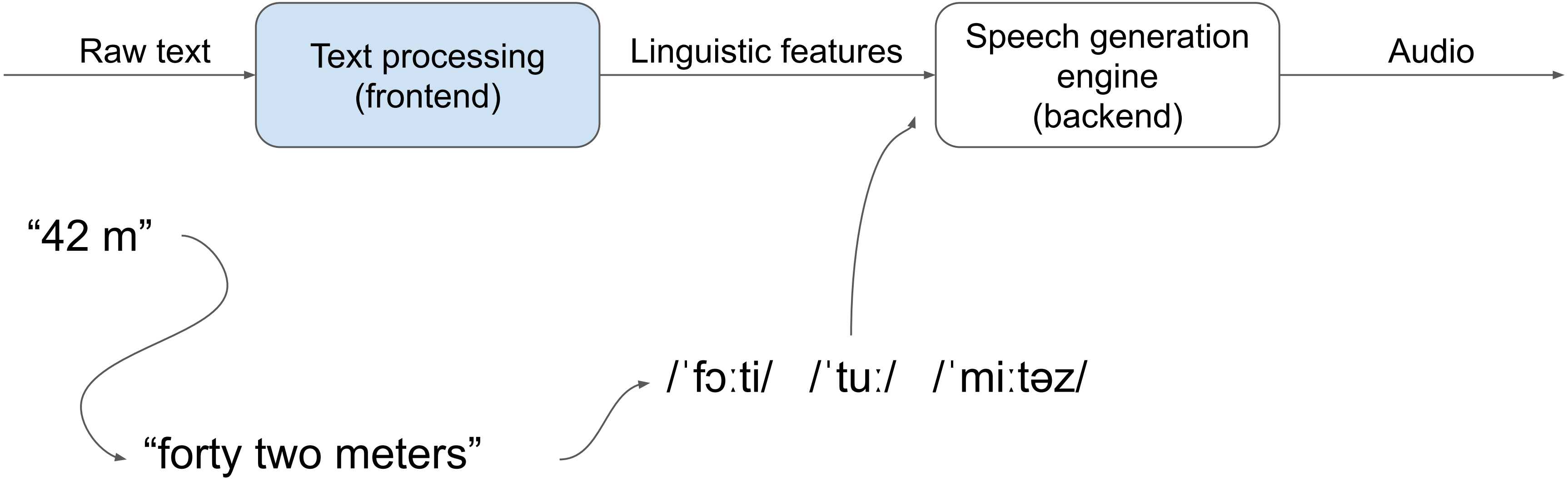
Datasets

- LJ Speech
  - https://keithito.com/LJ-Speech-Dataset/
  - ~24h of 1 speaker (EN)
- VCTK
  - https://homepages.inf.ed.ac.uk/jyamagis/page3/page58/page58.html
  - ~44h of 109 speakers (EN)
- M-AILABS
  - https://www.caito.de/2019/01/the-m-ailabs-speech-dataset/
  - ~1000h, 8 languages, 16kHz

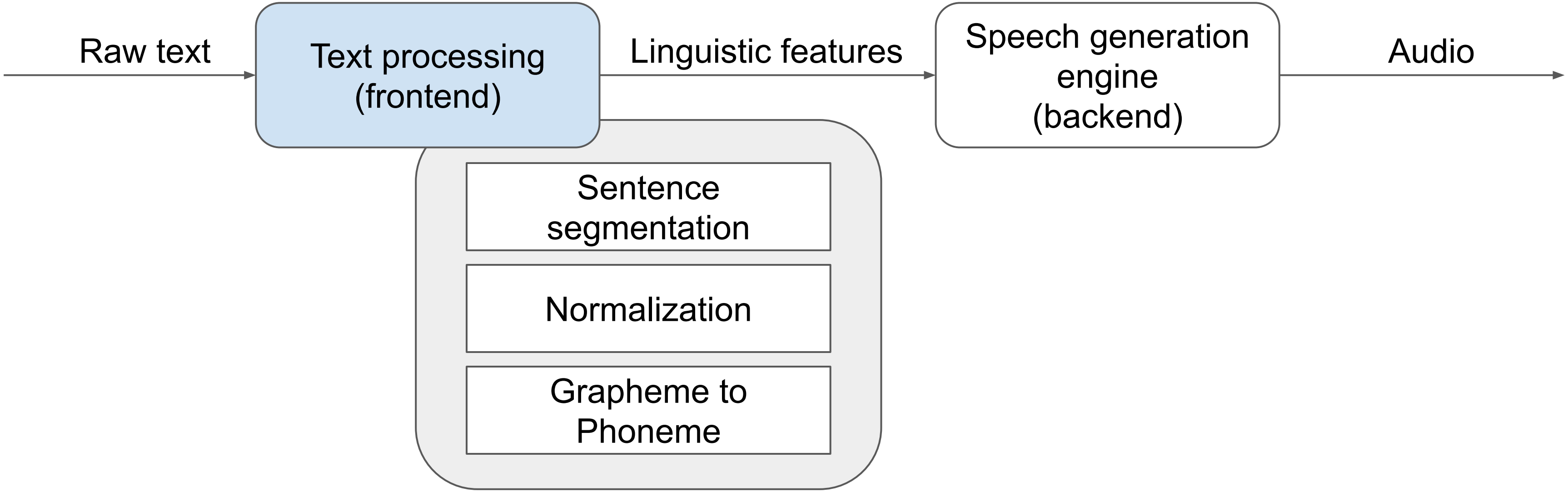
- Closed companies datasets

TTS Pipeline

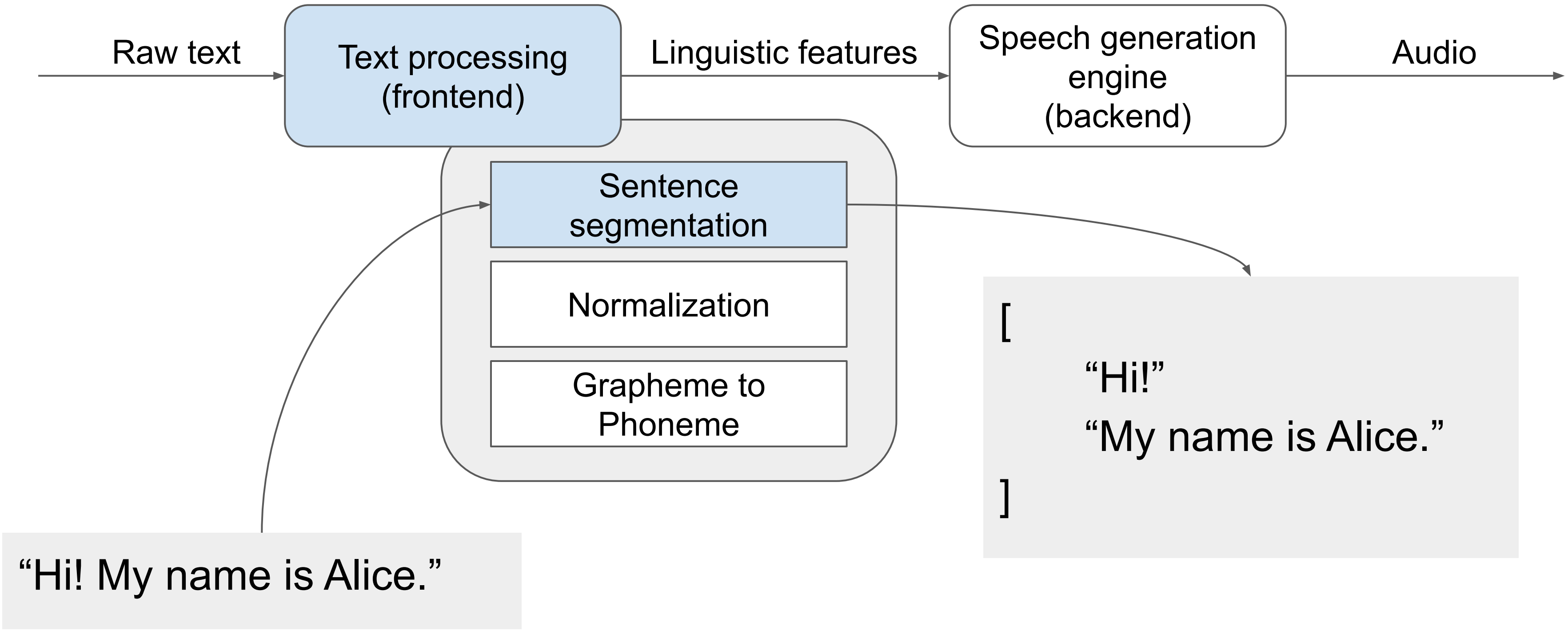
Preprocessor



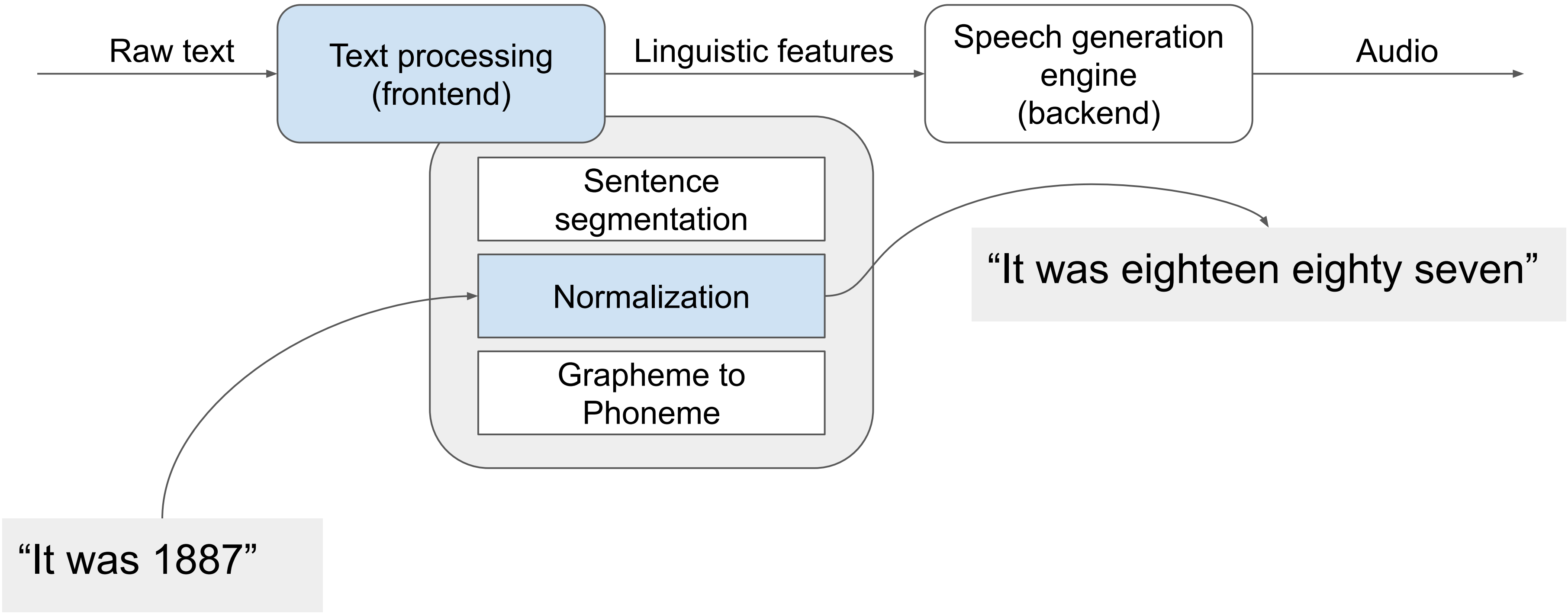
Preprocessor



Sentence tokenization



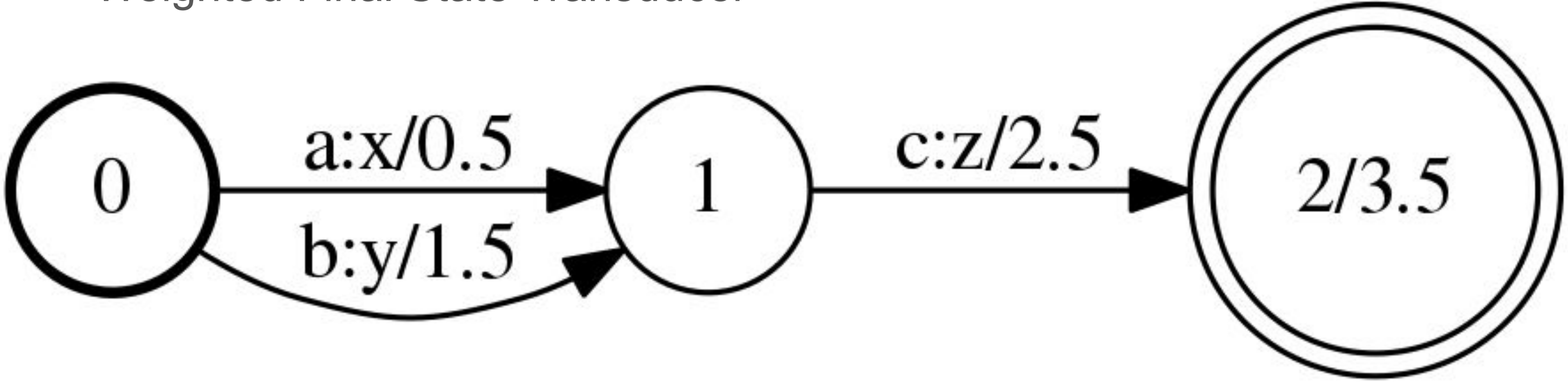
Normalizer



Raw text
Text processing (frontend)
Linguistic features
Speech generation engine (backend)
Audio
Sentence segmentation
Normalization
Grapheme to Phoneme
“It was eighteen eighty seven”
“It was 1887”

WFST Normalizer

Weighted Final State Transducer



WFST Normalizer

Rules are weighted substitutions:

- `s/\b1\b/one/w=0.5` - replace “1” with “one”

- `s/\b1\s*st\b/first/w=1.0` - if it is not followed by “st”

WFST Normalizer

Context problem:

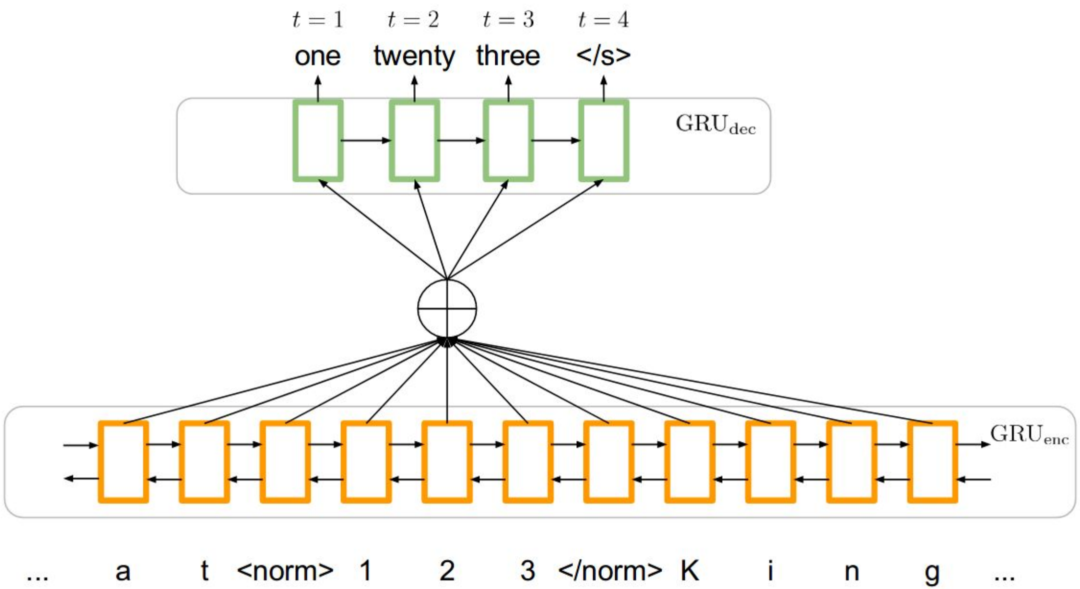
- “Дом 3 к. 3” - “дом три кубические копейки”

- “Нужно 1800 г.” - “Нужно тысяча восьмисотый год”

Rules could interfere:

- “Корпус.” - “корпустроение”

Neural Normalizer



https://storage.googleapis.com/pub-tools-public-publication-data/pdf/17a20b71a2c09100daaef5a8c39eeb930a7017f6.pdf

Neural Normalizer

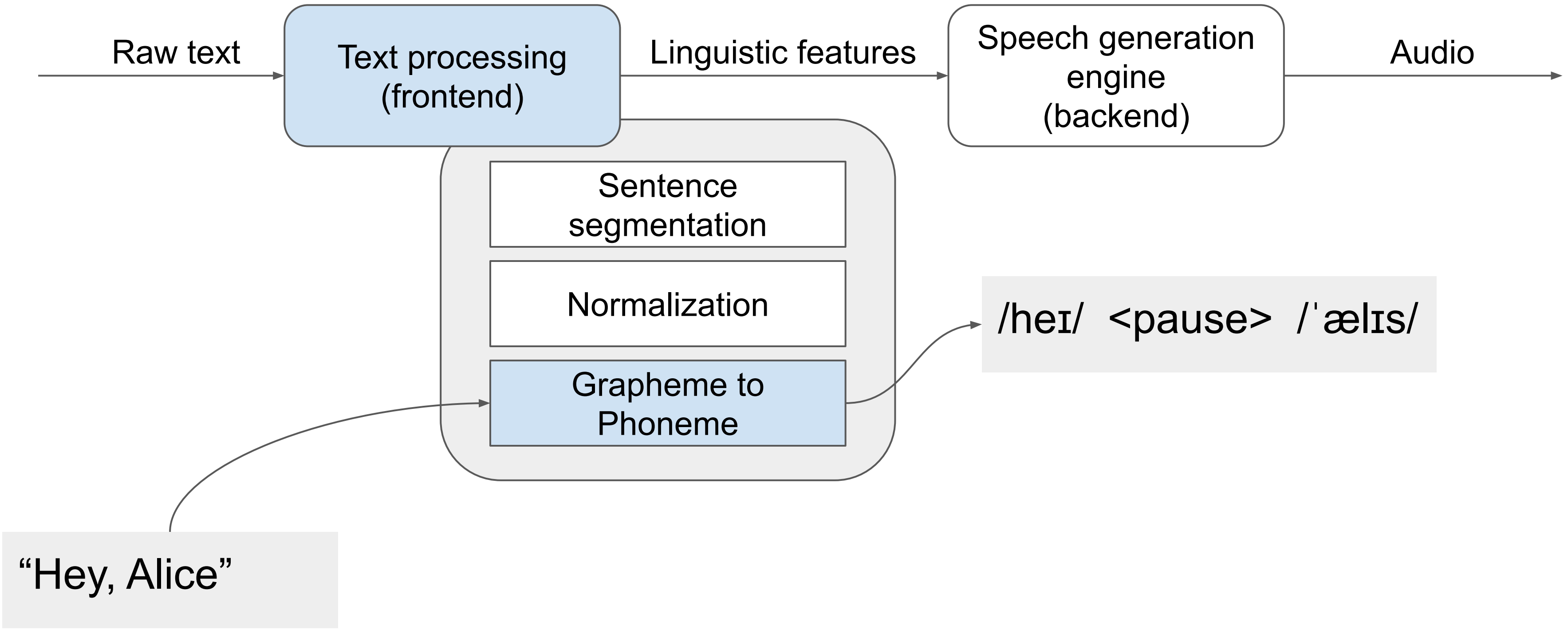
Pros:

- Has no context problems
- No need to write code for new case, just add more data

Cons:

- Need a lot of data (10^6-10^7 parallel sentences)
- Slower than WFST
- Impossible to fix intricate cases manually

Grapheme to Phoneme



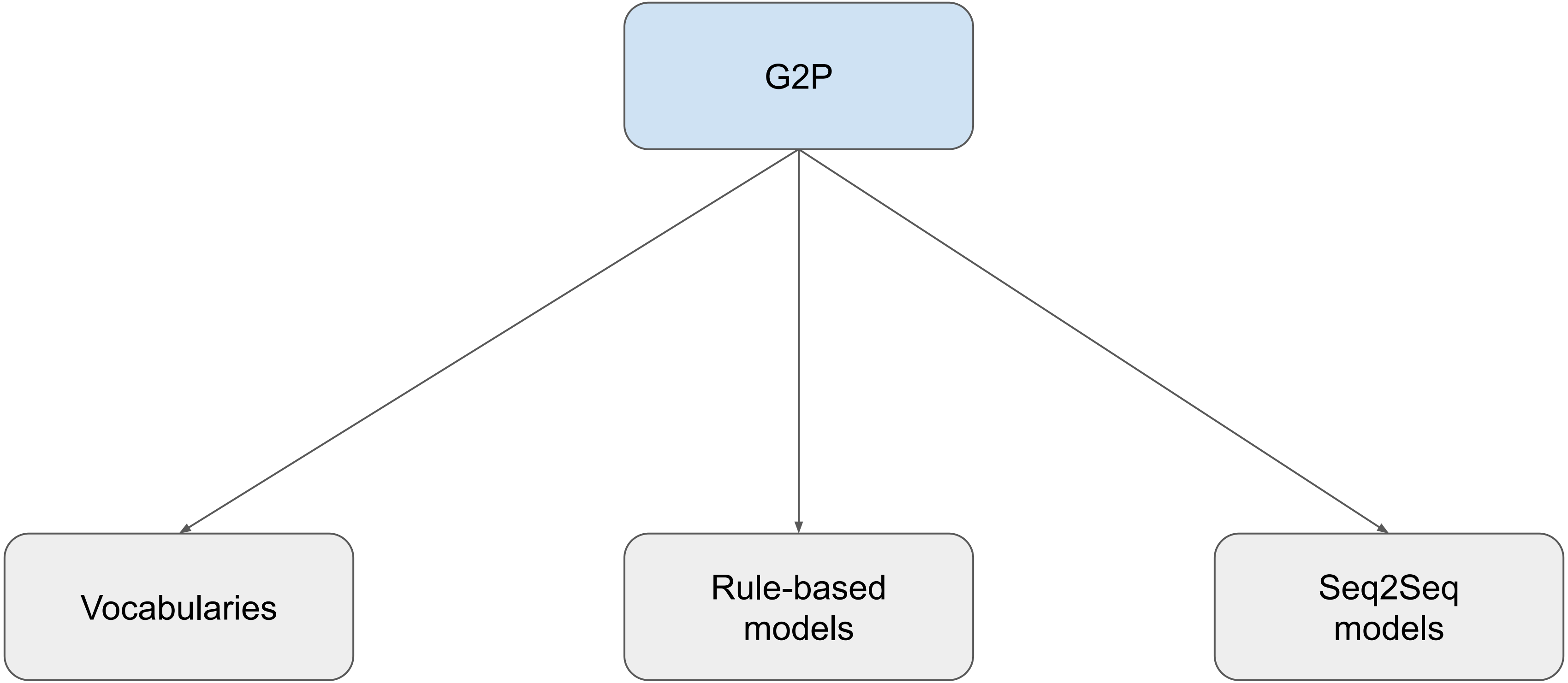
G2P Issues

- Homographs
  “Increase” - /ɪnˈkriːs/ or /ˈɪnkriːs/ ?

- Foreign words
  “A short poem **à la** Ogden Nash” - /ɑ lɑ/

- Abbreviations
  “CI” - /siː/ /aɪ/

G2P Components



Phonemes or Texts?

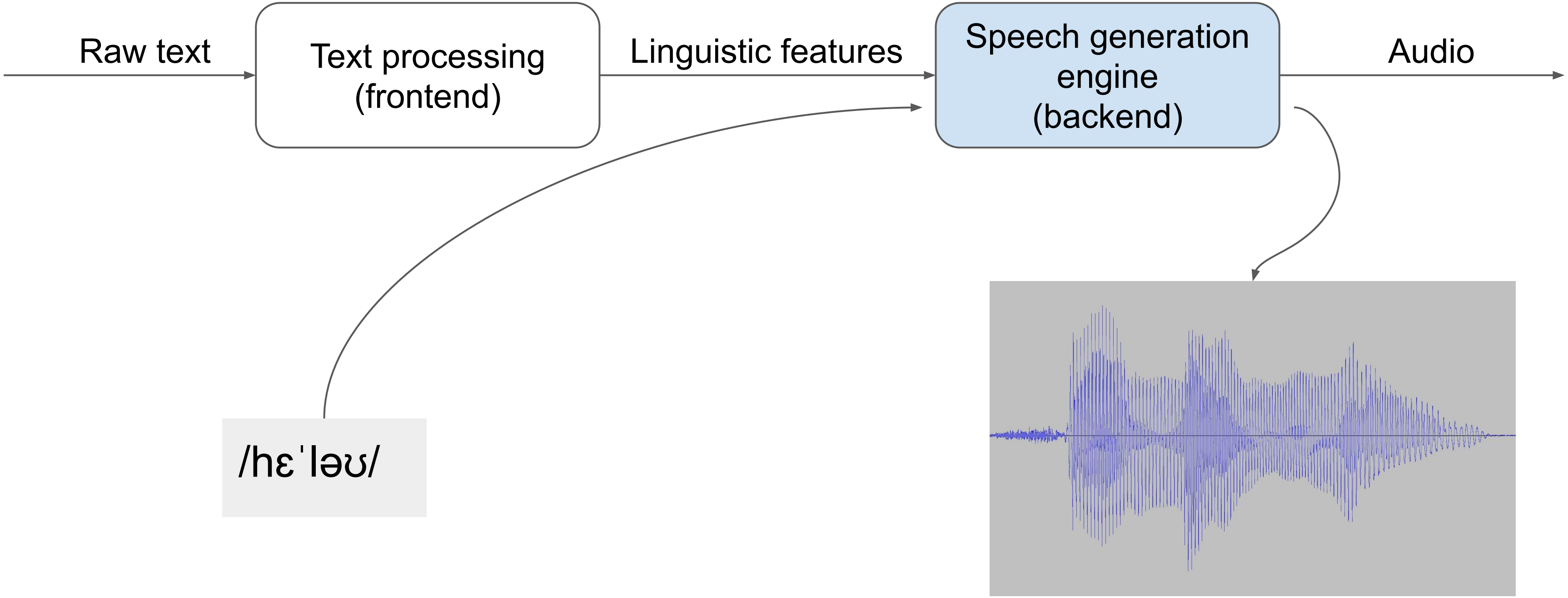
G2P Pros:

- A separate G2P engine can serve for several backends
- Easy to control and fix certain words pronunciation

G2P Cons:

- More expensive markup than text
- Additional components - additional bugs
- Additional information in text (e.g. punctuation) is lost
- Not all languages require G2P

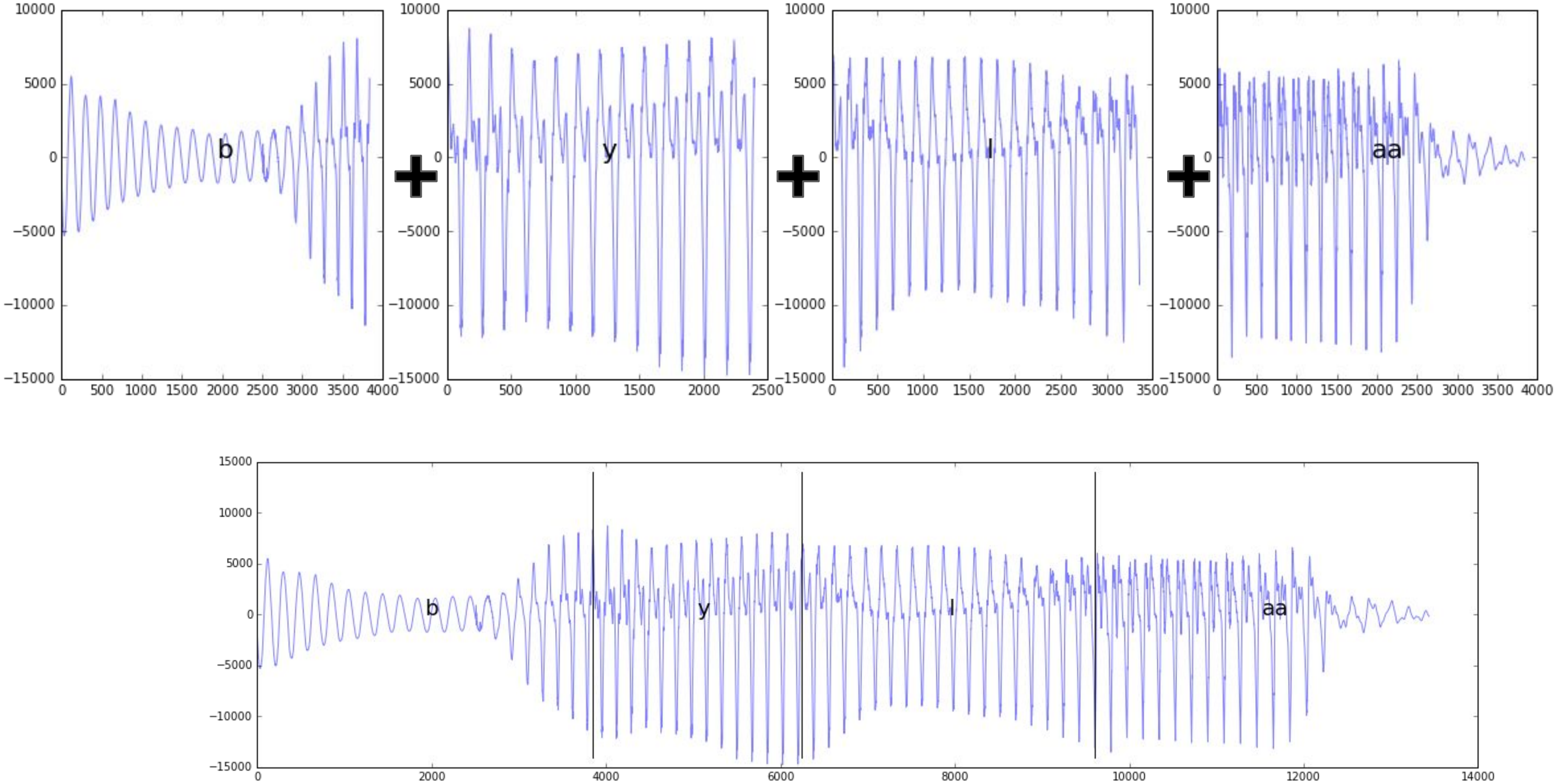
Engine


Raw text
Text processing
(frontend)
Linguistic features
Speech generation
engine
(backend)
Audio
/hɛˈləʊ/

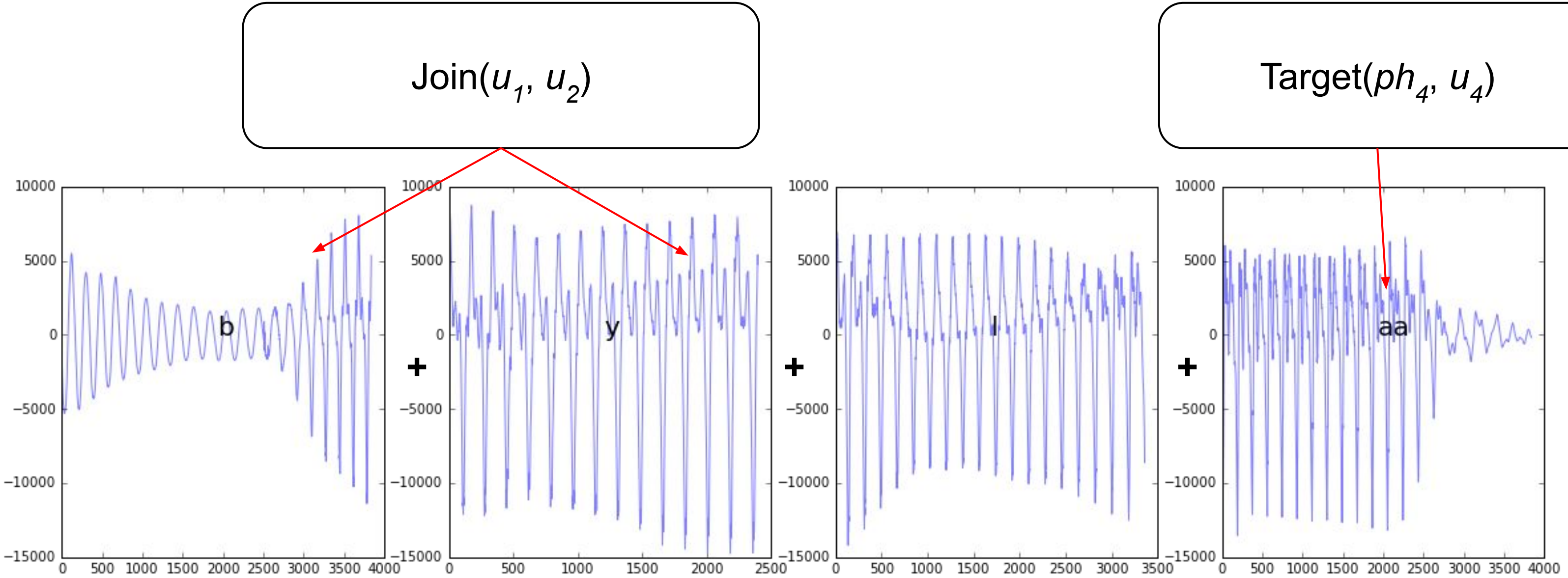
Dense PCM

- Duration of phoneme (in Russian) ~ 20-150 ms
- Duration of 22.05kHz PCM sample ~ 0.045 ms

Concatenative synthesis



Unit selection



$$\sum_i \left(Join\left(u_i, u_{i+1}\right) + Target\left(ph_i, u_i\right)\right) \to \min_{\{u\}}$$

*Deep Learning for Siri Voice
http://machinelearning.apple.com/2017/08/06/siri-voices.html

Unit selection

Pros:

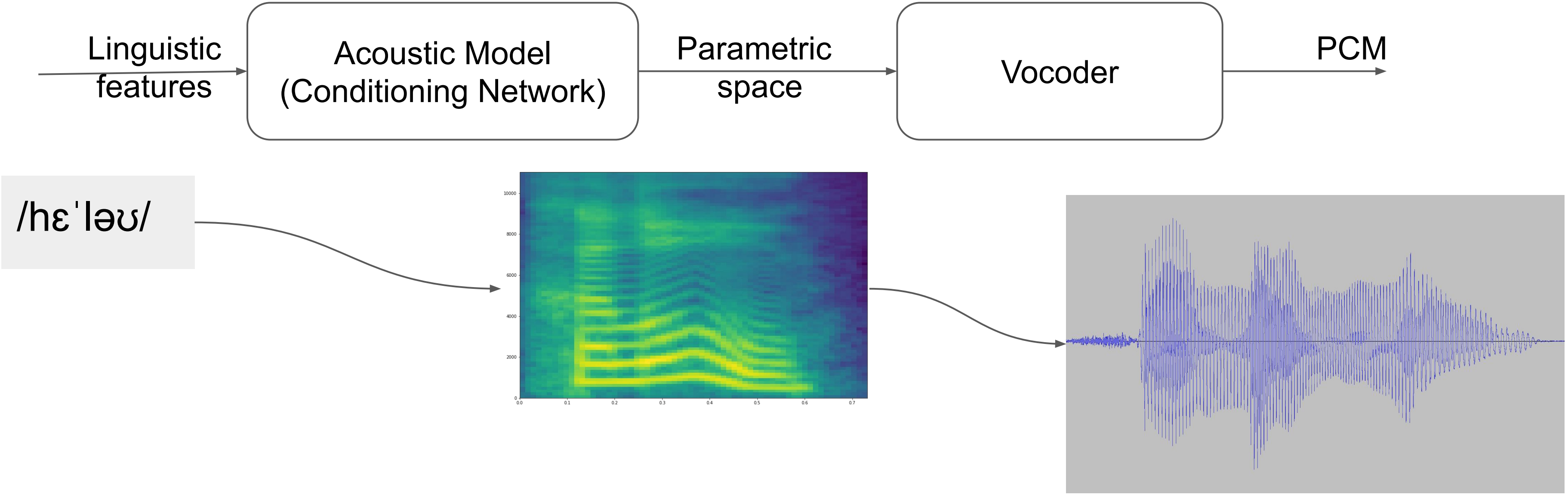
- Very fast synthesis, even with a high sample rate
- Sound quality inside units is equal to the original records

Cons:

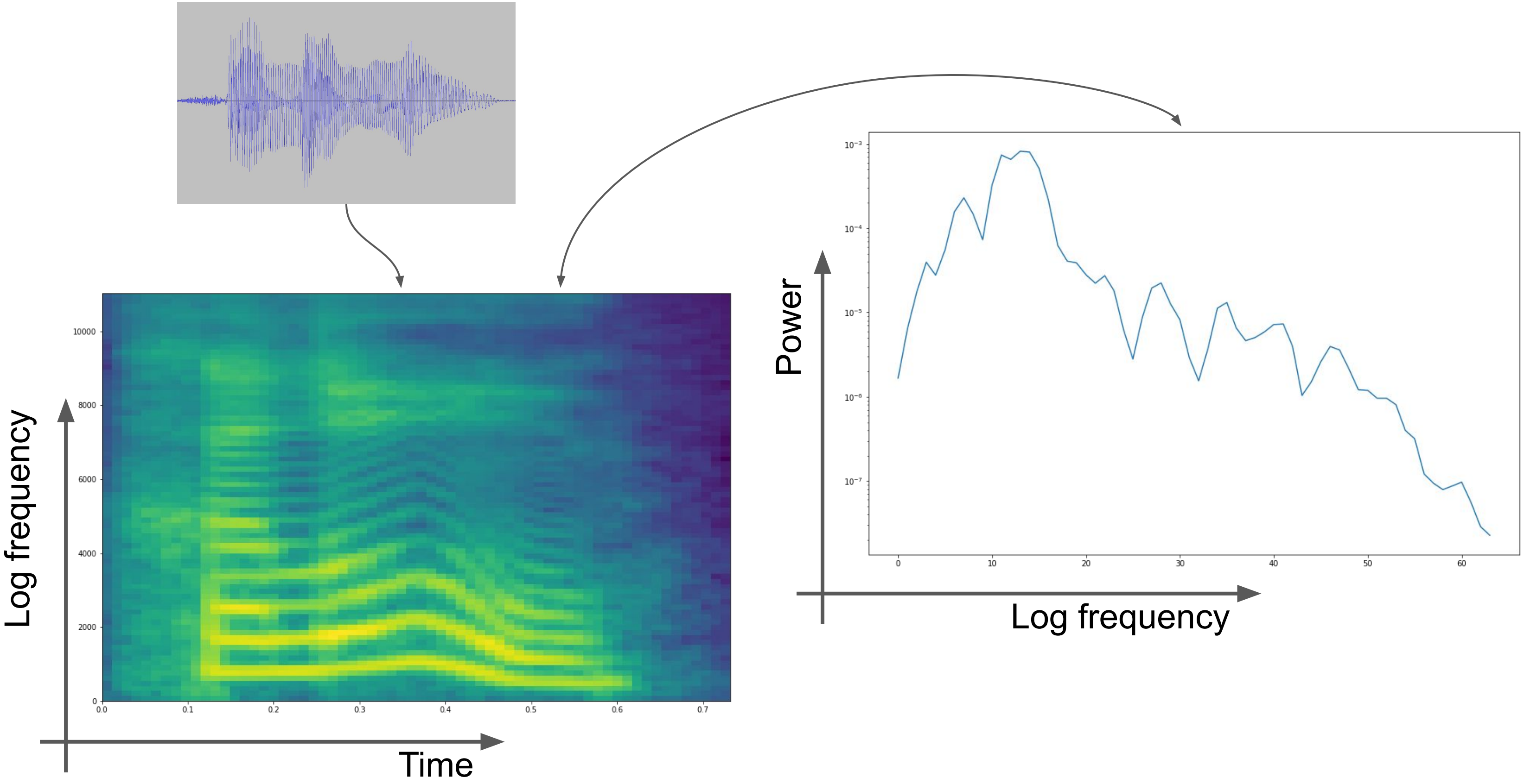
- Requires a lot of records to cover wide domain
- Produce annoying artifacts at bad unit concatenations
- Very bad at intonation variation
- Outside of records domain quality significantly decreases

Parametric synthesis

- Introduce an intermediate speech audio representation
- Generate audio in two steps, by two models

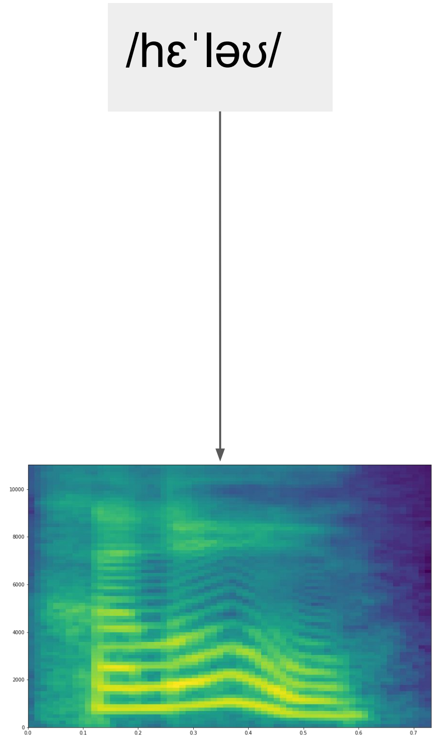


Parametric space - Mel Spectrogram



Acoustic Model

- Deals with phoneme durations
- Solves regression problem for acoustic features
- Is responsible for intonation contour
- Is responsible for phoneme articulation
- (for models with text input) Solves G2P problem



Acoustic Model

Models used for acoustic modeling:

1. Decision Trees + HMM
2. Deep Mixture Density Networks + HMM

3. RNN with duration prediction
   Deep Voice 2 https://arxiv.org/abs/1705.08947

4. Seq2Seq Networks
   Char2Wav https://openreview.net/pdf?id=B1VWyySKx
   Deep Voice 3 https://arxiv.org/abs/1710.07654
   Tacotron 1 https://arxiv.org/abs/1703.10135

Tacotron 2

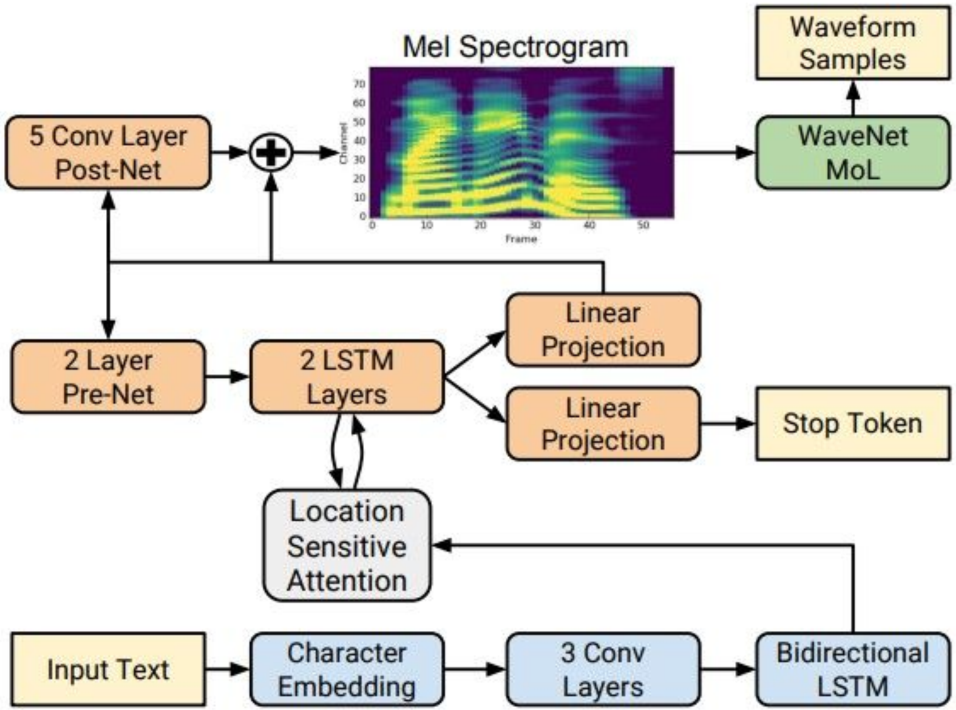


Fig. 1. Block diagram of the Tacotron 2 system architecture.

* Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions
http://arxiv.org/pdf/1712.05884.pdf

Tacotron 2 - Train Loss

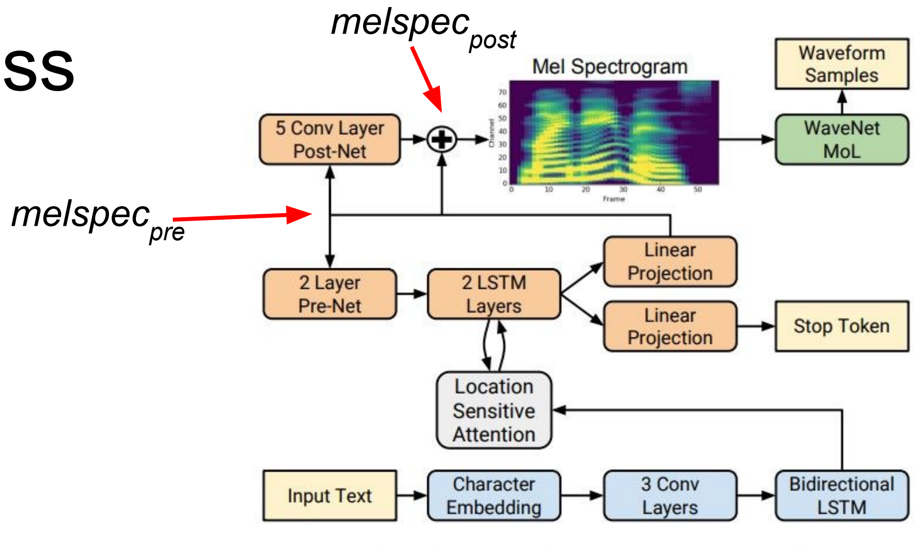


Fig. 1. Block diagram of the Tacotron 2 system architecture.

$$L_{pre} = ||melspec_{pre} - melspec_{gt}||_2^2$$

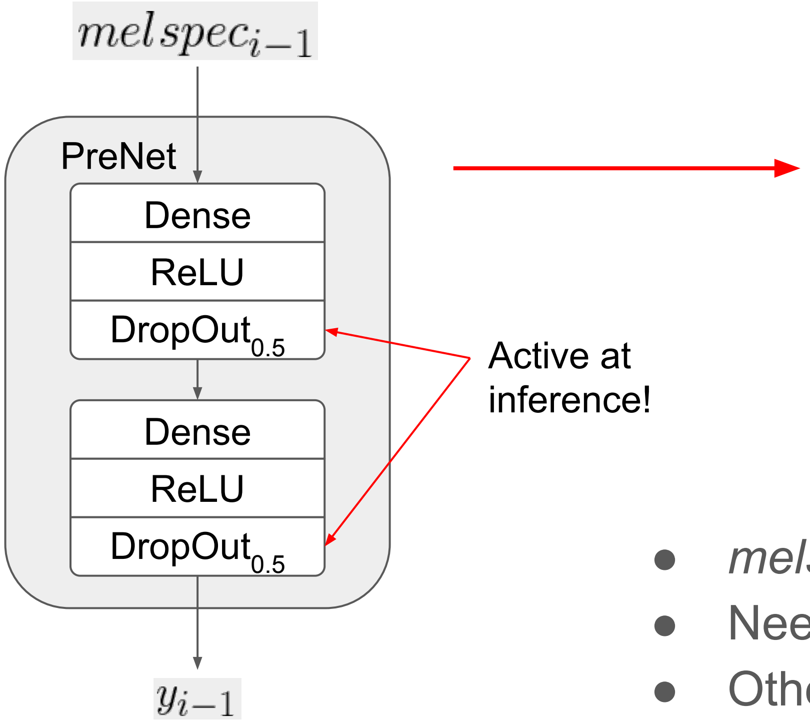
$$L_{post} = ||melspec_{post} - melspec_{gt}||_2^2$$

$$L_{stop} = \mathrm{XEnt}\left(stop\ token, \mathbb{I}_{EOS}\right)$$

$$L = L_{pre} + L_{post} + L_{stop}$$

- No discrete output - cannot add EOS token
- Separate head for EOS prediction
- Two MSE losses for MelSpec prediction

Tacotron 2 - Inference



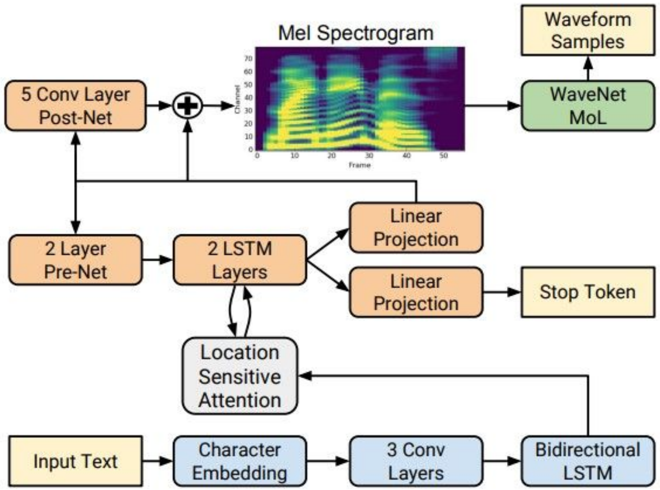
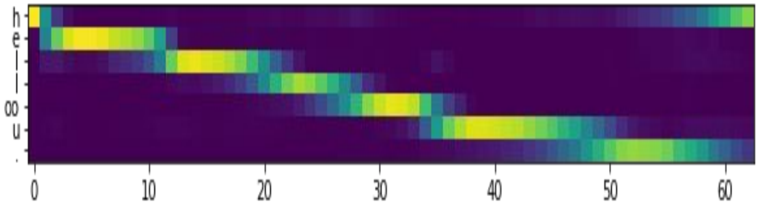


Fig. 1. Block diagram of the Tacotron 2 system architecture.

- *melspec*$_i$ is close to *melspec*$_{i-1}$
- Need to prevent data leak in autoregression
- Otherwise the network learns to copy previous value

Tacotron 2 - Attention

- Attention is (mostly) monotonic
- Depends on the previous state
- More stable than Bahdanau or Transformer attention



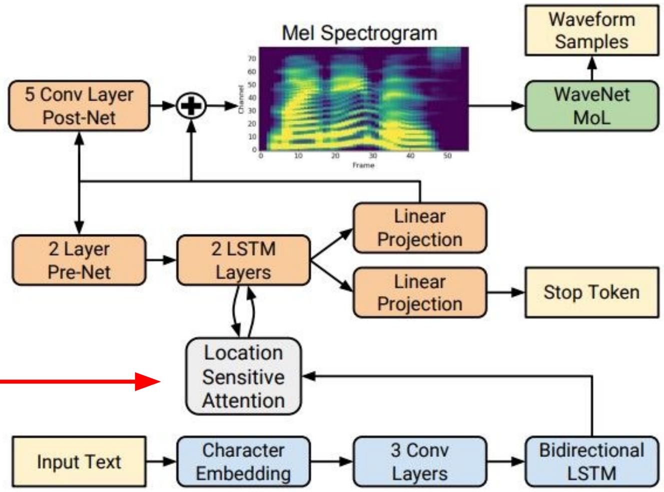


Fig. 1. Block diagram of the Tacotron 2 system architecture.

Location-sensitive Attention

$$y_{i-1} = \mathrm{PreNet}\left(melspec_{i-1}\right)$$

$$s_i = RNN_{att}\left(s_{i-1}, c_{i-1}, y_{i-1}\right)$$

$$S_i = \sum_{k=1}^{i} \alpha_k$$

$$f_i = F * [\alpha_{i-1}, S_{i-1}]$$

$$e_{ij} = v^T \tanh\left(W s_i + V h_j + U f_{ij} + b\right)$$

$$h_j = Encoder(x)_j$$

$$\alpha_i = \mathrm{SoftMax}\left(e_i\right)$$

$$c_i = \sum_j \alpha_{ij} h_j$$

$$d_i = Decoder(c_i, s_i, d_{i-1})$$

$$melspec_i = \mathrm{LinProj}(d_i)$$

[*] Attention-Based Models for Speech Recognition
https://arxiv.org/abs/1803.09017

Location-sensitive Attention

$$y_{i-1} = \mathrm{PreNet}\left(melspec_{i-1}\right)$$

$$S_i = \sum_{k=1}^{i} \alpha_k$$

$$f_i = F * [\alpha_{i-1}, S_{i-1}]$$

$$\boxed{s_i = RNN_{att}\left(s_{i-1}, c_{i-1}, y_{i-1}\right)}$$

$$e_{ij} = v^T \tanh\left(W s_i + V h_j + U f_{ij} + b\right)$$

$$h_j = Encoder(x)_j$$

$$\alpha_i = \mathrm{SoftMax}\left(e_i\right)$$

$$c_i = \sum_j \alpha_{ij} h_j$$

$$d_i = Decoder(c_i, s_i, d_{i-1})$$

$$melspec_i = \mathrm{LinProj}(d_i)$$

* Attention-Based Models for Speech Recognition
https://arxiv.org/abs/1803.09017

Location-sensitive Attention

$$y_{i-1} = \mathrm{PreNet}\left(melspec_{i-1}\right)$$

$$s_i = RNN_{att}\left(s_{i-1}, c_{i-1}, y_{i-1}\right)$$

$$\boxed{\begin{aligned} S_i &= \sum_{k=1}^{i} \alpha_k \\ f_i &= F * [\alpha_{i-1}, S_{i-1}] \end{aligned}}$$

$$\boxed{e_{ij} = v^T \tanh\left(W s_i + V h_j + U f_{ij} + b\right)}$$

$$h_j = Encoder(x)_j$$

$$\alpha_i = \mathrm{SoftMax}\left(e_i\right)$$

$$c_i = \sum_j \alpha_{ij} h_j$$

$$d_i = Decoder(c_i, s_i, d_{i-1})$$

$$melspec_i = \mathrm{LinProj}(d_i)$$

* Attention-Based Models for Speech Recognition
https://arxiv.org/abs/1803.09017

What’s Next? - Attention

More robust:

- Guided Attention https://arxiv.org/abs/1710.08969
  Stitch attention to record alignment (e.g. with ASR)

- Monotonic Attention https://arxiv.org/abs/1906.00672
  Enforce attention to read all input tokens sequentially

- Location-Relative Attention https://arxiv.org/abs/1910.10288
  Dynamic convolutions in attention make it more robust at very long utterances

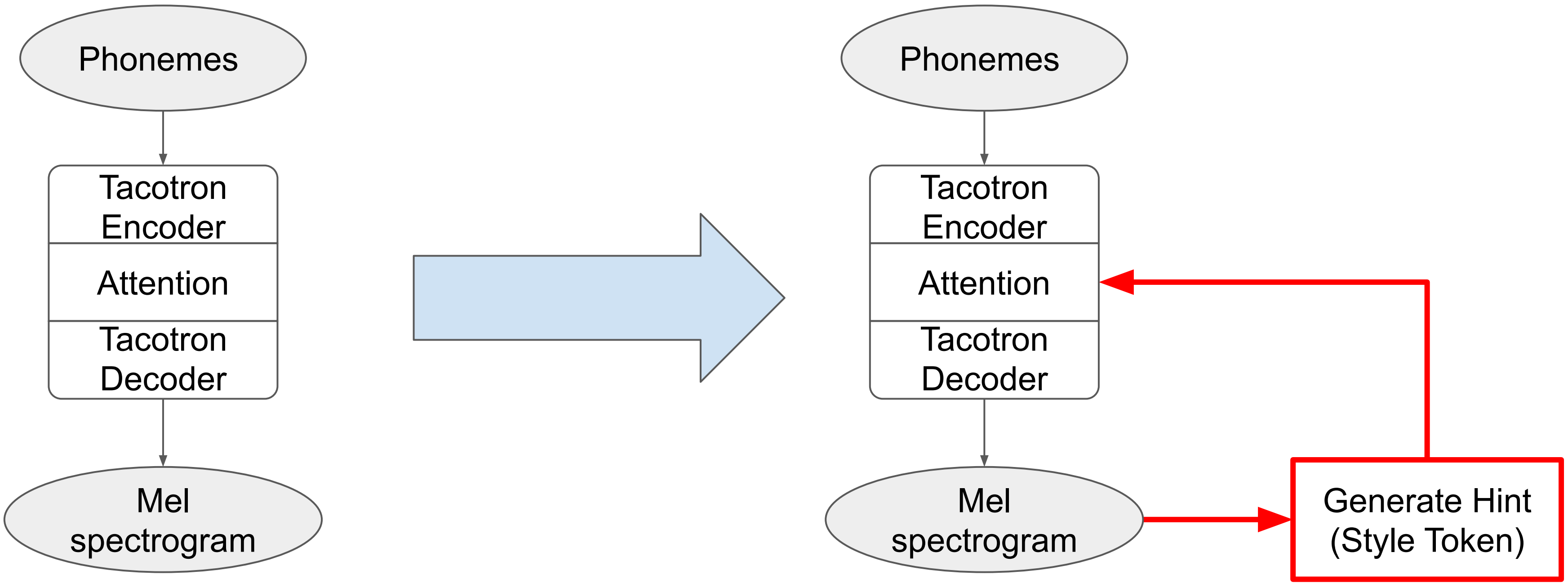
What’s Next? - Attention

Faster:

- Transformer https://arxiv.org/abs/1809.08895
  Fast convergence
  Issues with monotonicity

- FastSpeech https://arxiv.org/abs/1905.09263
  Transformers with duration prediction (use pre-trained Transformer TTS)
  No autoregression - very fast inference
  Issues with quality

What's Next? - Style Tokens



* Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis
https://arxiv.org/abs/1803.09017

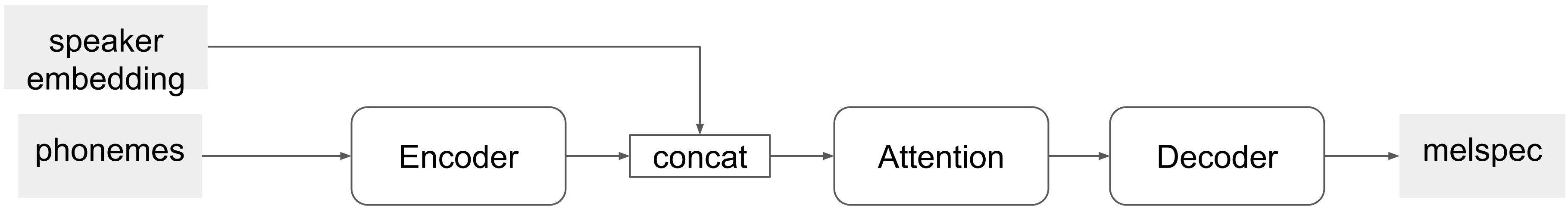
What's Next? - Style Tokens

- Different mechanisms of style modeling:
    - Several tokens with attention
      https://arxiv.org/abs/1803.09017
    - Variational Autoencoders
      https://arxiv.org/abs/1804.02135

- Predicting style:
    - From text encoder
      https://arxiv.org/abs/1808.01410
    - Use additional interpretable features
      https://arxiv.org/abs/1810.07217

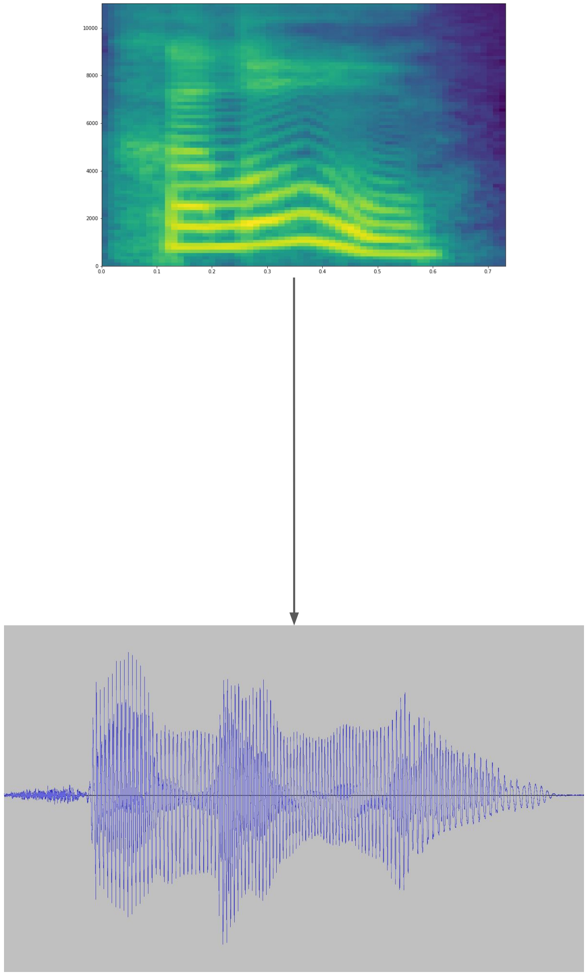
What's Next? - Multi-Speaker

- Multiple speaker training
    - Single model trained with multiple speakers/languages
    - More stable training
    - Transfer language/accents
    - Multilingual Multispeaker Tacotron https://arxiv.org/abs/1907.04448
- Speaker few-shot learning
    - Create new speaker for 1-2 records
    - Fast but unstable
    - Transfer Learning from Speaker Verification to Multispeaker https://arxiv.org/abs/1806.04558



Vocoder

- Reconstructs audio from para-space
- Sets loudness, PCM sampling rate and precision
- Responsible for low-level sound quality



Vocoder

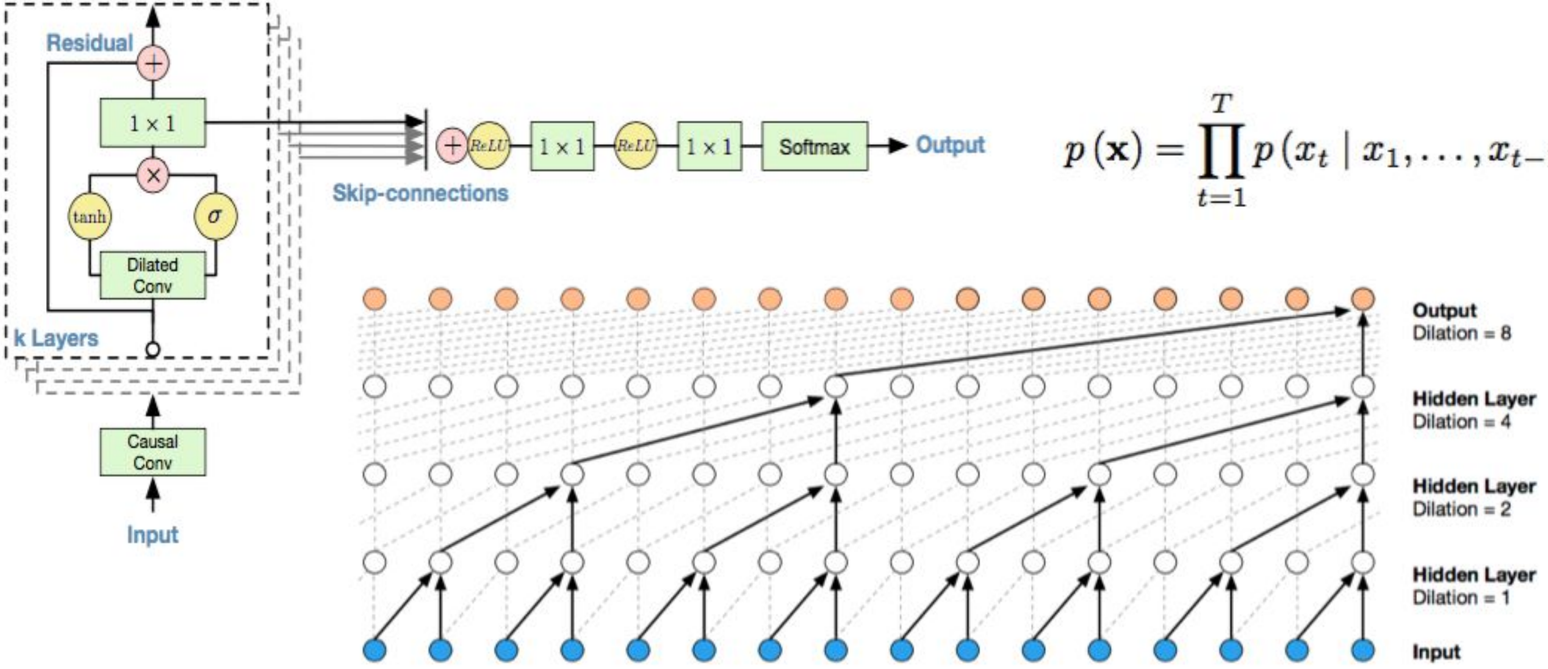
- DSP Vocoders
    - Faster, smaller
    - Worse quality
    - Usually works with more complex para-space

  World, Straight, Griffin-Lim

- Neural Vocoders
    - Requires much more computing power
    - Better quality (SotA)
    - Usually works with mel spectrograms

  WaveNet, LPCNet, ClariNet, WaveRNN, WaveGlow, MelGAN

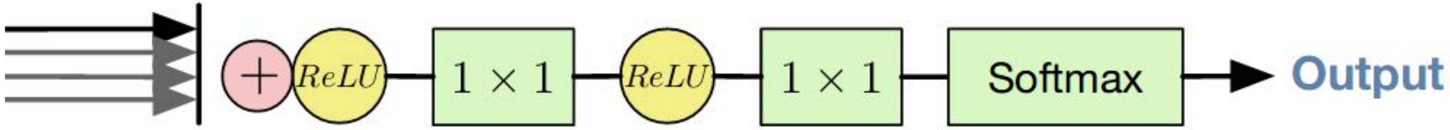
WaveNet



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

* WaveNet: A Generative Model for Raw Audio
https://arxiv.org/abs/1609.03499

WaveNet - Output



- 8bit audio => classification in 2^8 classes - cannot work with 16bit
- To upsample μ-law transformation was used

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

- Later* SML was replaced with a mixture of logistians:

$$\sum_i \pi_i \frac{e^{-(x-\mu_i)/s_i}}{s_i\left(1+e^{-(x-\mu_i)/s_i}\right)^2}$$

* Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications
https://arxiv.org/abs/1701.05517

WaveNet

Pros:

- Best audio quality for para-synthesis
- Works with continuous melspecs (good for MSE-loss Acoustic Model)

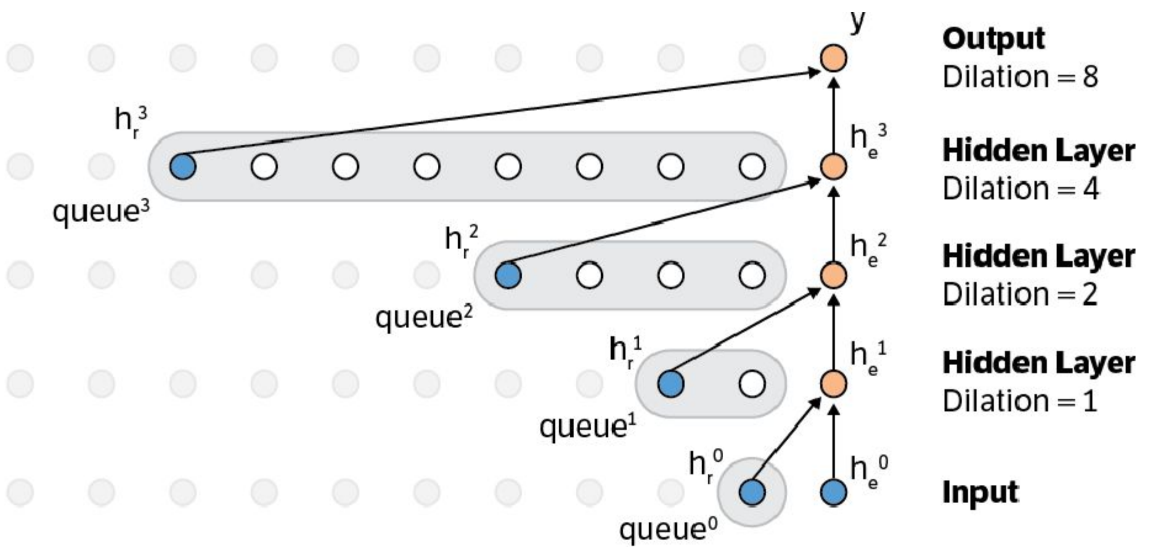
Cons:

- **200** times slower than realtime

Faster? Cache!

- No need to compute intermediate values twice

- WaveNet with caching is able to run in realtime



* Fast Wavenet Generation Algorithm
https://arxiv.org/abs/1611.09482

WaveNet with Caching

Pros:

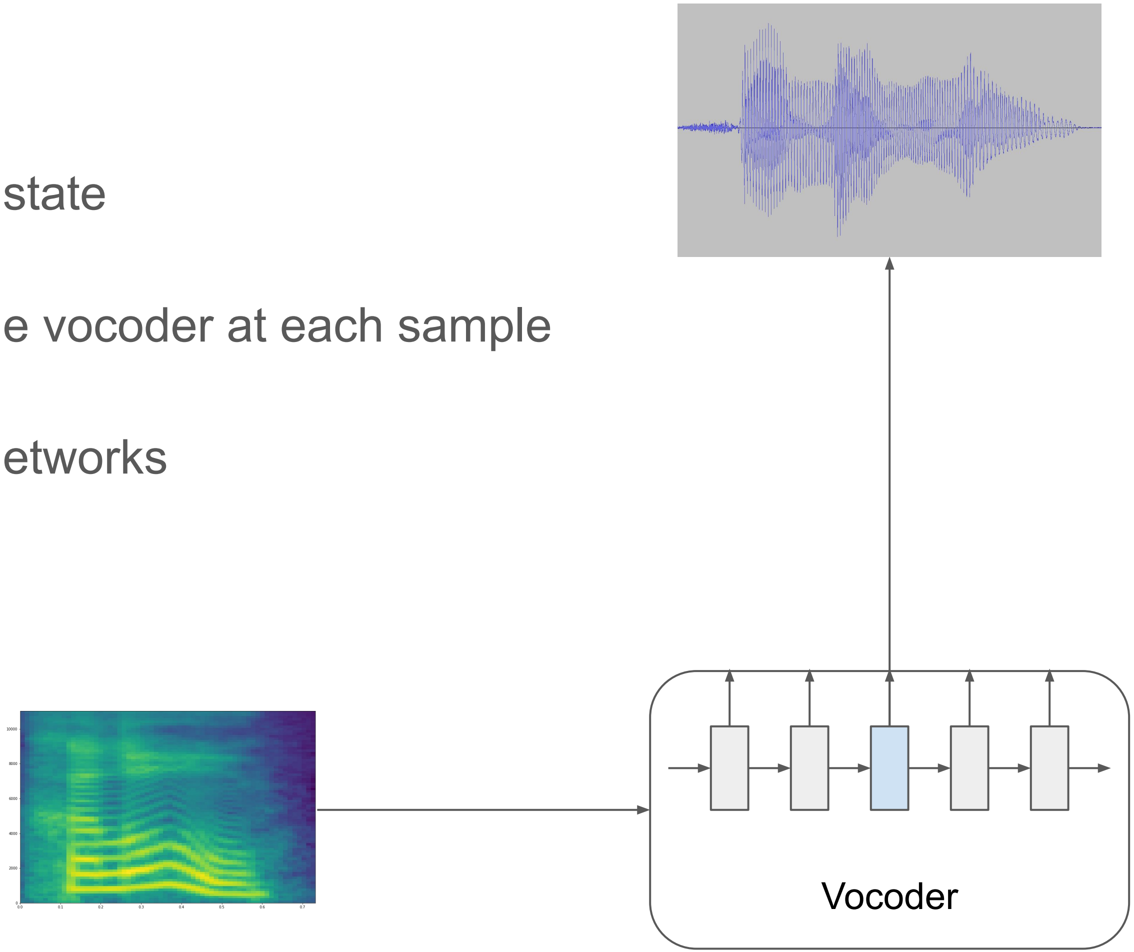
- Realtime at 2-4 CPU cores
- Vanilla WaveNet quality

Cons:

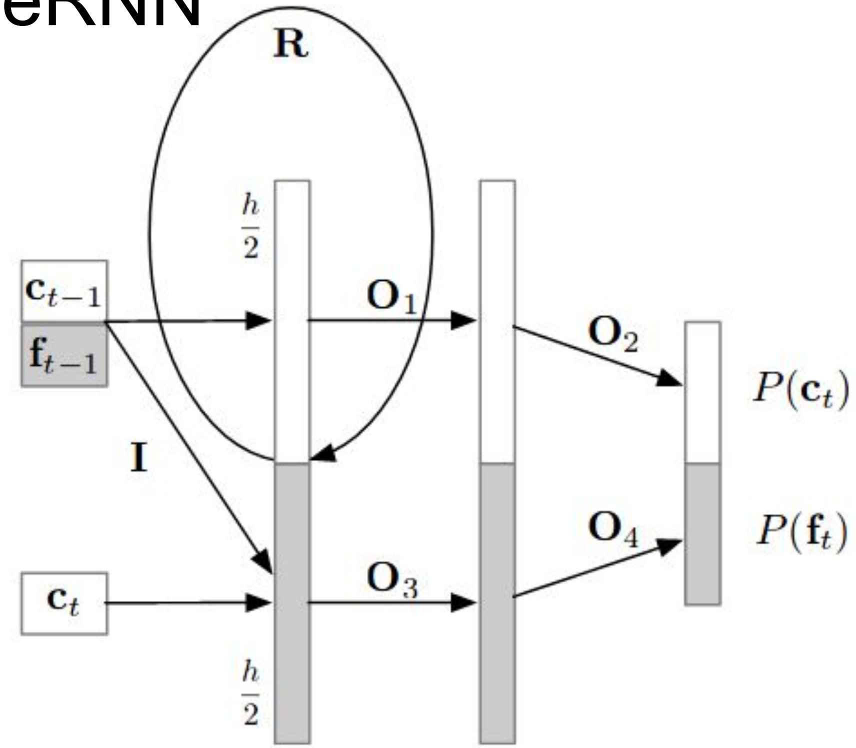
- Very hard to implement efficiently (assembler, special CPU instructions)
- Almost impossible to implement efficiently at GPU

Faster? RNN!

- RNNs can keep the state
- No need to re-run the vocoder at each sample
- Smaller and faster networks



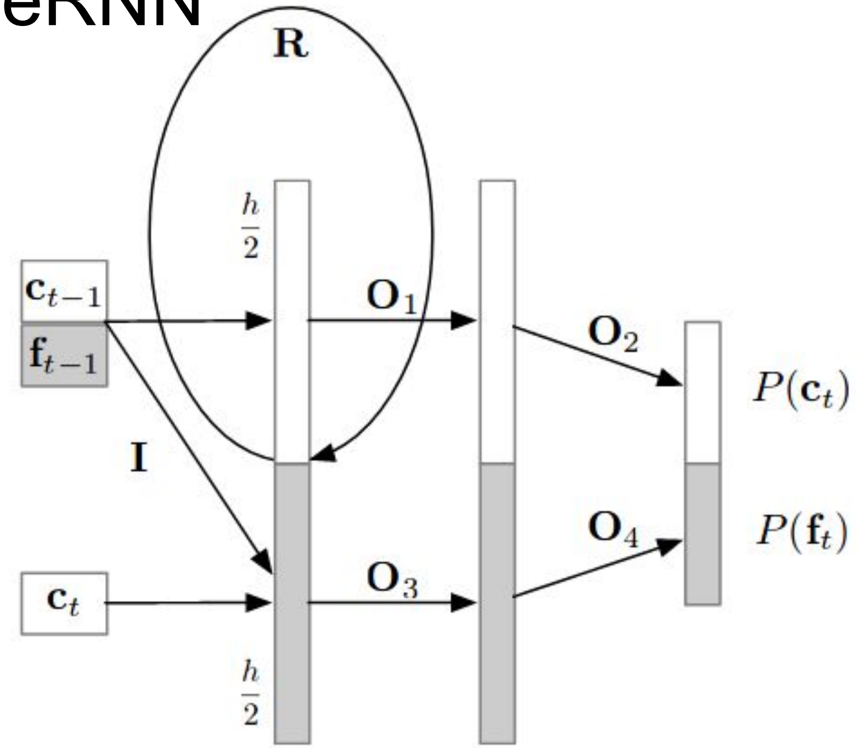
WaveRNN



$$
\begin{aligned}
\mathbf{x}_t &= [\mathbf{c}_{t-1}, \mathbf{f}_{t-1}, \mathbf{c}_t] \\
\mathbf{u}_t &= \sigma(\mathbf{R}_u \mathbf{h}_{t-1} + \mathbf{I}^{\star}_u \mathbf{x}_t) \\
\mathbf{r}_t &= \sigma(\mathbf{R}_r \mathbf{h}_{t-1} + \mathbf{I}^{\star}_r \mathbf{x}_t) \\
\mathbf{e}_t &= \tau(\mathbf{r}_t \circ (\mathbf{R}_e \mathbf{h}_{t-1}) + \mathbf{I}^{\star}_e \mathbf{x}_t) \\
\mathbf{h}_t &= \mathbf{u}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \circ \mathbf{e}_t \\
\mathbf{y}_c, \mathbf{y}_f &= \text{split}(\mathbf{h}_t) \\
P(\mathbf{c}_t) &= \text{softmax}(\mathbf{O}_2\ \text{relu}(\mathbf{O}_1 \mathbf{y}_c)) \\
P(\mathbf{f}_t) &= \text{softmax}(\mathbf{O}_4\ \text{relu}(\mathbf{O}_3 \mathbf{y}_f))
\end{aligned}
$$

* Efficient Neural Audio Synthesis
https://arxiv.org/abs/1802.08435v1

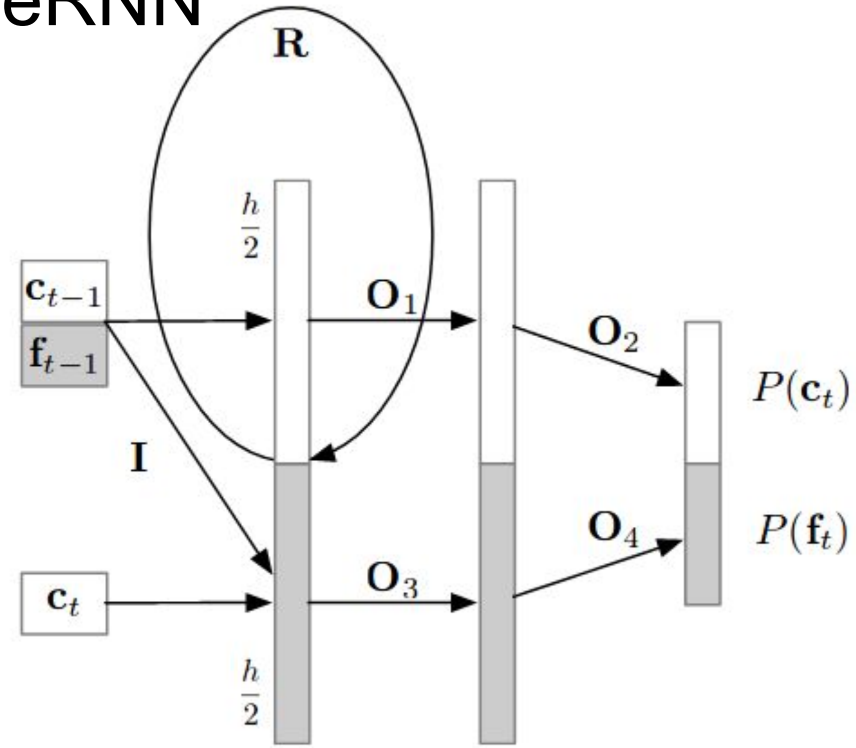
WaveRNN



$$
\begin{aligned}
\mathbf{x}_t &= [\mathbf{c}_{t-1}, \mathbf{f}_{t-1}, \mathbf{c}_t] \\
\text{GRU(x,h)} \sim \left\{ \begin{aligned}
\mathbf{u}_t &= \sigma(\mathbf{R}_u \mathbf{h}_{t-1} + \mathbf{I}^{\star}_u \mathbf{x}_t) \\
\mathbf{r}_t &= \sigma(\mathbf{R}_r \mathbf{h}_{t-1} + \mathbf{I}^{\star}_r \mathbf{x}_t) \\
\mathbf{e}_t &= \tau(\mathbf{r}_t \circ (\mathbf{R}_e \mathbf{h}_{t-1}) + \mathbf{I}^{\star}_e \mathbf{x}_t) \\
\mathbf{h}_t &= \mathbf{u}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \circ \mathbf{e}_t
\end{aligned} \right. \\
\mathbf{y}_c, \mathbf{y}_f &= \text{split}(\mathbf{h}_t) \\
P(\mathbf{c}_t) &= \text{softmax}(\mathbf{O}_2 \, \text{relu}(\mathbf{O}_1 \mathbf{y}_c)) \\
P(\mathbf{f}_t) &= \text{softmax}(\mathbf{O}_4 \, \text{relu}(\mathbf{O}_3 \mathbf{y}_f))
\end{aligned}
$$

* Efficient Neural Audio Synthesis
https://arxiv.org/abs/1802.08435v1

WaveRNN



$$\mathbf{x}_t = [\mathbf{c}_{t-1}, \mathbf{f}_{t-1}, \mathbf{c}_t]$$

$$\text{GRU(x,h)} \sim \begin{cases} \mathbf{u}_t = \sigma(\mathbf{R}_u \mathbf{h}_{t-1} + \mathbf{I}^\star_u \mathbf{x}_t) \\ \mathbf{r}_t = \sigma(\mathbf{R}_r \mathbf{h}_{t-1} + \mathbf{I}^\star_r \mathbf{x}_t) \\ \mathbf{e}_t = \tau(\mathbf{r}_t \circ (\mathbf{R}_e \mathbf{h}_{t-1}) + \mathbf{I}^\star_e \mathbf{x}_t) \\ \mathbf{h}_t = \mathbf{u}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \circ \mathbf{e}_t \end{cases}$$

$$\mathbf{y}_c, \mathbf{y}_f = \text{split}(\mathbf{h}_t)$$

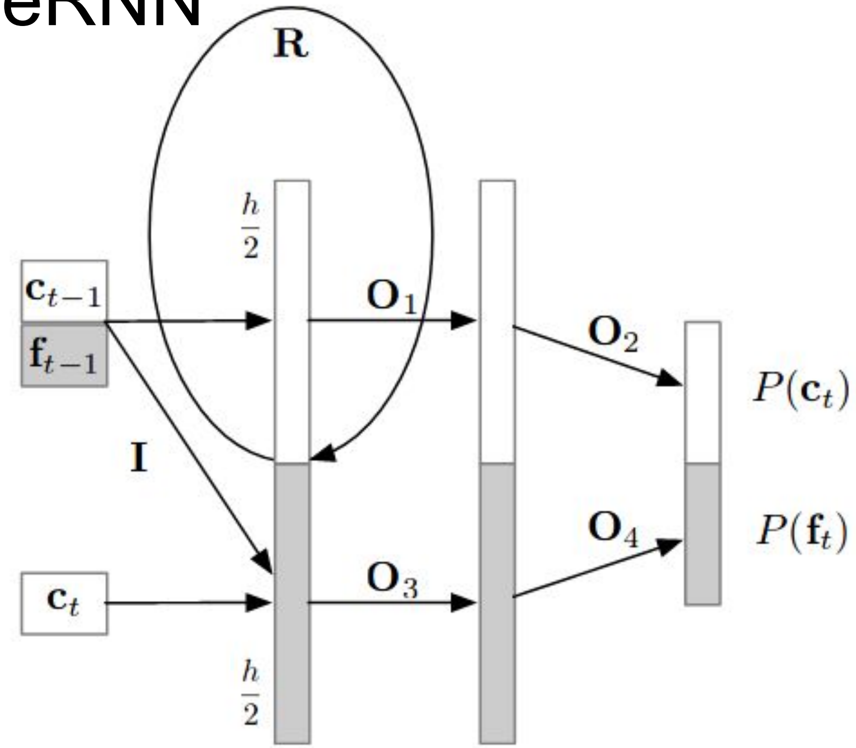
$$P(\mathbf{c}_t) = \text{softmax}(\mathbf{O}_2 \text{ relu}(\mathbf{O}_1 \mathbf{y}_c))$$

$$P(\mathbf{f}_t) = \text{softmax}(\mathbf{O}_4 \text{ relu}(\mathbf{O}_3 \mathbf{y}_f))$$

1. $(h_{t-1}, [c_{t-1}, c_{t-1}]) \rightarrow h_t^c \rightarrow y_c \rightarrow c_t$

* Efficient Neural Audio Synthesis
https://arxiv.org/abs/1802.08435v1

WaveRNN



$$\mathbf{x}_t = [\mathbf{c}_{t-1}, \mathbf{f}_{t-1}, \mathbf{c}_t]$$

$$\text{GRU(x,h)} \sim \left\{ \begin{aligned} \mathbf{u}_t &= \sigma(\mathbf{R}_u \mathbf{h}_{t-1} + \mathbf{I}_u^{\star} \mathbf{x}_t) \\ \mathbf{r}_t &= \sigma(\mathbf{R}_r \mathbf{h}_{t-1} + \mathbf{I}_r^{\star} \mathbf{x}_t) \\ \mathbf{e}_t &= \tau(\mathbf{r}_t \circ (\mathbf{R}_e \mathbf{h}_{t-1}) + \mathbf{I}_e^{\star} \mathbf{x}_t) \\ \mathbf{h}_t &= \mathbf{u}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \circ \mathbf{e}_t \end{aligned} \right.$$

$$\mathbf{y}_c, \mathbf{y}_f = \text{split}(\mathbf{h}_t)$$

$$P(\mathbf{c}_t) = \text{softmax}(\mathbf{O}_2 \,\text{relu}(\mathbf{O}_1 \mathbf{y}_c))$$

$$P(\mathbf{f}_t) = \text{softmax}(\mathbf{O}_4 \,\text{relu}(\mathbf{O}_3 \mathbf{y}_f))$$

1. $(h_{t-1}, [c_{t-1}, c_{t-1}]) \rightarrow h_t^c \rightarrow y_c \rightarrow c_t$
2. $(h_{t-1}, [c_{t-1}, f_{t-1}, c_t]) \rightarrow h_t^f \rightarrow y_f \rightarrow f_t$

* Efficient Neural Audio Synthesis
https://arxiv.org/abs/1802.08435v1

WaveRNN

Pros:

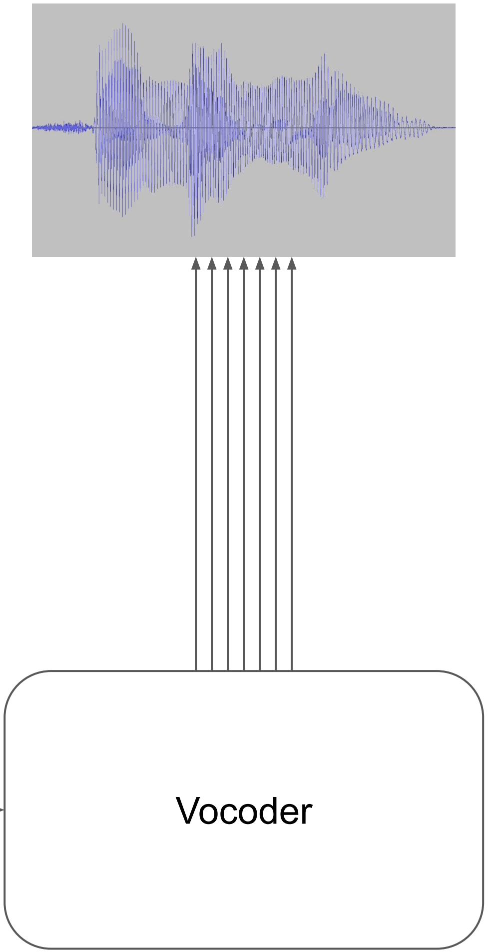
- Real-time inference
- Quality comparable with WaveNet
- Can run (compressed version) at CPU in realtime

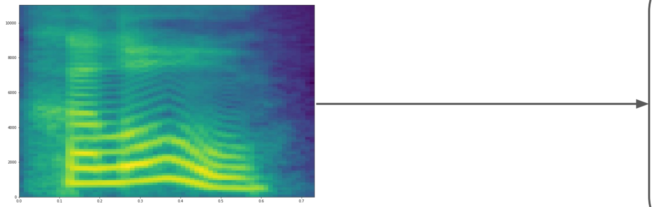
Cons:

- Hard to make GPU inference effective

Faster? Parallel!

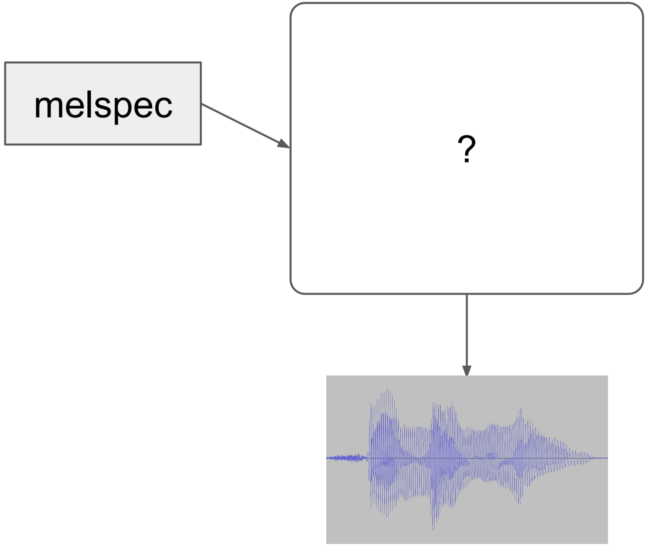
- Very efficient generation at GPU
- Need to break autoregression loop
- Vocoders:
  - Parallel WaveNet https://arxiv.org/abs/1711.10433
  - ClariNet https://arxiv.org/abs/1807.07281
  - WaveGlow https://arxiv.org/abs/1811.00002





WaveGlow

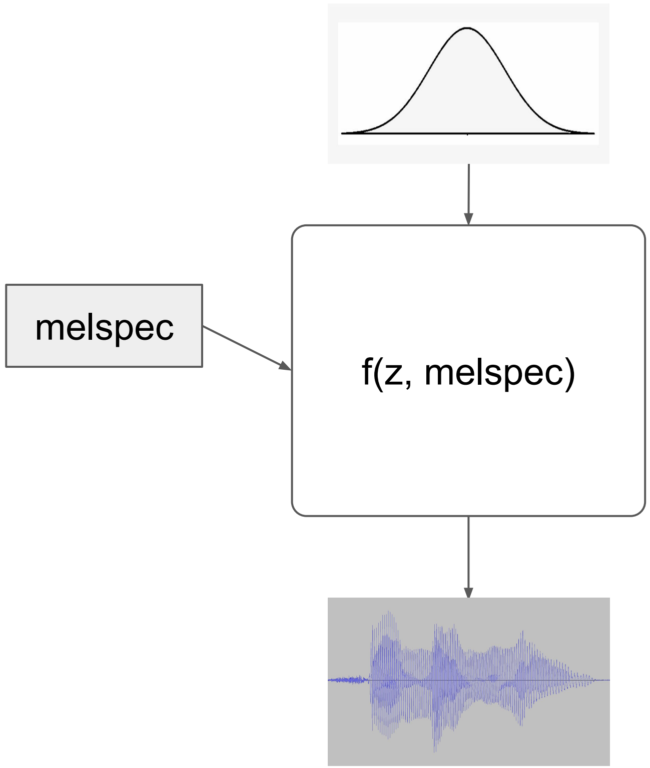
1. Need to sample: $x \sim D_{wav}(\text{melspec})$
2. $D_{wav} - ?$



[*] WaveGlow: A Flow-based Generative Network for Speech Synthesis
https://arxiv.org/abs/1811.00002

WaveGlow

1. Need to sample: $x \sim D_{wav}(\text{melspec})$
2. $D_{wav} - ?$
3. $x = f(z, \text{melspec});\ z \sim \mathcal{N}(0, 1)$

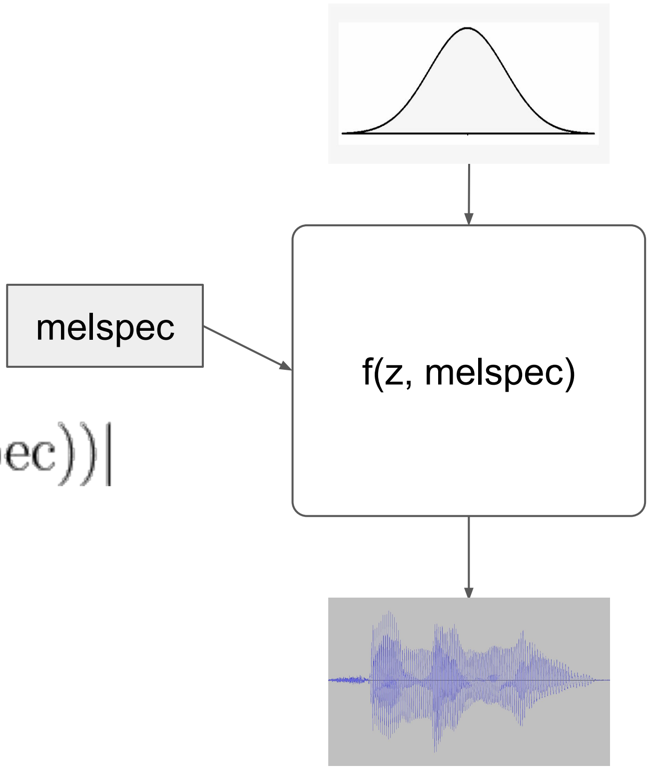


* WaveGlow: A Flow-based Generative Network for Speech Synthesis
https://arxiv.org/abs/1811.00002

WaveGlow

1. Need to sample: $x \sim D_{wav}(\text{melspec})$
2. $D_{wav} - ?$
3. $x = f(z, \text{melspec});\; z \sim \mathcal{N}(0,1)$
4. $z = f^{-1}(x)$

$$p(x|\text{melspec}) = p(z(x))|\det J(f^{-1}(x, \text{melspec}))|$$



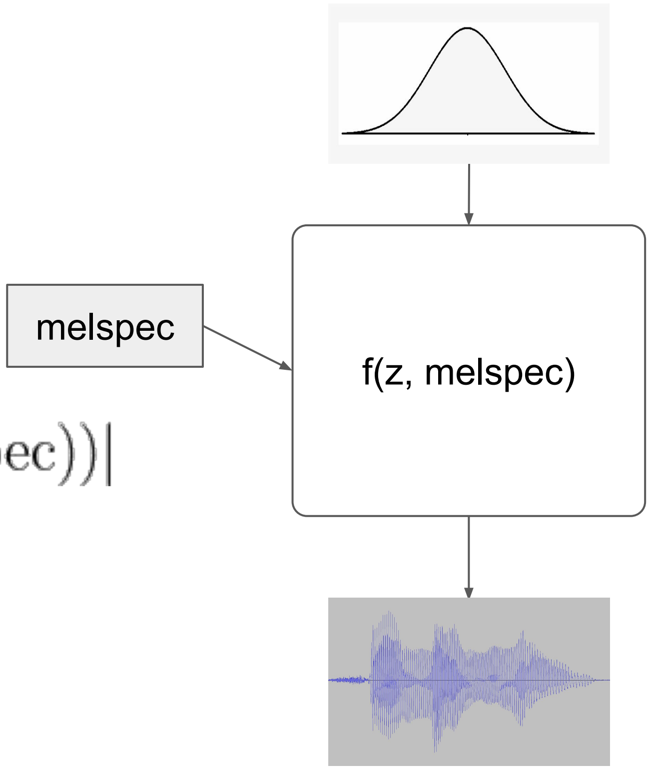
* WaveGlow: A Flow-based Generative Network for Speech Synthesis
https://arxiv.org/abs/1811.00002

WaveGlow

1. Need to sample: $x \sim D_{wav}(\text{melspec})$
2. D_{wav} – ?
3. $x = f(z, \text{melspec});\ z \sim \mathcal{N}(0, 1)$
4. $z = f^{-1}(x)$

$$p(x|\text{melspec}) = p(z(x))|\det J(f^{-1}(x, \text{melspec}))|$$

5. Train to maximize $p_\theta(x|\text{melspec})$



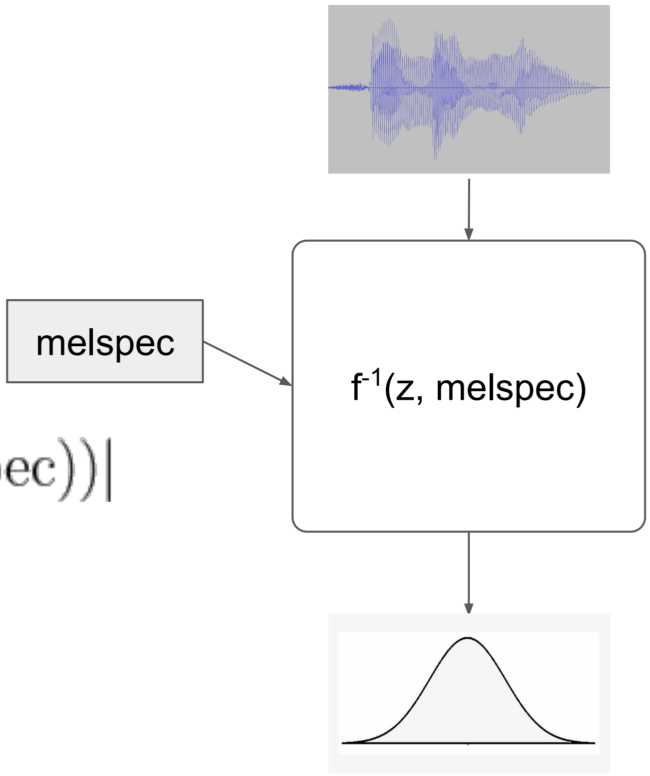
* WaveGlow: A Flow-based Generative Network for Speech Synthesis
https://arxiv.org/abs/1811.00002

WaveGlow

1. Need to sample: $x \sim D_{wav}(\text{melspec})$
2. D_{wav} – ?
3. $x = f(z, \text{melspec});\ z \sim \mathcal{N}(0, 1)$
4. $z = f^{-1}(x)$

$$p(x|\text{melspec}) = p(z(x))|\det J(f^{-1}(x, \text{melspec}))|$$

5. Train to maximize $p_\theta(x|\text{melspec})$
6. Train invertible function $f^{-1}(x, \text{melspec})$



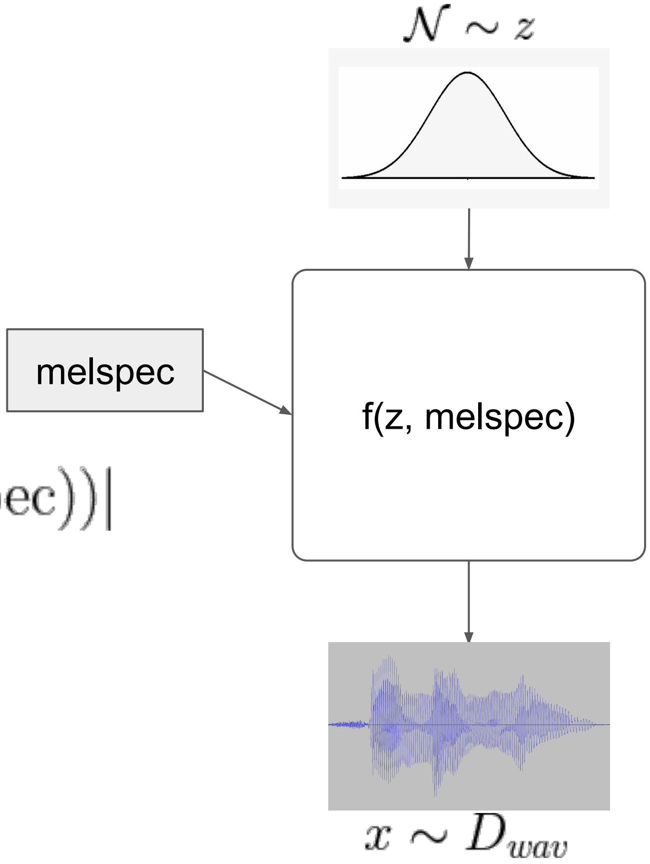
* WaveGlow: A Flow-based Generative Network for Speech Synthesis
https://arxiv.org/abs/1811.00002

WaveGlow

1. Need to sample: $x \sim D_{wav}(\text{melspec})$
2. D_{wav} – ?
3. $x = f(z, \text{melspec});\ z \sim \mathcal{N}(0, 1)$
4. $z = f^{-1}(x)$

$$p(x|\text{melspec}) = p(z(x))|\det J(f^{-1}(x, \text{melspec}))|$$

5. Train to maximize $p_\theta(x|\text{melspec})$
6. Train invertible function $f^{-1}(x, \text{melspec})$
7. Revert and sample x via z



[*] WaveGlow: A Flow-based Generative Network for Speech Synthesis
https://arxiv.org/abs/1811.00002

WaveGlow - Coupling

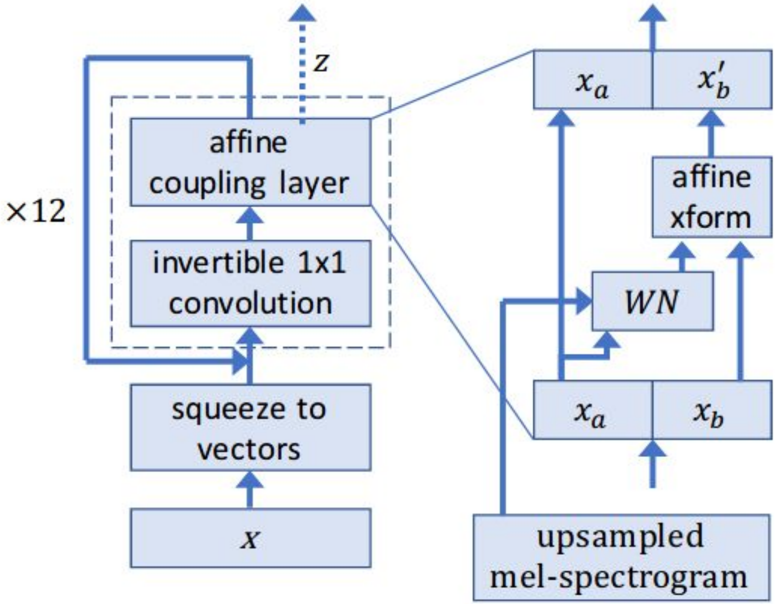


Fig. 1: WaveGlow network

$$x_a, x_b = \mathrm{split}(x)$$
$$(\log s,\ t) = WN(x_a, \mathrm{melspec})$$
$$x'_b = s \odot x_b + t$$
$$f^{-1}_{coupling}(x) = \mathrm{concat}(x_a, x'_b)$$

WaveGlow - Coupling

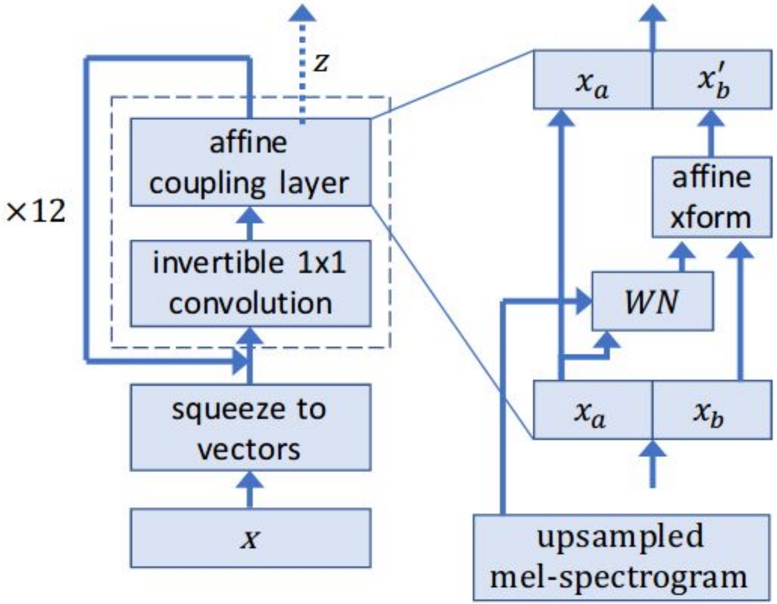


Fig. 1: WaveGlow network

$$x_a, x_b = \mathrm{split}(x)$$
$$(\log s,\ t) = WN(x_a, \mathrm{melspec})$$
$$x'_b = s \odot x_b + t$$
$$f^{-1}_{coupling}(x) = \mathrm{concat}(x_a, x'_b)$$

$$x'_a, x'_b = \mathrm{split}(x')$$
$$(\log s,\ t) = WN(x'_a, \mathrm{melspec})$$
$$x_b = s^{-1} \odot (x'_b - t)$$
$$f_{coupling}(x) = \mathrm{concat}(x'_a, x_b)$$

WaveGlow - Coupling

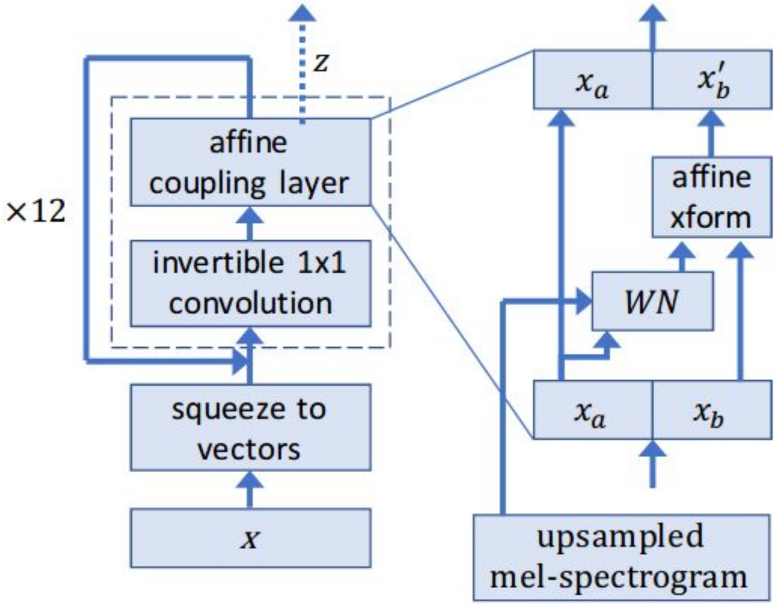


Fig. 1: WaveGlow network

$$x_a, x_b = \mathrm{split}(x)$$
$$(\log s,\ t) = WN(x_a, \mathrm{melspec})$$
$$x'_b = s \odot x_b + t$$
$$f^{-1}_{coupling}(x) = \mathrm{concat}(x_a, x'_b)$$

$$x'_a, x'_b = \mathrm{split}(x')$$
$$(\log s,\ t) = WN(x'_a, \mathrm{melspec})$$
$$x_b = s^{-1} \odot (x'_b - t)$$
$$f_{coupling}(x) = \mathrm{concat}(x'_a, x_b)$$

$$p_\theta(x|\mathrm{melspec}) = exp\left(-\frac{z(x, \mathrm{melspec})^2}{2\sigma^2}\right) \prod_{j=0}^{\#coupling} s_j(x, \mathrm{melspec}) \prod_{k=0}^{\#conv} |\det W_k|$$

WaveGlow

Pros:

- Quality comparable to vanilla WaveNet
- Fast (520k samples / second) inference at GPU
- Code at GitHub

Cons:

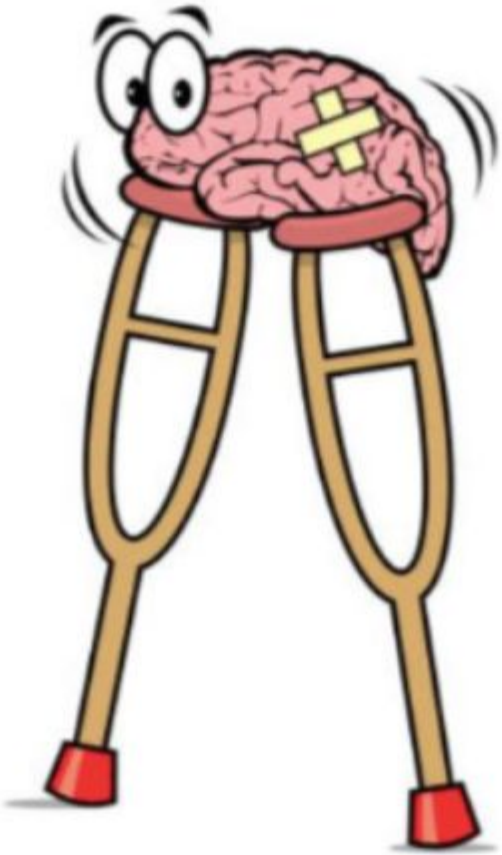
- Difficult streaming inference
- Impossible to implement on CPU

Hack of the Day

1. There is a github repo for it
   - https://github.com/NVIDIA/waveglow
   - https://github.com/NVIDIA/tacotron2
   - https://github.com/r9y9/wavenet_vocoder

2. P("there's a bug") >> P("it underfits")

3. Don't trust losses. Listen



?

Extras

Parametric space - Mel Spectrogram

PCM

↓

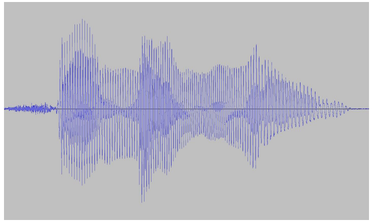
STFT

↓

Mel filterbanks

↓

Mel spectrogram



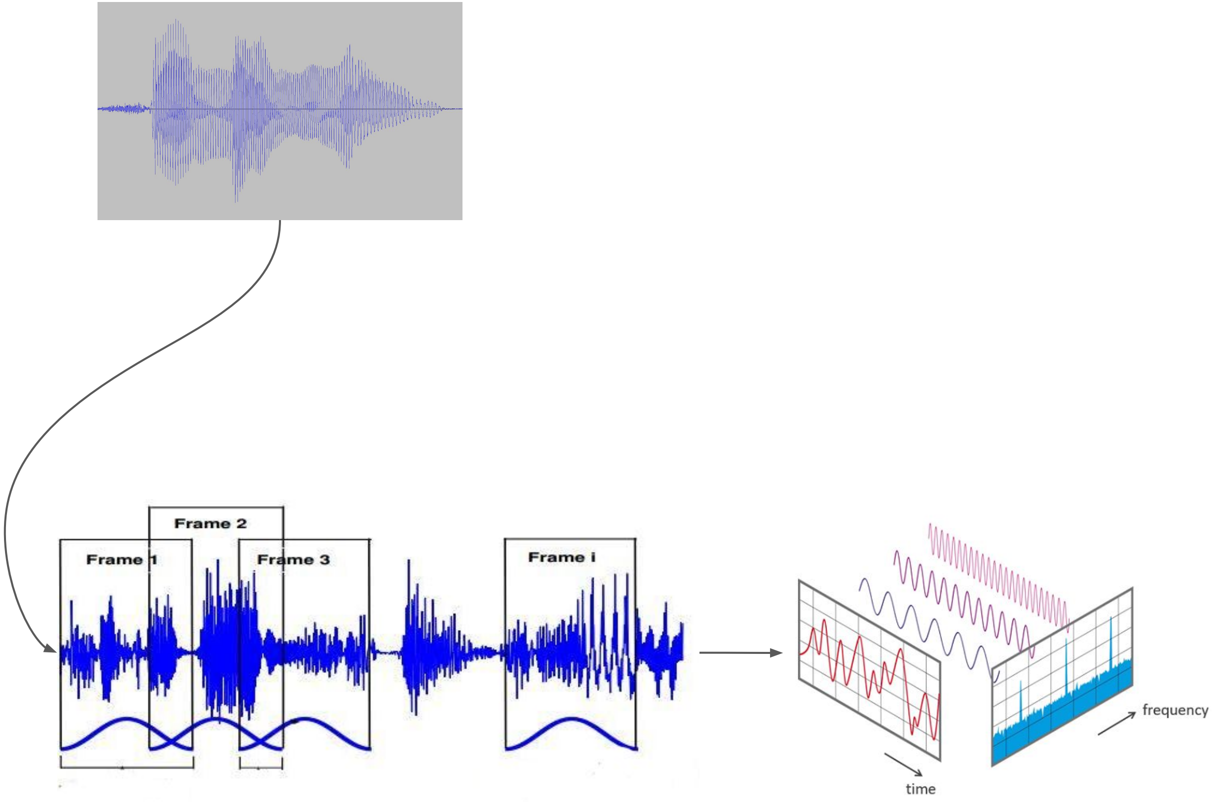
Parametric space - Mel Spectrogram

PCM

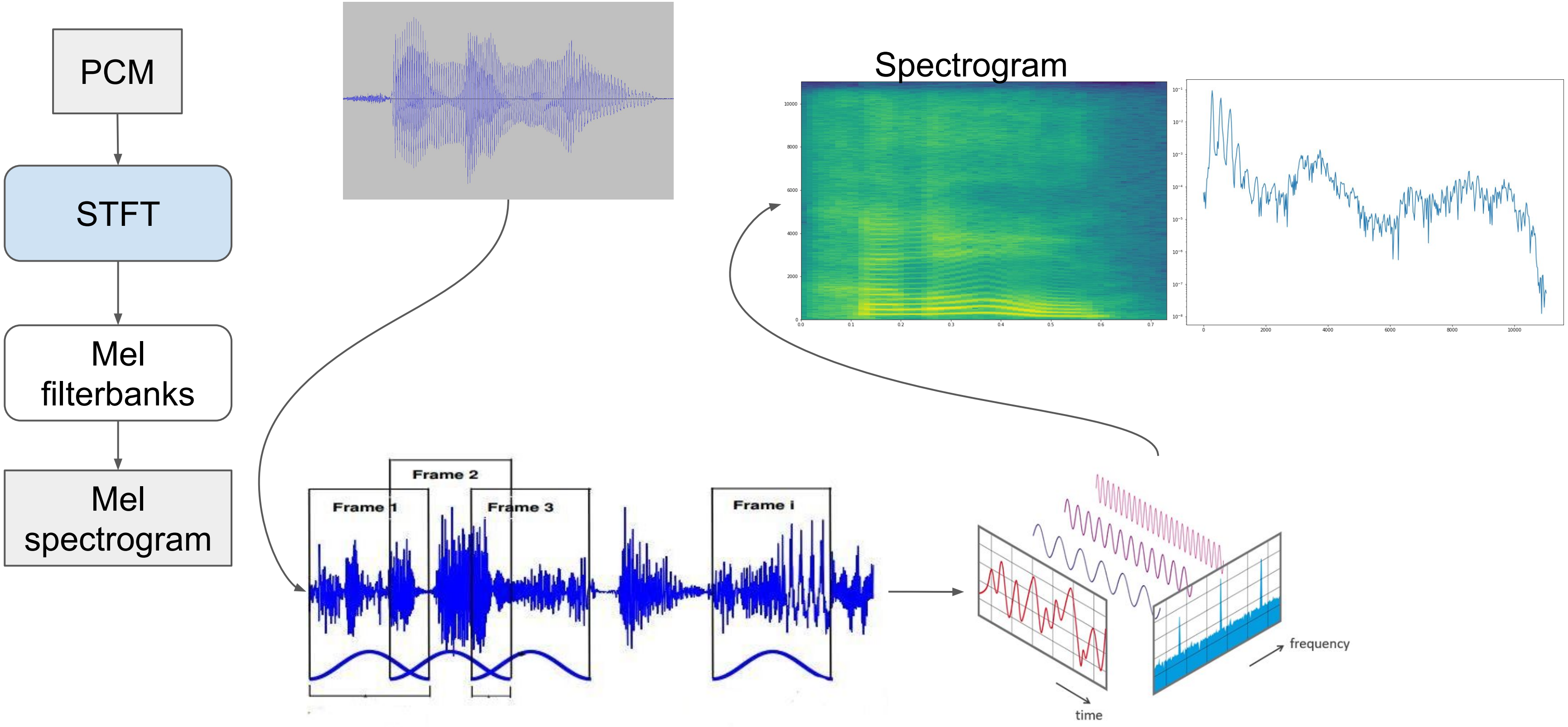
STFT

Mel filterbanks

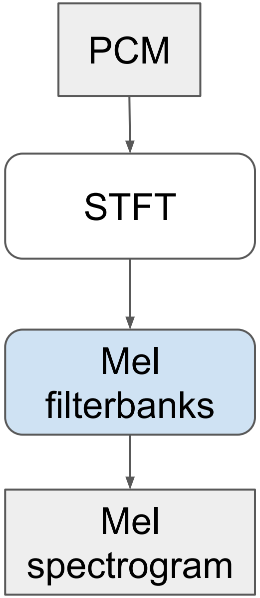
Mel spectrogram

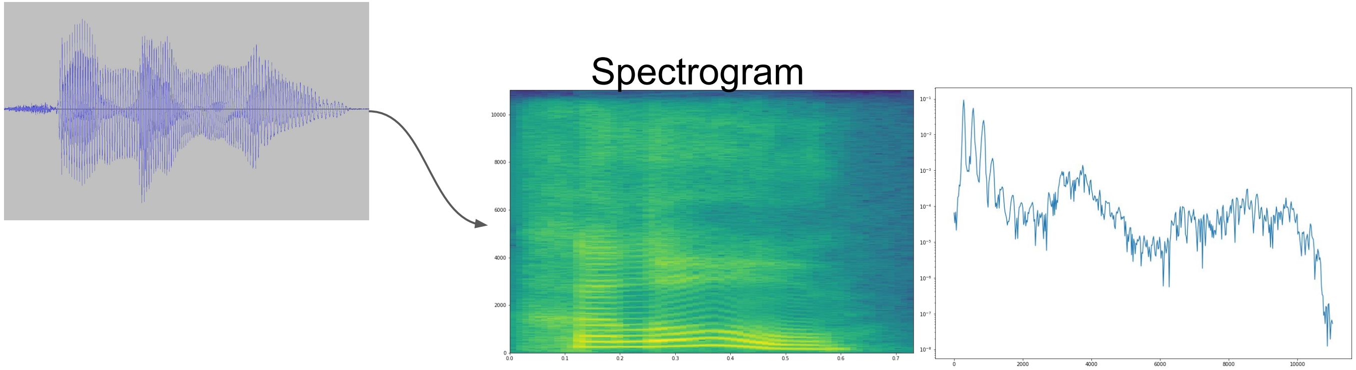


Parametric space - Mel Spectrogram

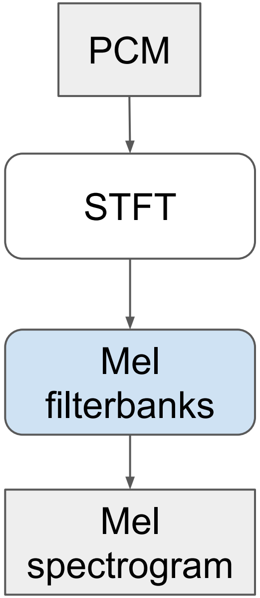


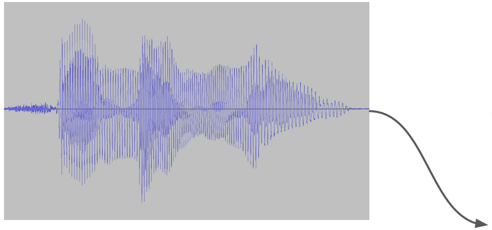
Parametric space - Mel Spectrogram



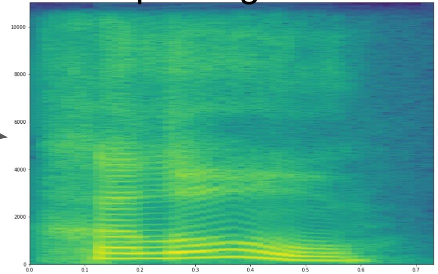


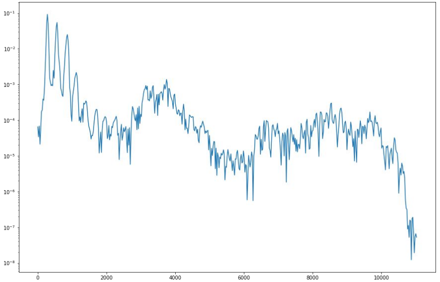
Parametric space - Mel Spectrogram



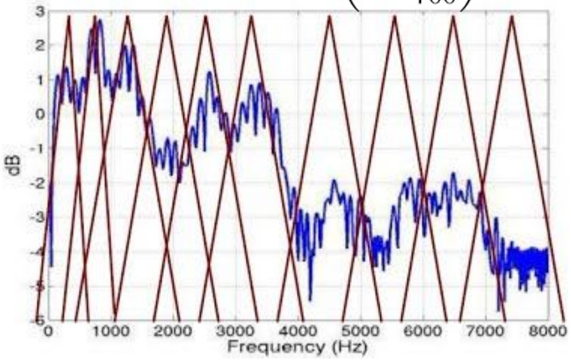


Spectrogram





$$mel = 2595\ \log_{10}\left(1 + \frac{f}{700}\right)$$



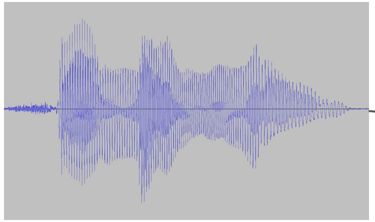
Parametric space - Mel Spectrogram

PCM

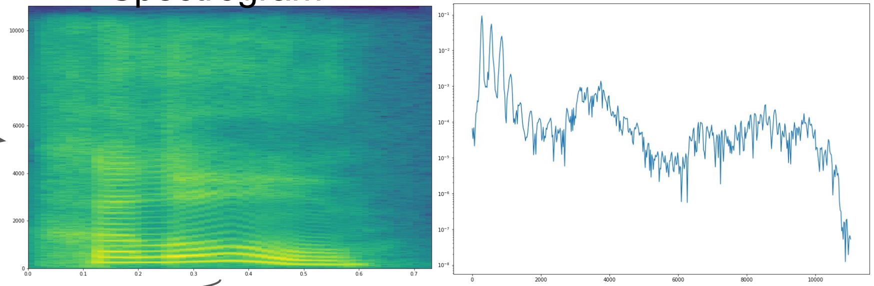
STFT

Mel filterbanks

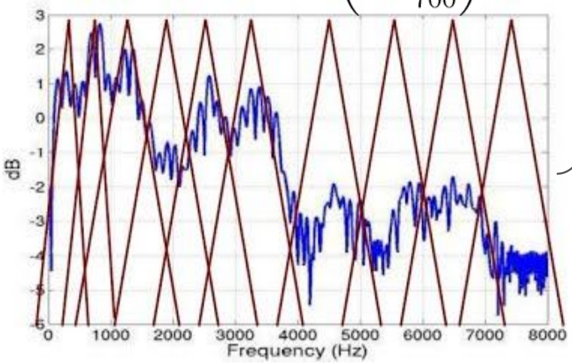
Mel spectrogram



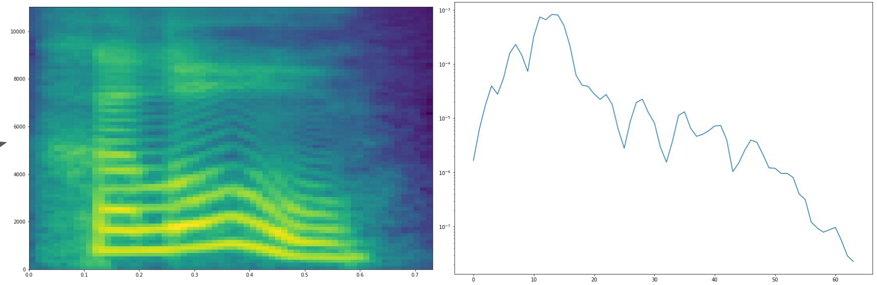
Spectrogram



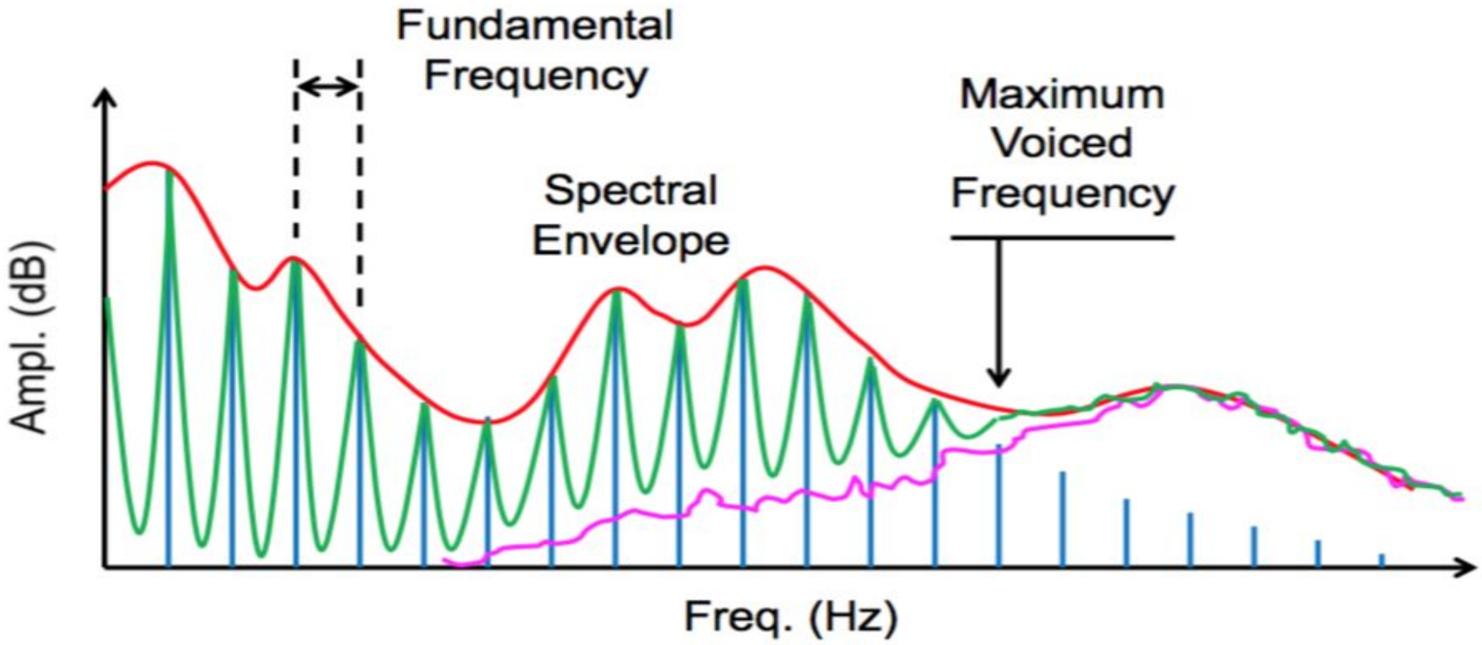
$$mel = 2595\ \log_{10}\left(1 + \frac{f}{700}\right)$$



Mel Spectrogram



Other para-spaces



Other para-spaces

We could further process MelSpec to make more complex and compact features:

- F0 - frequency at which vocal chords vibrate in voiced sound

- Spectral Envelope, could be described with:
  - Cepstral Coefficients - Discrete Cosine Transform of (mel) spectrum
  - Linear Predictive Coding - autoregressive model describing envelope

- Periodicity
  - Frequency-wise
  - Max Voiced Frequency