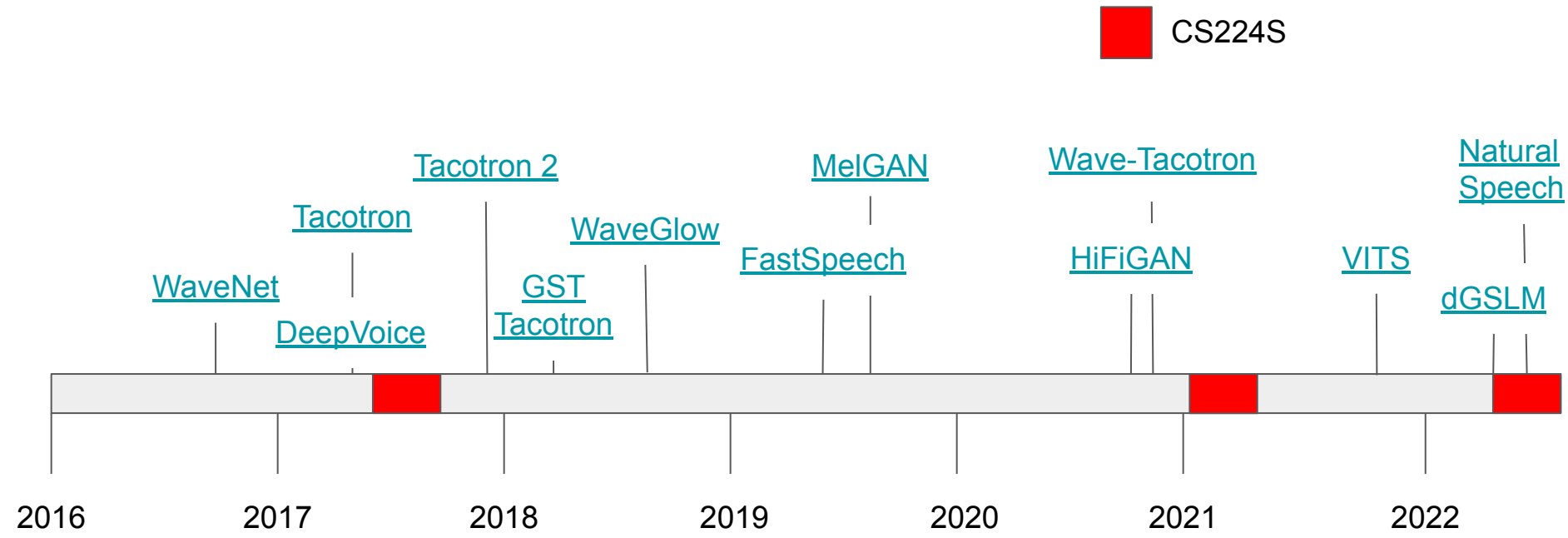
Gridspace

CS224S
Lecture 16: Deep Learning for TTS

Alex Barron
Stanford University
Spring 2022

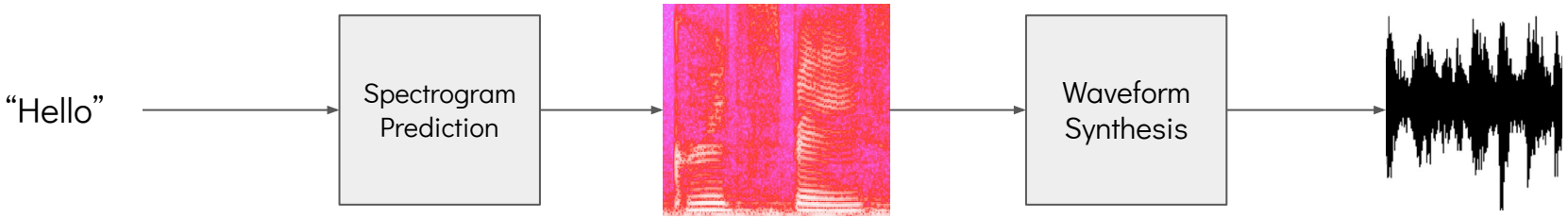
Rapid Progress in TTS over the last few years



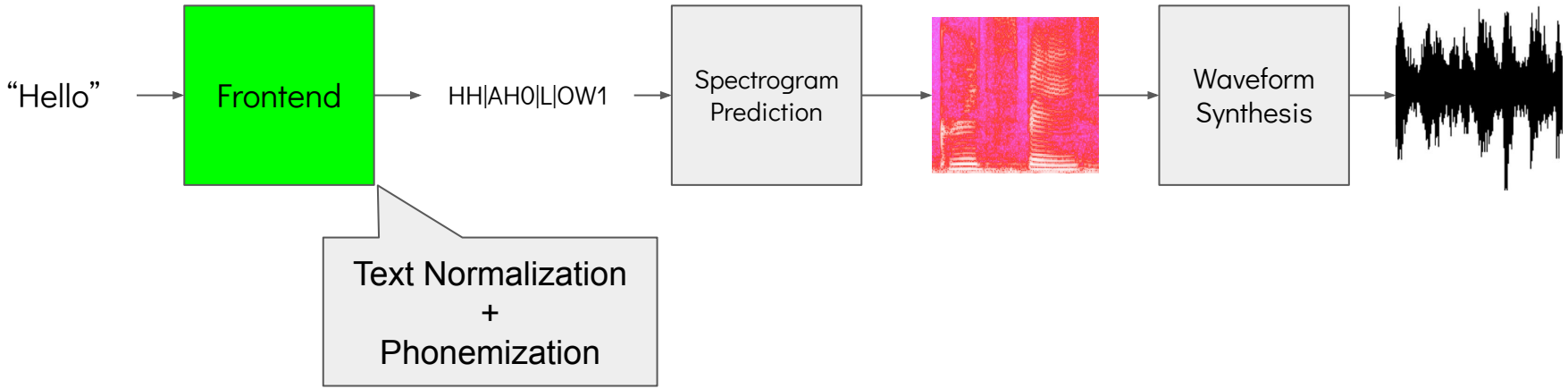
Overview

1. Text to Spectrogram Models
2. Spectrogram to Audio Models (Vocoders)
3. Speaker and Style Embeddings
4. End to End Models
5. Generative Spoken Language Modeling

Neural TTS Paradigm



Neural TTS Paradigm



Why Neural TTS?

- Upper bound on quality is higher
- Works with a wider variety of datasets
- Much more easily extended for speaker/style customization
- Many fewer components to train than traditional TTS

Why Neural TTS? (continued)

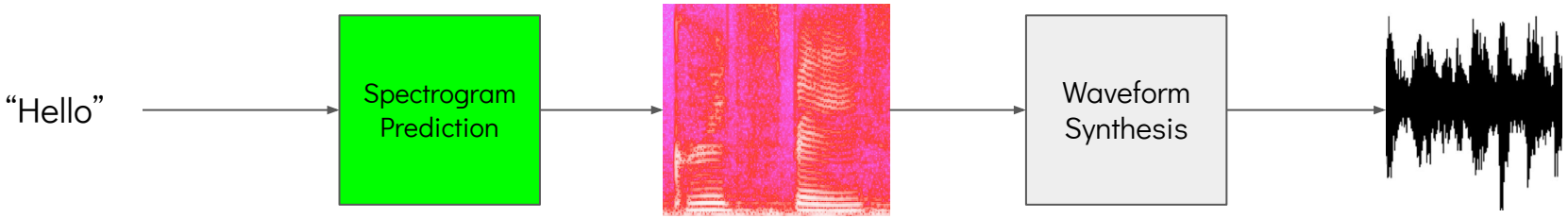
- Accessible!
- Single speaker datasets are 1-10Gb e.g. LJSpeech
- You can get decent results in a night on a solid GPU with most models

Why Use Intermediate Spectrograms?

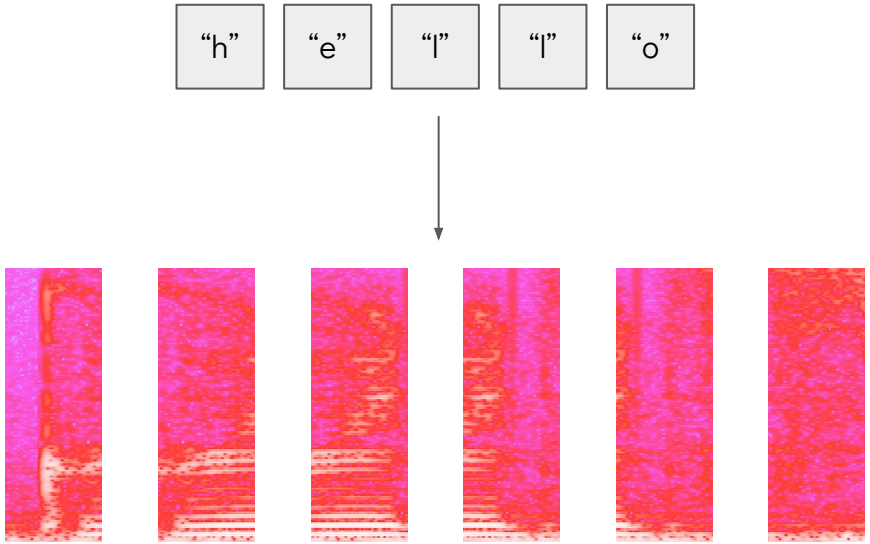
- Prosodic/phonemic aspects of speech can be modelled without phase information
- Allows focus on human speech frequency bands with mel filters
- STFT chunks speech into frames of a useful duration for phoneme and prosody modeling
- Fast to generate thanks to FFT
- Separate model can be used to fill in the phase

Text to Spectrogram Models

Neural TTS Paradigm

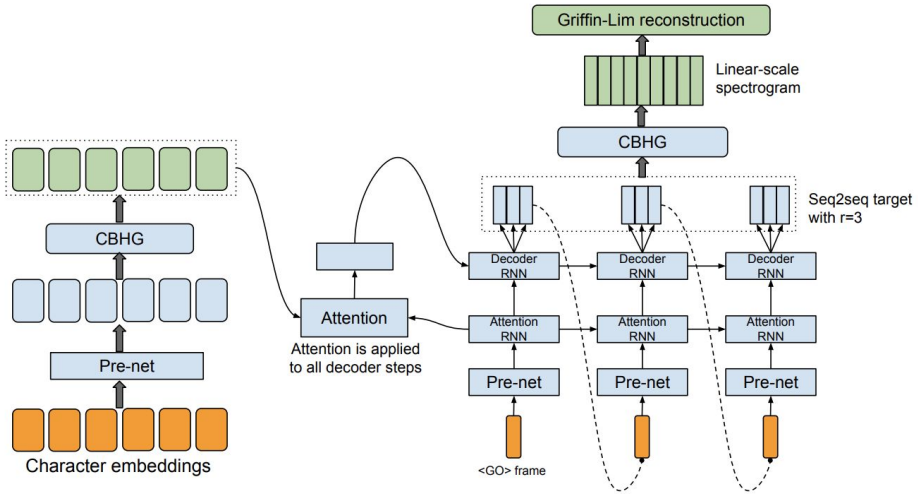


Sequence to Sequence Problem



Tacotron

- Encoder decoder model with attention
- Predicts mel spectrograms from character inputs
- “Information bottleneck” in pre-net crucial for regularization



Wang et al 2017

Tacotron

$$\{\boldsymbol{h}_j\}_{j=1}^{L} = \text{Encoder}(\{\boldsymbol{x}_j\}_{j=1}^{L}) \tag{1}$$

$$\boldsymbol{s}_i = \text{RNN}_{\text{Att}}(\boldsymbol{s}_{i-1}, \boldsymbol{c}_{i-1}, \boldsymbol{y}_{i-1}) \tag{2}$$

$$\boldsymbol{\alpha}_i = \text{Attention}(\boldsymbol{s}_i, \ \ldots) \qquad \boldsymbol{c}_i = \sum_j \alpha_{i,j} \boldsymbol{h}_j \tag{3}$$

$$\boldsymbol{d}_i = \text{RNN}_{\text{Dec}}(\boldsymbol{d}_{i-1}, \boldsymbol{c}_i, \boldsymbol{s}_i) \qquad \boldsymbol{y}_i = f_{\text{o}}(\boldsymbol{d}_i) \tag{4}$$

Wang et al 2017

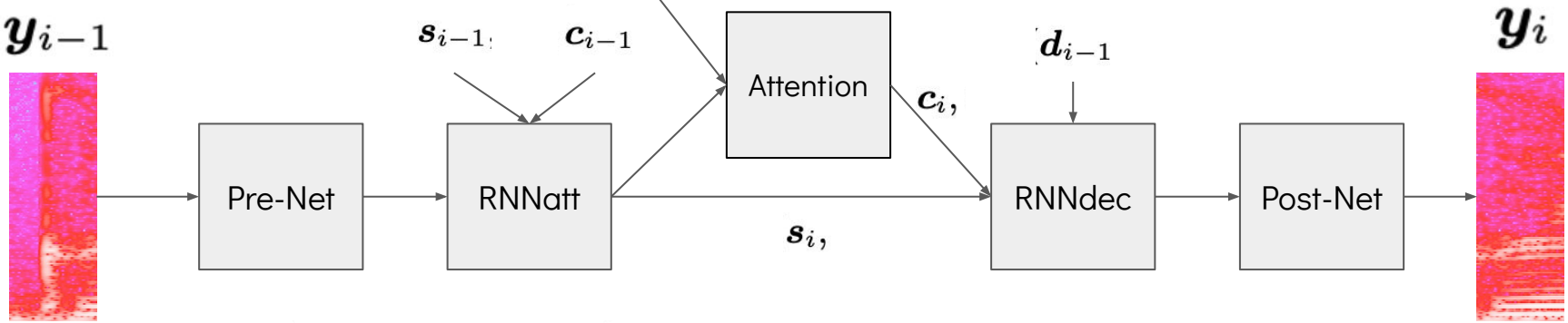
$$\{\boldsymbol{h}_j\}_{j=1}^{L} = \text{Encoder}(\{\boldsymbol{x}_j\}_{j=1}^{L})$$

Optionally takes previous alignment, encoder states

"g" "r" "a" "c" "e"

$$\boldsymbol{\alpha}_i = \text{Attention}(\boldsymbol{s}_i, \ \ldots)$$

$$\boldsymbol{c}_i = \sum_j \alpha_{i,j} \boldsymbol{h}_j$$

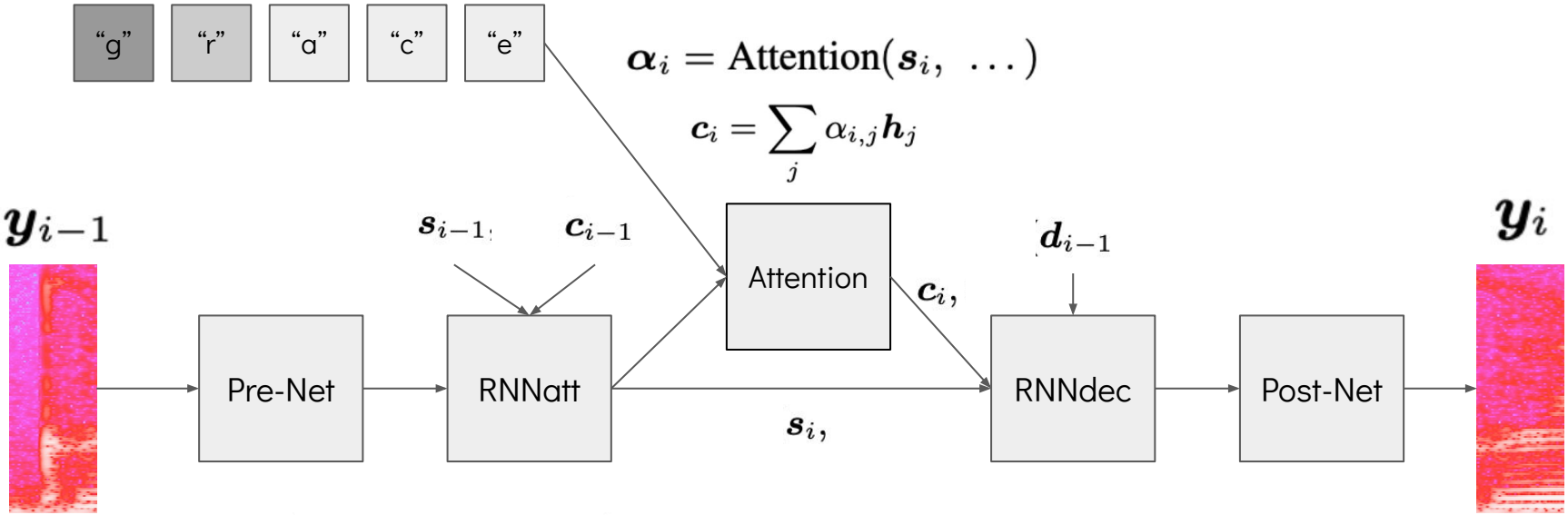


$$\boldsymbol{s}_i = \text{RNN}_{\text{Att}}(\boldsymbol{s}_{i-1}, \boldsymbol{c}_{i-1}, \boldsymbol{y}_{i-1}) \qquad \boldsymbol{d}_i = \text{RNN}_{\text{Dec}}(\boldsymbol{d}_{i-1}, \boldsymbol{c}_i, \boldsymbol{s}_i)$$

Can be 1-5 mel frames (reduction factor)

$i = 0$

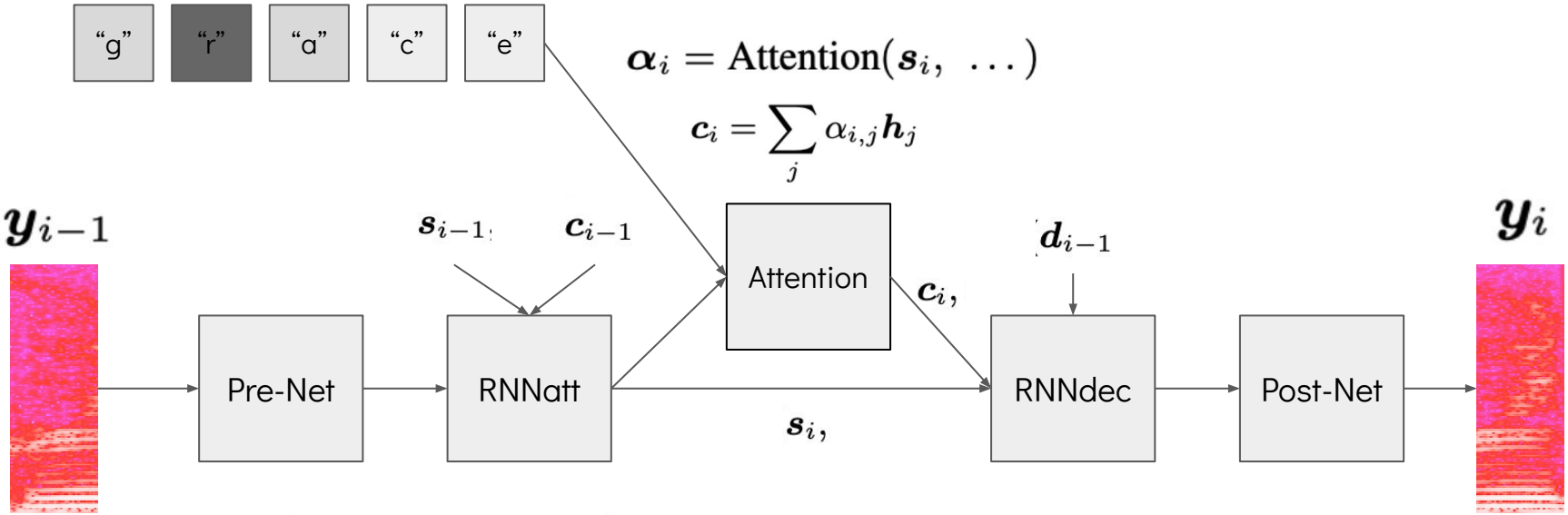
$$\{\boldsymbol{h}_j\}_{j=1}^{L} = \text{Encoder}(\{\boldsymbol{x}_j\}_{j=1}^{L})$$



$$\boldsymbol{s}_i = \text{RNN}_{\text{Att}}(\boldsymbol{s}_{i-1}, \boldsymbol{c}_{i-1}, \boldsymbol{y}_{i-1}) \qquad \boldsymbol{d}_i = \text{RNN}_{\text{Dec}}(\boldsymbol{d}_{i-1}, \boldsymbol{c}_i, \boldsymbol{s}_i)$$

$i = 1$

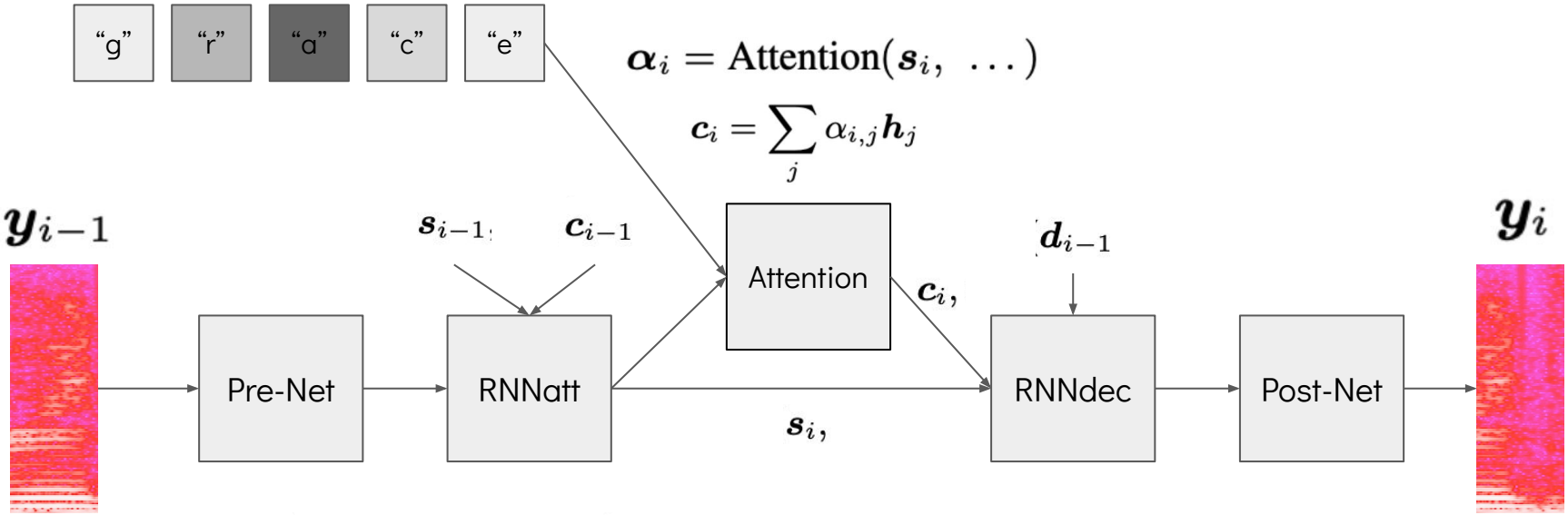
$$\{\boldsymbol{h}_j\}_{j=1}^{L} = \text{Encoder}(\{\boldsymbol{x}_j\}_{j=1}^{L})$$



$$\boldsymbol{s}_i = \text{RNN}_{\text{Att}}(\boldsymbol{s}_{i-1}, \boldsymbol{c}_{i-1}, \boldsymbol{y}_{i-1}) \qquad \boldsymbol{d}_i = \text{RNN}_{\text{Dec}}(\boldsymbol{d}_{i-1}, \boldsymbol{c}_i, \boldsymbol{s}_i)$$

$i = 2$

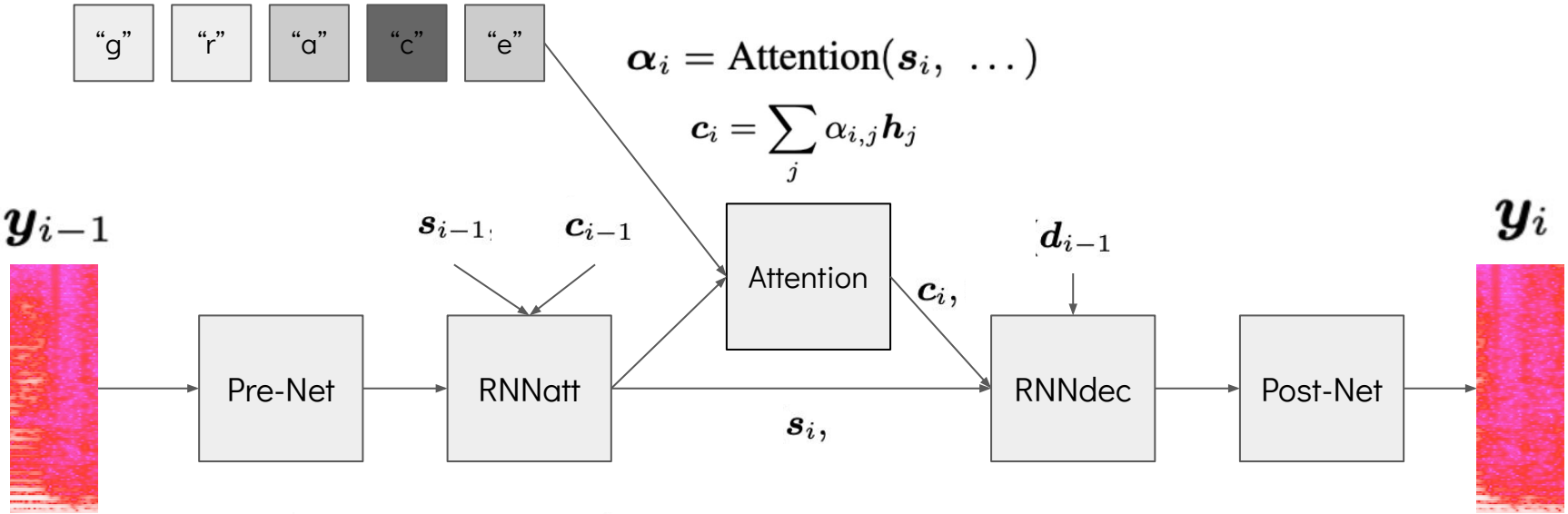
$$\{\boldsymbol{h}_j\}_{j=1}^{L} = \text{Encoder}(\{\boldsymbol{x}_j\}_{j=1}^{L})$$



$$\boldsymbol{s}_i = \text{RNN}_{\text{Att}}(\boldsymbol{s}_{i-1}, \boldsymbol{c}_{i-1}, \boldsymbol{y}_{i-1}) \qquad \boldsymbol{d}_i = \text{RNN}_{\text{Dec}}(\boldsymbol{d}_{i-1}, \boldsymbol{c}_i, \boldsymbol{s}_i)$$

$i = 3$

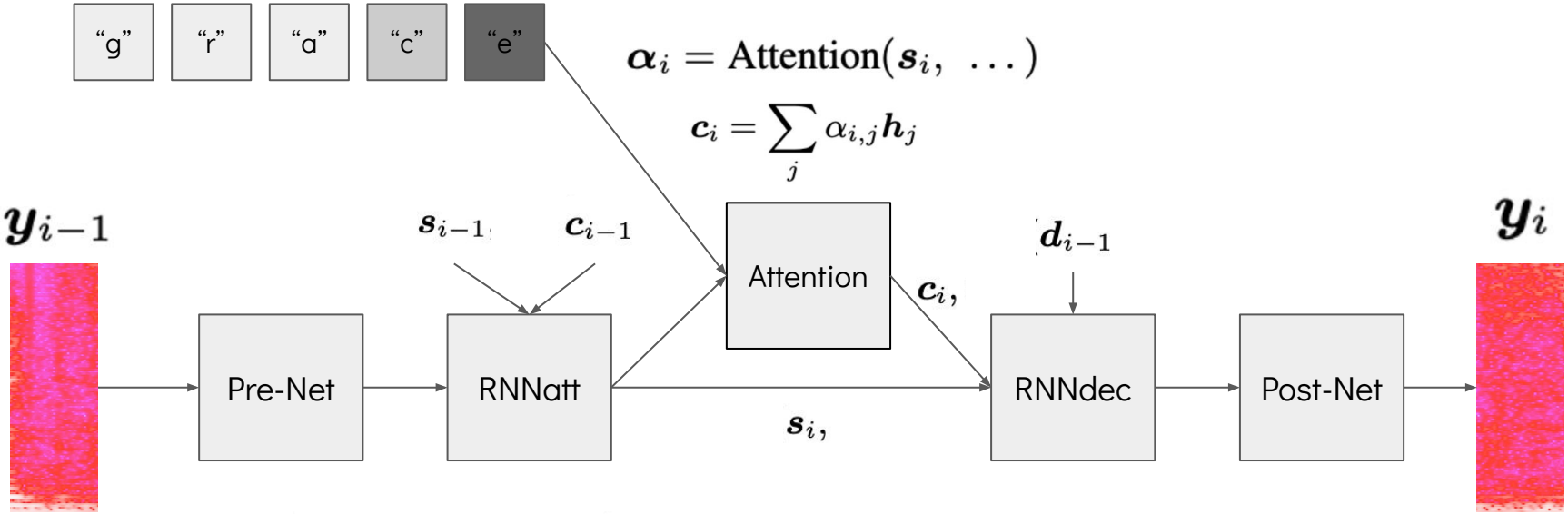
$$\{\boldsymbol{h}_j\}_{j=1}^{L} = \text{Encoder}(\{\boldsymbol{x}_j\}_{j=1}^{L})$$



$$\boldsymbol{s}_i = \text{RNN}_{\text{Att}}(\boldsymbol{s}_{i-1}, \boldsymbol{c}_{i-1}, \boldsymbol{y}_{i-1}) \qquad \boldsymbol{d}_i = \text{RNN}_{\text{Dec}}(\boldsymbol{d}_{i-1}, \boldsymbol{c}_i, \boldsymbol{s}_i)$$

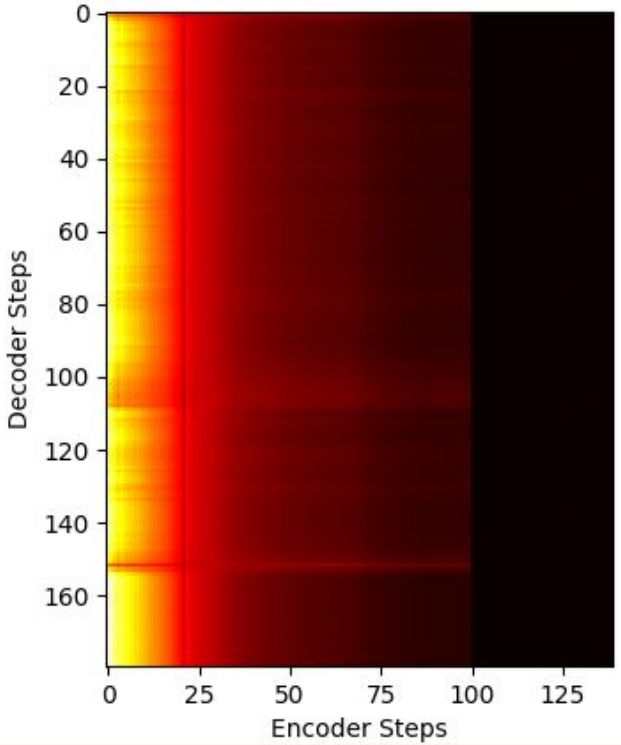
$i = 4$

$$\{\boldsymbol{h}_j\}_{j=1}^{L} = \text{Encoder}(\{\boldsymbol{x}_j\}_{j=1}^{L})$$

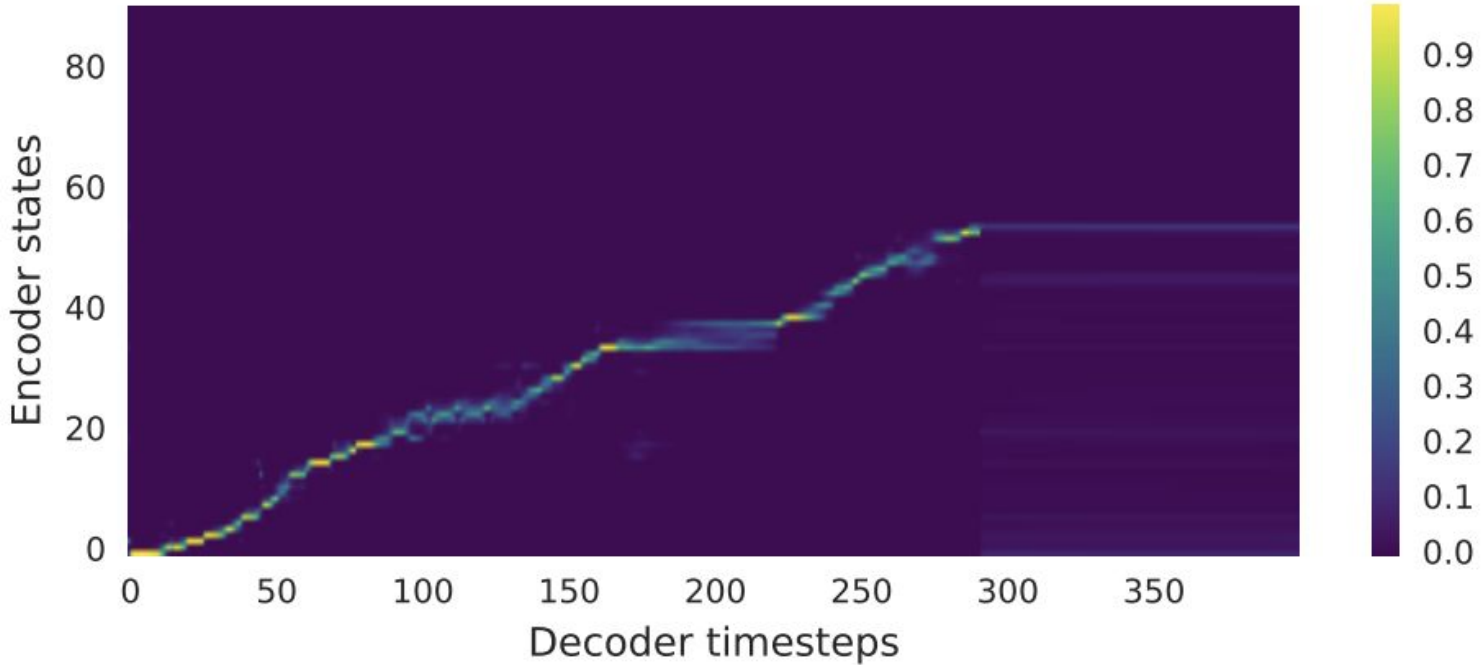


$$\boldsymbol{s}_i = \text{RNN}_{\text{Att}}(\boldsymbol{s}_{i-1}, \boldsymbol{c}_{i-1}, \boldsymbol{y}_{i-1}) \qquad \boldsymbol{d}_i = \text{RNN}_{\text{Dec}}(\boldsymbol{d}_{i-1}, \boldsymbol{c}_i, \boldsymbol{s}_i)$$

Learning an Alignment



Attention can be fickle



Wang et al. 2017

Many Forms of Attention

- Content Based (Bahdanau)
- Location Sensitive
- Location Relative (GMM, DCA)

Content Based Attention

RNN_att Query

Encoder Keys

$$e_{i,j} = v^T tanh(W s_i + V h_j)$$

$$\alpha_i = softmax(e_i)$$

Bahdanau et al. 2014

Location Sensitive Attention

Convolution with previous alignment

$$f_{i,j} = F * \alpha_{i-1}$$

$$e_{i,j} = v^T tanh(W s_i + V h_j + U f_{i,j})$$

$$\alpha_i = softmax(e_i)$$

Shen et al. 2018

Location Sensitive Attention

- Allows the model to explicitly use previous alignments for computing the next attention state
- Achieves much stronger alignments in practice than plain Bahdanau attention
- Enough model flexibility to learn a high quality text to spectrogram mapping

Dynamic Convolutional Attention

$$f_{i,j} = F * \alpha_{i-1}$$

$$G(s_i) = v_g^T tanh(W_g s_i + b_g)$$

Dynamic filters computed from attention state

$$g_{i,j} = G(s_i) * \alpha_{i-1}$$

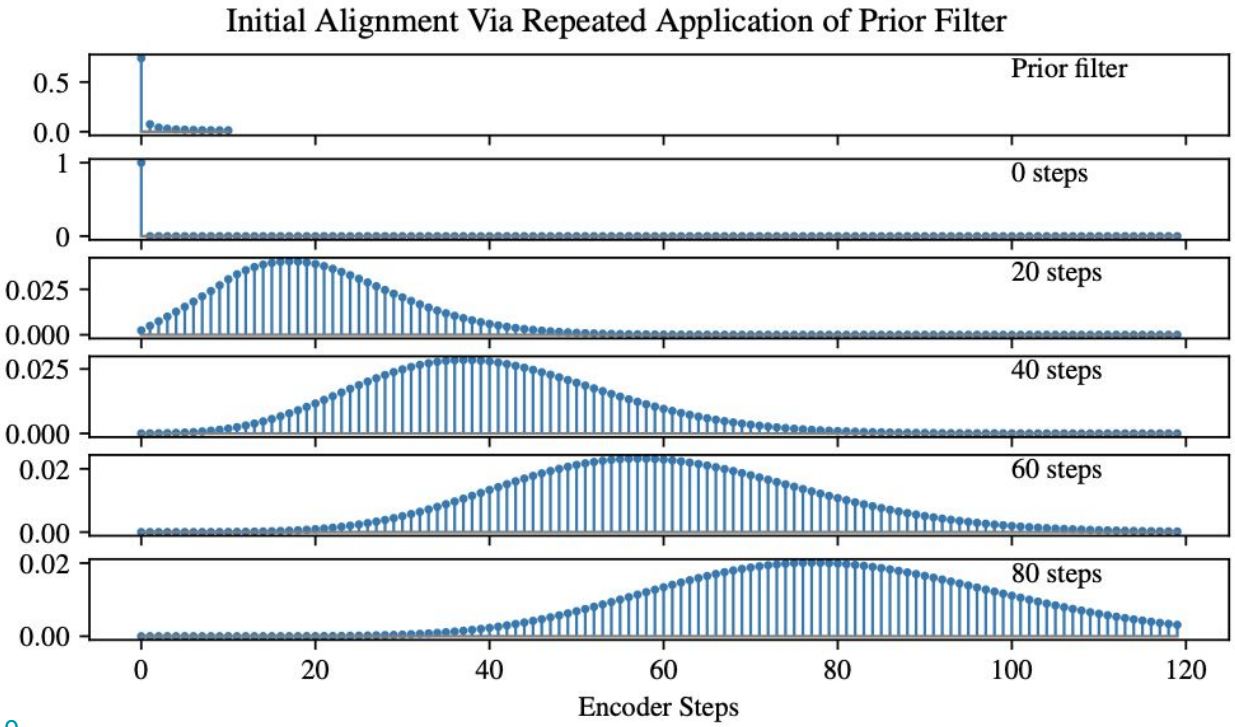
$$p_i = \log(P * \alpha_{i-1})$$

Prior bias to encourage monotonicity

$$e_{i,j} = v^T tanh(U f_{i,j} + T g_{i,j}) + p_{i,j}$$

Battenberg et al. 2019

$$p_i = \log(P * \alpha_{i-1})$$



Battenberg et al. 2019

Dynamic Convolutional Attention

- Dynamic filters on previous alignment instead of directly using the encoder outputs and query
- Add a prior bias which softly encourages monotonicity
- Learns even more consistent alignments than location sensitive attention
- Better generalization to long utterances
- Tends to reach an alignment faster

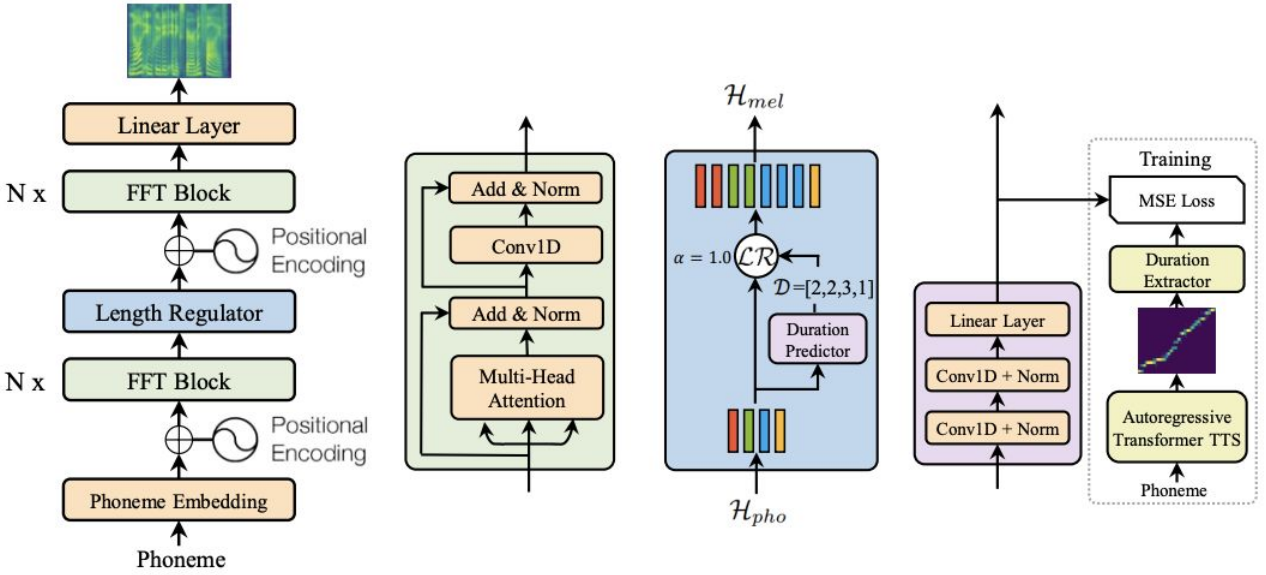
Tips for training attention TTS models

- Alignments are everything, a good alignment in training almost certainly means good generalization
- Make sure your examples are well trimmed, consider normalizing volume and removing especially noisy samples
- Use a location based attention. LSA is simple and works well. DCA/GMM can be even better
- Make sure your log mel spectrograms are well normalized
- Fine tuning from existing models can be useful for small/noisy datasets
- Reduction factor is your friend if you’re struggling to get an alignment

Attention model drawbacks

- Autoregressive => Slow
- Occasionally prone to skipping, repeating etc even with LSA, DCA

An Alternative: Explicit Duration Modelling



[Ren et al 2019]

FastSpeech 1/2

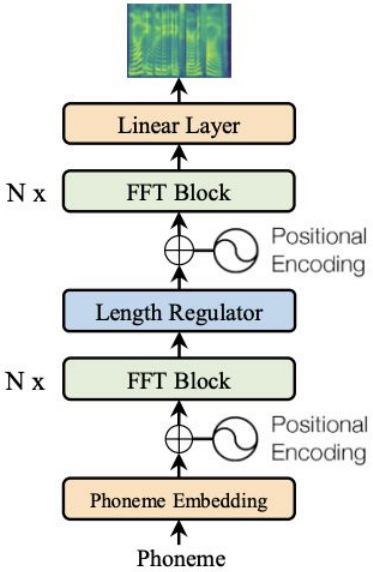
- Similar to earlier DNN TTS systems
- Explicitly predict phoneme durations, f0 and pitch
- Durations for training come from an autoregressive model (e.g. tacotron) or from traditional HMM forced alignments
- To match the input and output lengths, repeat input states according to phoneme durations
- Use a transformer to predict in parallel rather than frame by frame

Ren et al 2019, Ren et al. 2020

FastSpeech Architecture

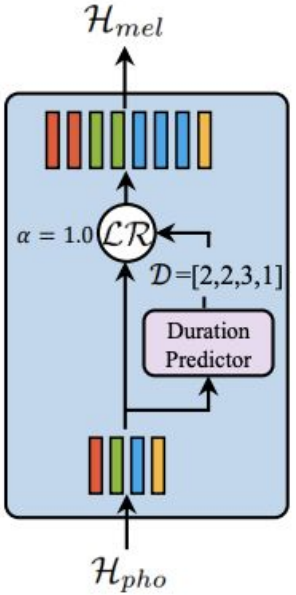
Encoder and Decoder are both fully parallel transformer blocks

(FFT = Feed Forward Transformer not Fast Fourier Transform)



Upsample by repeating encoder states by the predicted duration in frames

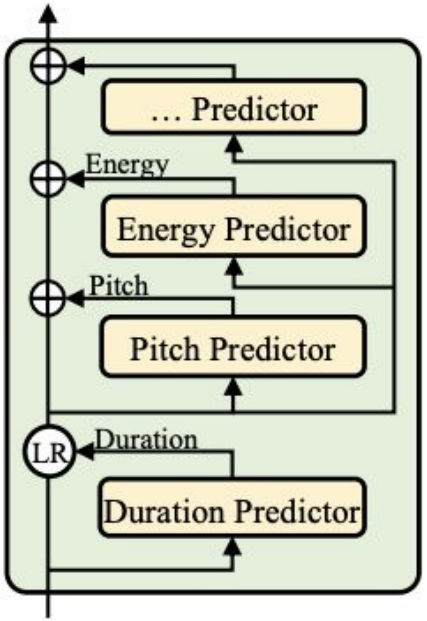
Length Regulator



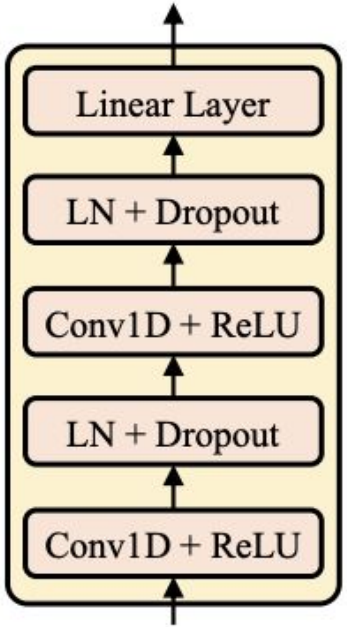
[Ren et al 2019]

FastSpeech 2 Variance Predictors

At training time use the ground truth duration, energy, f0 and pitch for synthesis and train predictors with MSE



Variance Predictor Structure:



Ren et al. 2020

Attention vs Duration based models

Attention-based

- No alignments needed
- Adaptable to diverse, noisy datasets
- Capable of more natural prosody

Duration-based

- Fast parallel inference
- Less chance of alignment problems
- Easier to train (Once you have alignments)
- More robust to silence in training data

A Compromise: FastSpeech with soft attention

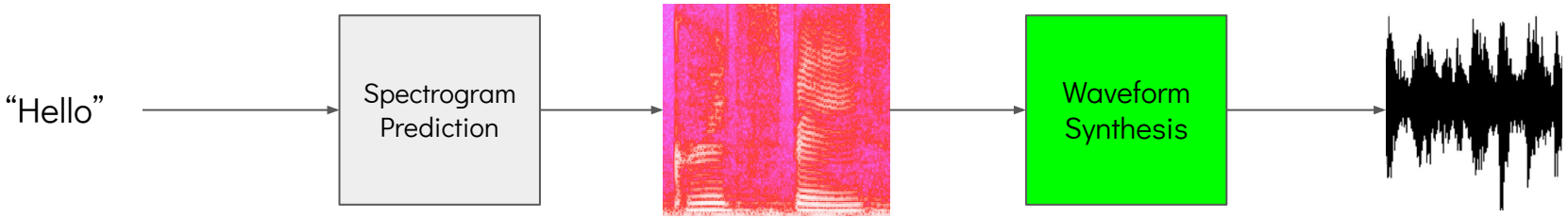
- Add a soft attention module to FastSpeech style TTS
- Compute a softmax across all pairs of text and spectrogram frames
- Use forward sum algorithm to compute the optimal alignment
- Can reuse CTC loss from ASR
- Examples: JETS

An Alternative: Flow-based Models

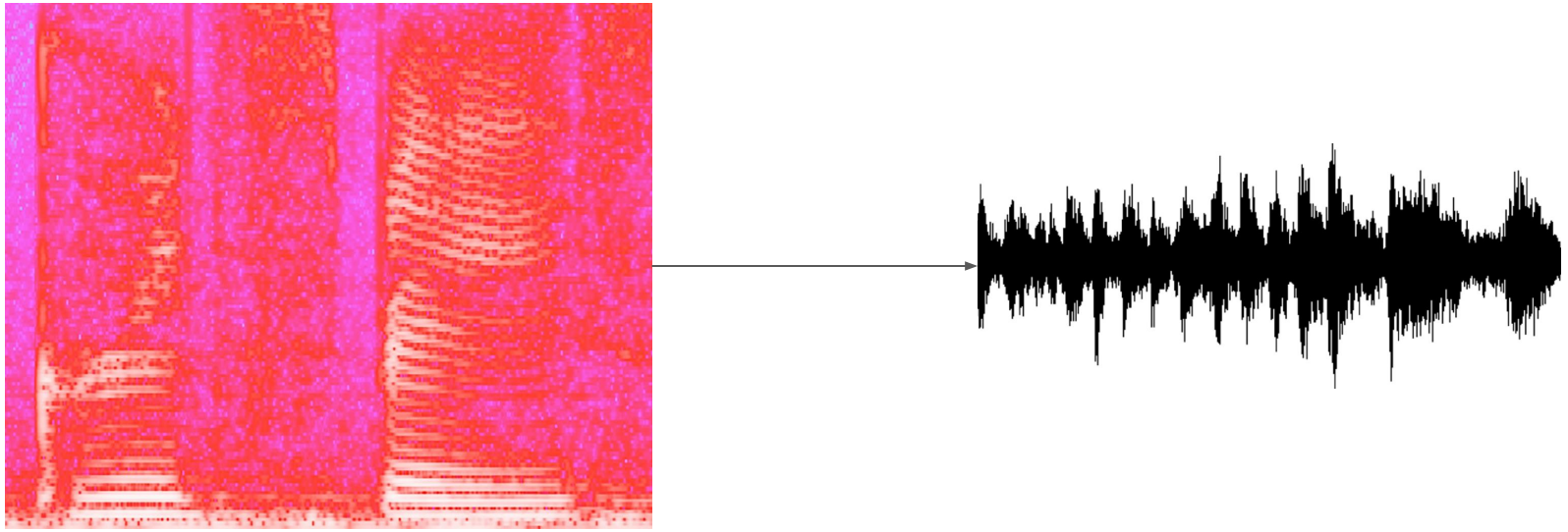
- Flow-based models combine transformer backbones with learned duration/attention
- All the benefits of FastSpeech – fast parallel inference, high quality
- All the benefits of Tacotron – no alignments needed, more flexible
- Examples included Glow-TTS, VITS, NaturalSpeech
- We'll discuss these in depth at the end of the lecture

Vocoders

Neural TTS Paradigm



Spectrogram to Waveform Conversion



Phase Prediction

- We have a magnitude log mel spectrogram from Tacotron/FastSpeech etc.
- We need to fill in the phase to get clear audio

Griffin-Lim

- Pure signal processing approach to phase reconstruction
- No learned parameters
- Iteratively reconstructs phase information from just the magnitude spectrogram
- Used in the original Tacotron paper

Griffin, Lim 1983

How Griffin-Lim Works

- The STFT transform is not perfectly invertible
- Many magnitude/phase pairs which cannot be produced from 1D audio
- Start with a random prediction for the phase
- Iteratively apply istft and stft to generate more “consistent” spectrograms which could have been generated by a time-series signal
- These sound much clearer than random/zero phase in practice

```python
import numpy as np
import librosa

def griffinlim(magnitude_spec, num_iters=50):
    angles = np.exp(2j * np.pi * np.random.rand(*magnitude_spec.shape))

    for i in range(num_iters):
        spectrogram = magnitude_spec.astype(np.complex) * angles
        inverse = librosa.istft(spectrogram)
        rebuilt = librosa.stft(inverse)
        angles = np.exp(1j * np.angle(rebuilt))

    spectrogram = magnitude_spec.astype(np.complex) * angles
    inverse = librosa.istft(spectrogram)

    return inverse
~
~
```

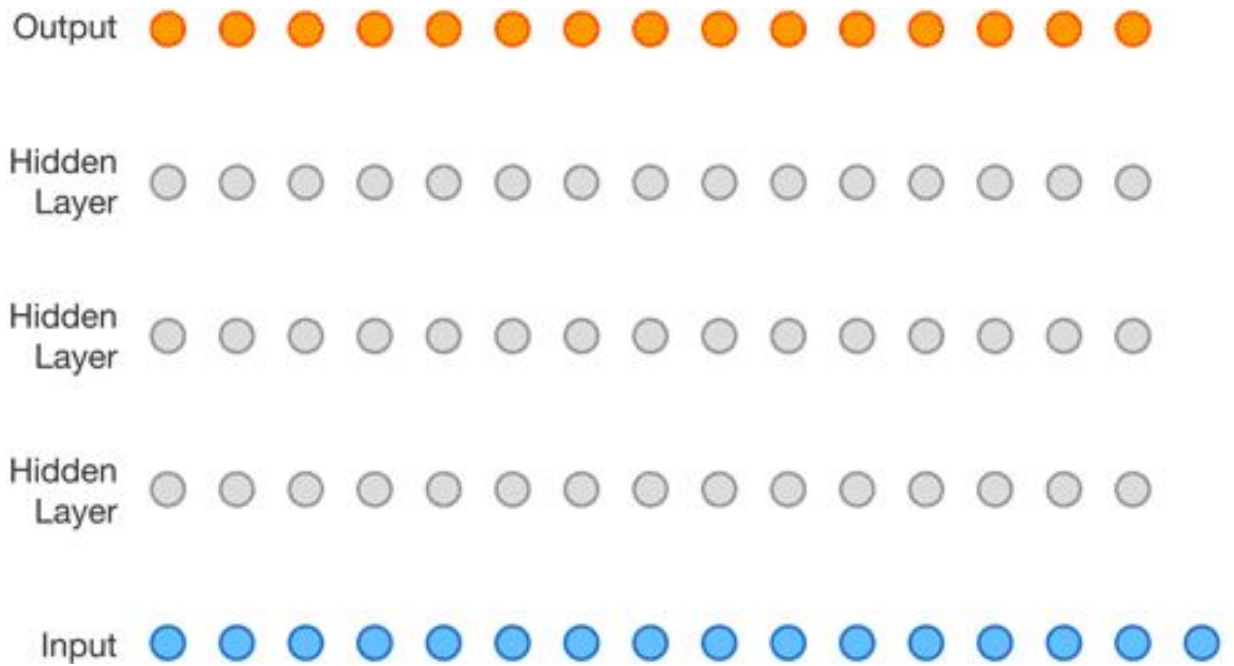
Griffin-Lim Limitations

- Since it has no parameters, Griffin-Lim can only provide a coarse reconstruction of the phase
- Neural models trained on spectrogram/audio pairs are needed for higher quality outputs

Neural Networks on Raw Audio

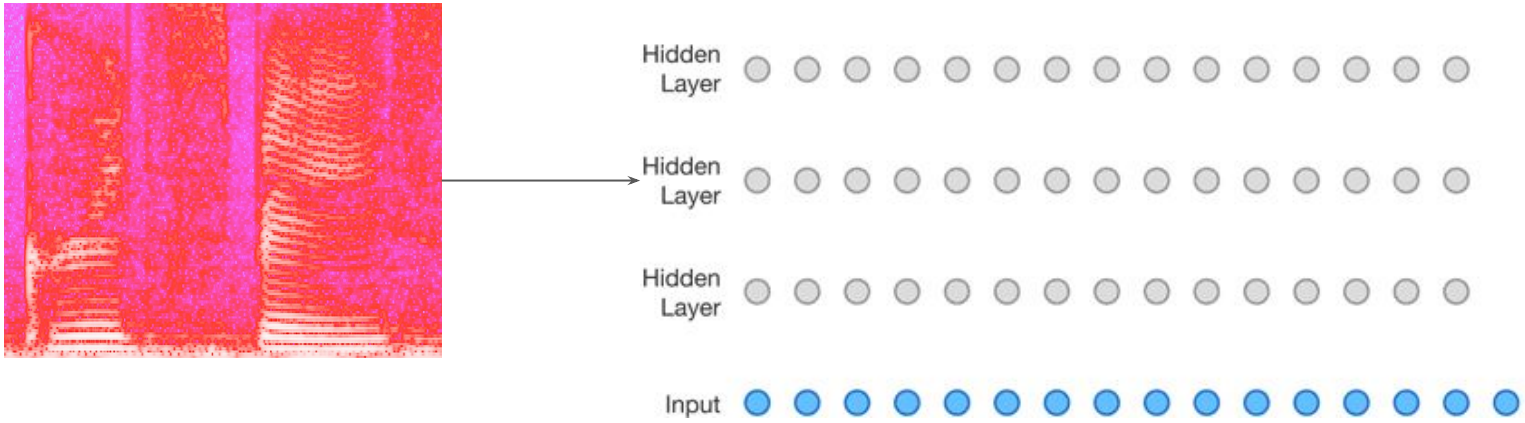
- Extremely long time series in the output
- Endless training data
- A test piece for modern generative models

WaveNet



https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

WaveNet



https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

WaveNet

- Generative model of audio
- Autoregressive: generates one sample of audio at a time
- Many layers of dilated convolutions for a high receptive field
- Very high output quality
- Extremely Slow
- Can be conditioned on linguistic features or spectrograms to generate speech for specific utterances

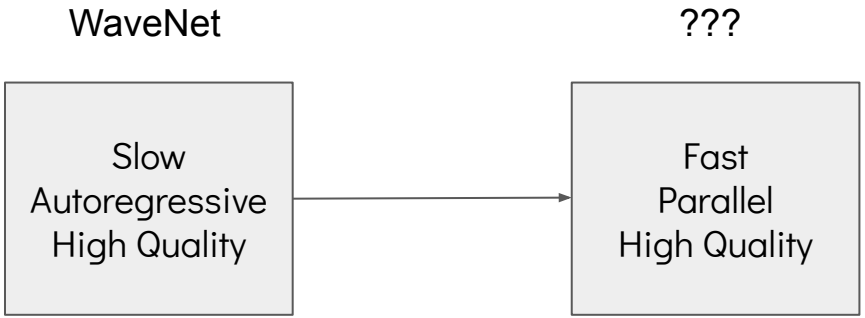
Oord et al. 2016

WaveRNN

- Hyper-optimize a simple, autoregressive GRU model instead of WaveNet
- Up to 96% (!) weight sparsification and subsampling
- Runs ~4x real time even on smartphone CPUs
- Diverse applications in audio (see LPCNet, Lyra codec / WaveNetEQ packet loss smoothing)

Kalchbrenner et al. 2018

Parallelizing WaveNet



Parallelizing WaveNet

WaveNet

Parallel Training
Sequential Inference

???

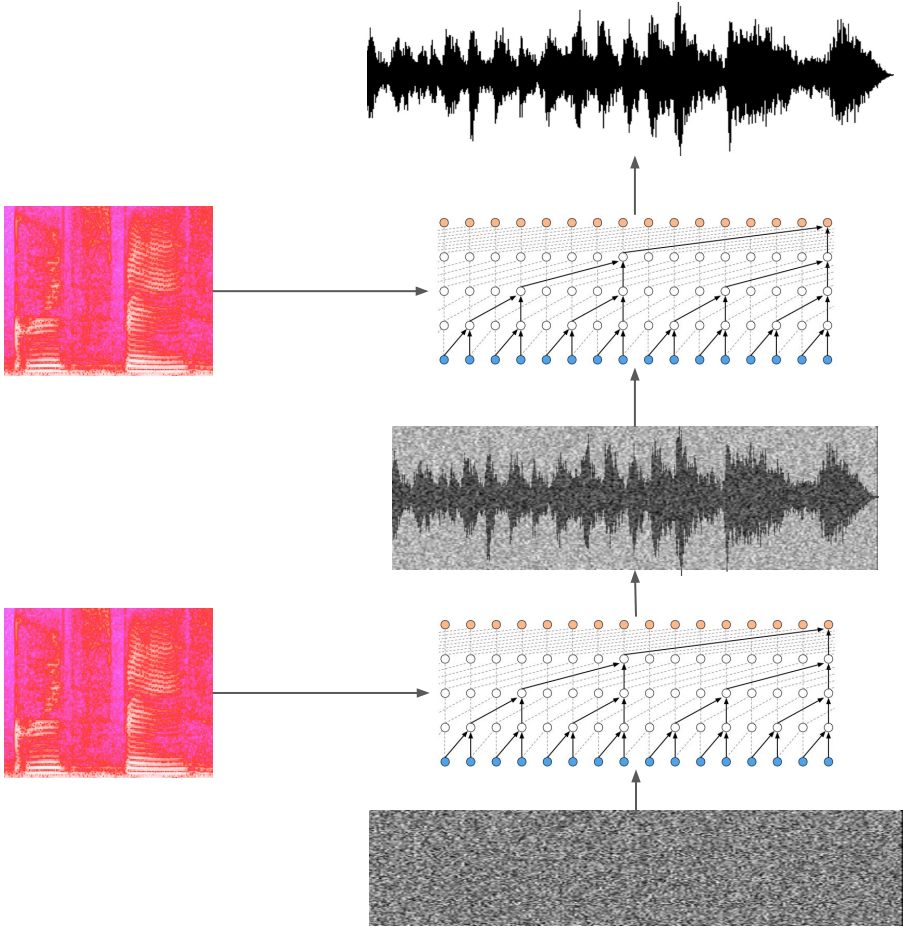
Sequential Training
Parallel Inference

Inverse Autoregressive Flows

- Sample the number of audio samples we want to generate from a unit gaussian distribution
- Transform those samples by a mean and variance predicted by a neural net
- This produces the full waveform in parallel
- Each step is as follows:

$$x_t = z_t \cdot s(\boldsymbol{z}_{<t}, \boldsymbol{\theta}) + \mu(\boldsymbol{z}_{<t}, \boldsymbol{\theta})$$

where s and mu are produced by running a WaveNet on z



x_t

$z_t \sim N(0, 1)$

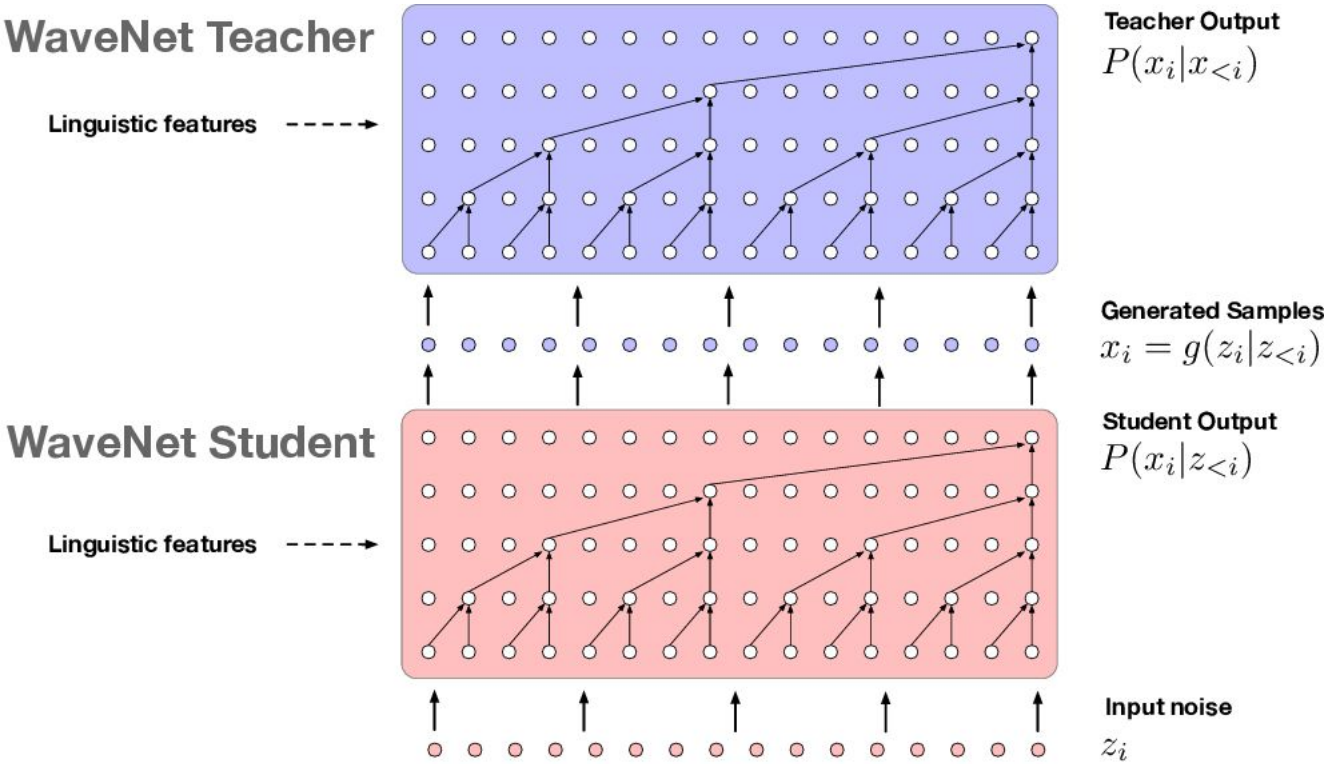
Inverse Autoregressive Flows

- Fast, parallel sampling
- Closed form for gradient update requires an autoregressive calculation
- This makes directly training the flow intractable
- In a sense, the inverse of WaveNet

Parallel WaveNet: Student and Teacher

- Use a trained normal WaveNet model as a “teacher” for an IAF
- Minimize the KL divergence between the output distribution of the IAF and teacher wavenet
- This can be done in parallel, so training is fast
- Once trained, the student IAF can then perform inference in parallel on its own

Parallel WaveNet

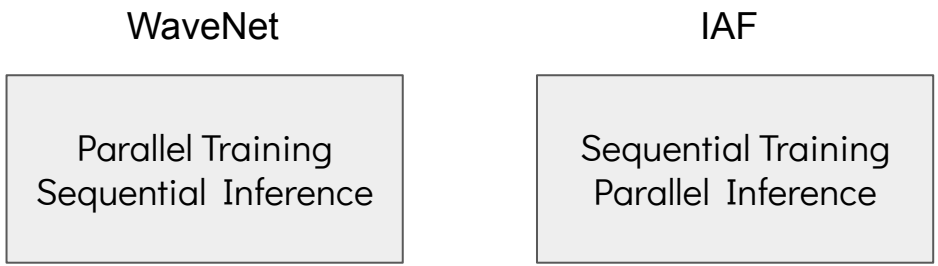


Oord et al. 2017

Parallel WaveNet Issues

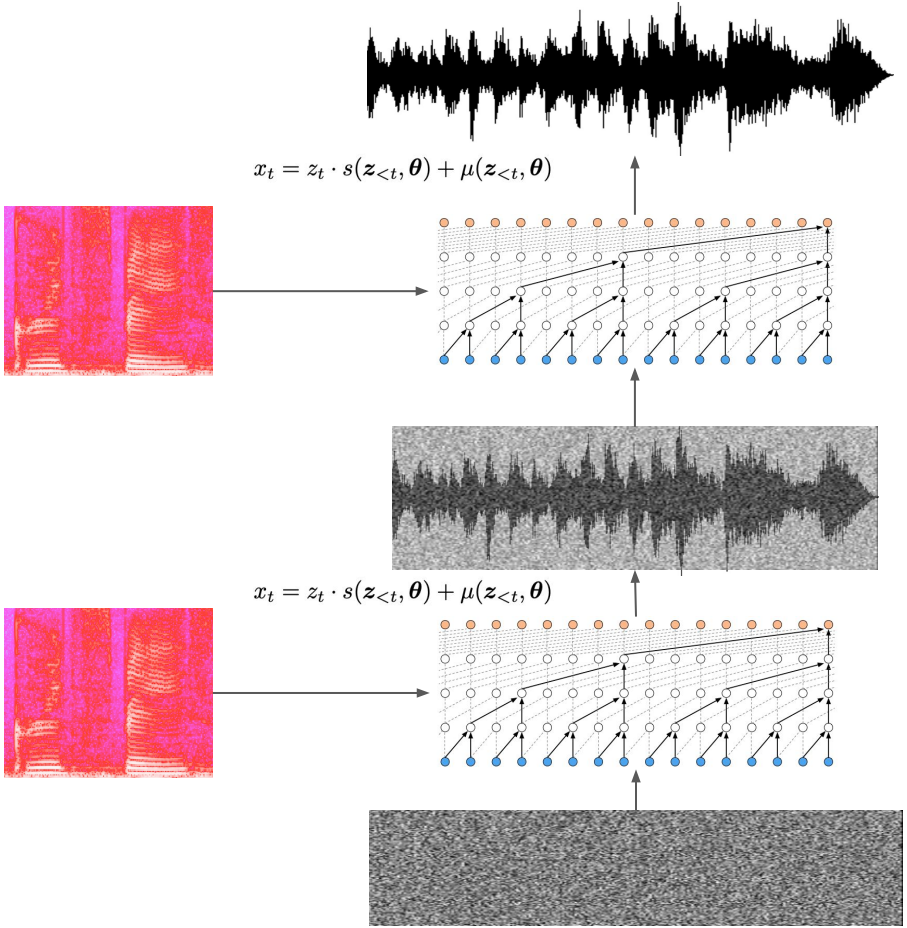
- Have to train two separate models
- Even with Clarinet, training the student distribution to match the teacher is extremely finicky
- Perceptual losses required which are hand tuned
- In practice, very hard to replicate the quality of the original WaveNet

Parallelizing WaveNet



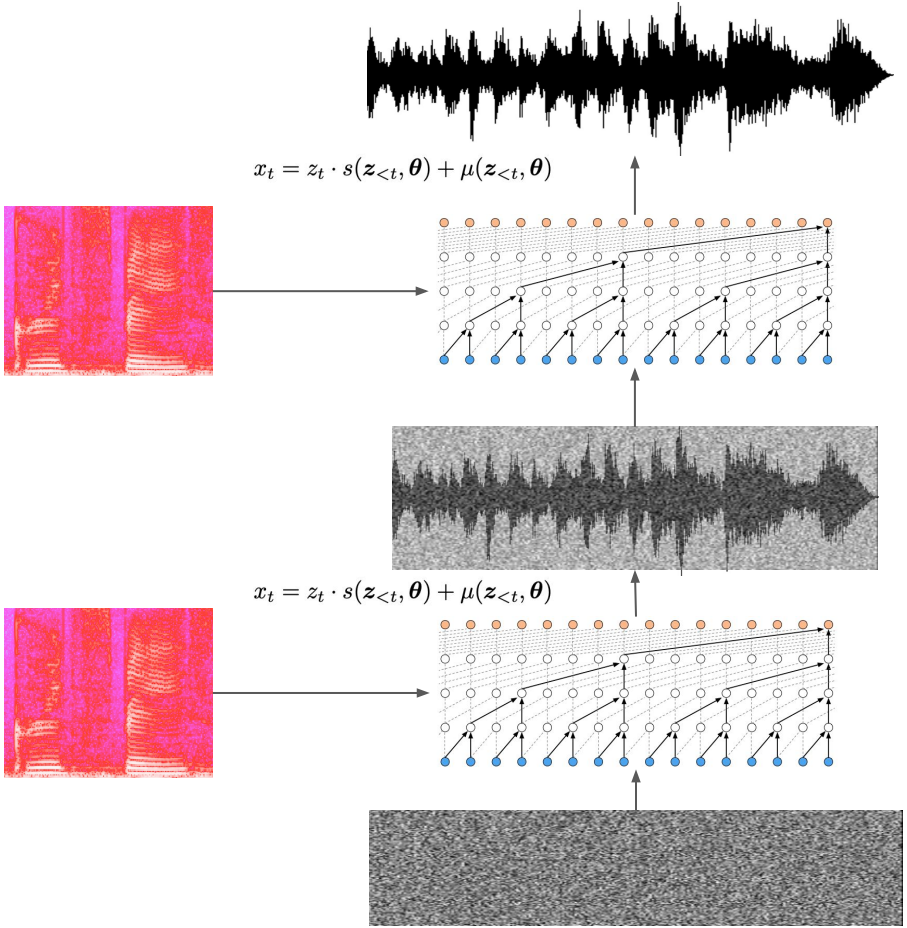
Parallelizing WaveNet

WaveNet	IAF	???
Parallel Training Sequential Inference	Sequential Training Parallel Inference	Parallel Training Parallel Inference


$x_t = z_t \cdot s(\boldsymbol{z}_{<t}, \boldsymbol{\theta}) + \mu(\boldsymbol{z}_{<t}, \boldsymbol{\theta})$
$x_t = z_t \cdot s(\boldsymbol{z}_{<t}, \boldsymbol{\theta}) + \mu(\boldsymbol{z}_{<t}, \boldsymbol{\theta})$

x_t

$z_t \sim N(0, 1)$



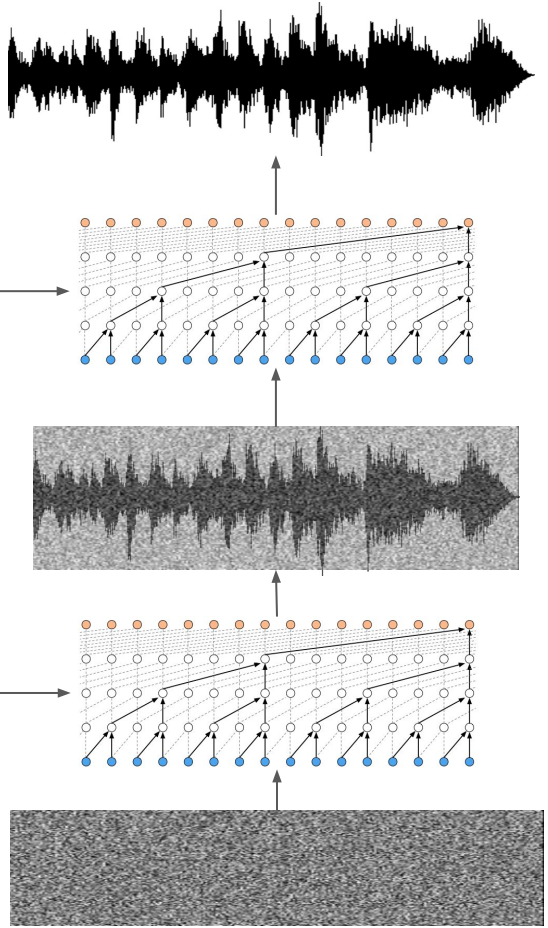
x_t

Autoregressive WaveNet => intractable log likelihood

Flow Training ⇔ WaveNet Inference => SLOW

$z_t \sim N(0, 1)$

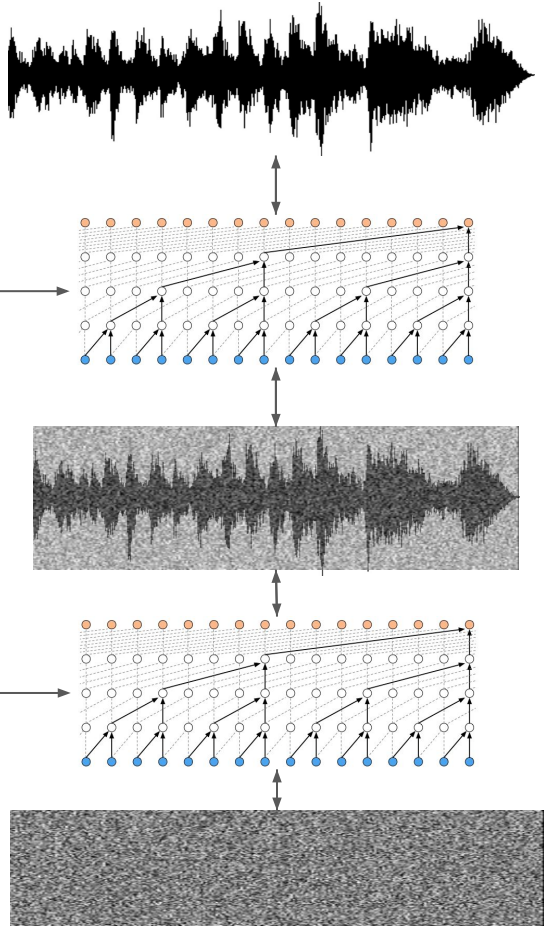
What if this was invertible?



x_t

$z_t \sim N(0, 1)$

What if this was invertible?



x_t

Inference:
Sample z and transform to x

Training:
Transform x to z and enforce a normal distribution on z

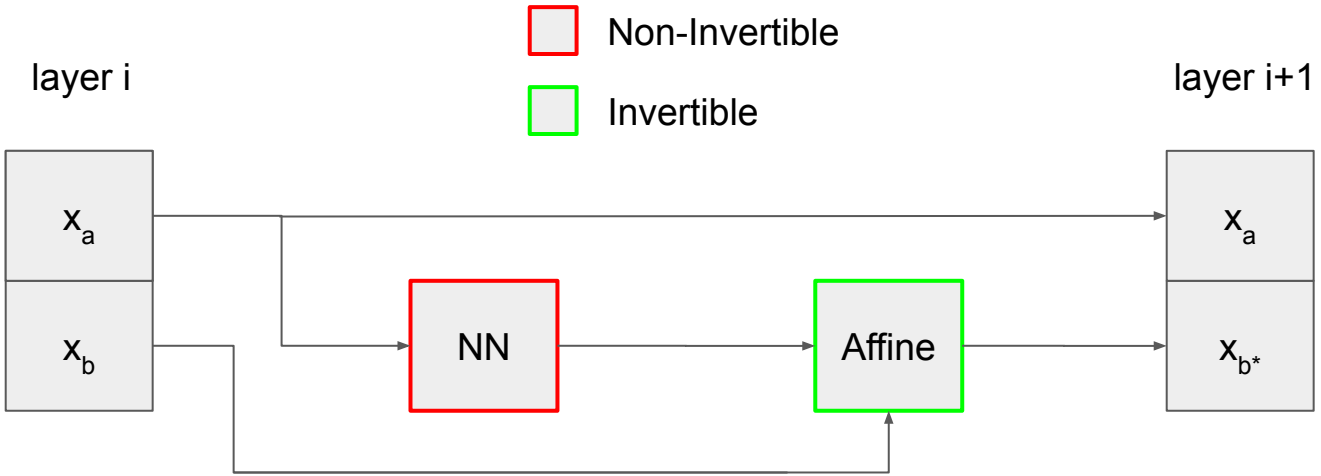
$z_t \sim N(0, 1)$

Glow

- Invertible flow based model
- Originally applied to image generation by OpenAI
- Quickly repurposed for audio generation with WaveGlow

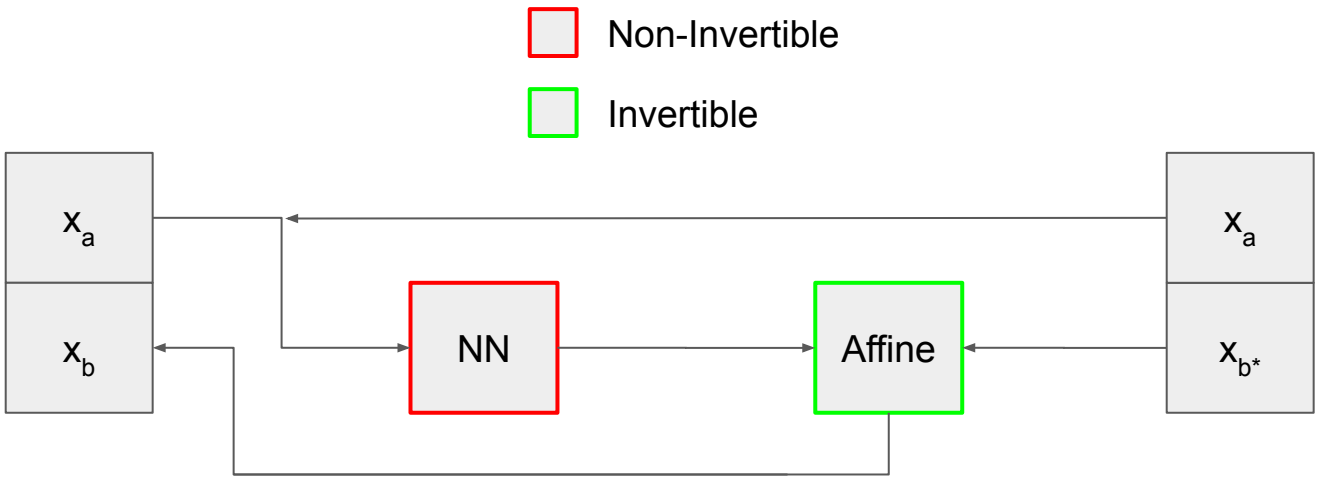
Kingma, Dhariwal 2018, Prenger et al. 2018

Affine Coupling Layer - Forwards



In the forward pass, x_a is unchanged and used to transform x_b into x_{b*}

Affine Coupling Layer - Backwards



In the backwards pass, WN produces the same scale and bias for the affine transformation since x_a is the same. This means we can just invert the affine transformation to transform x_{b^*} to x_b

Affine Coupling Layers

$$\boldsymbol{x}_a, \boldsymbol{x}_b = split(\boldsymbol{x})$$

$$(\log \boldsymbol{s}, \boldsymbol{t}) = WN(\boldsymbol{x}_a, mel\text{-}spectrogram)$$

$$\boldsymbol{x}_b\prime = \boldsymbol{s} \odot \boldsymbol{x}_b + \boldsymbol{t}$$

$$\boldsymbol{f}^{-1}_{coupling}(\boldsymbol{x}) = concat(\boldsymbol{x}_a, \boldsymbol{x}_b\prime)$$

Prenger et al. 2018

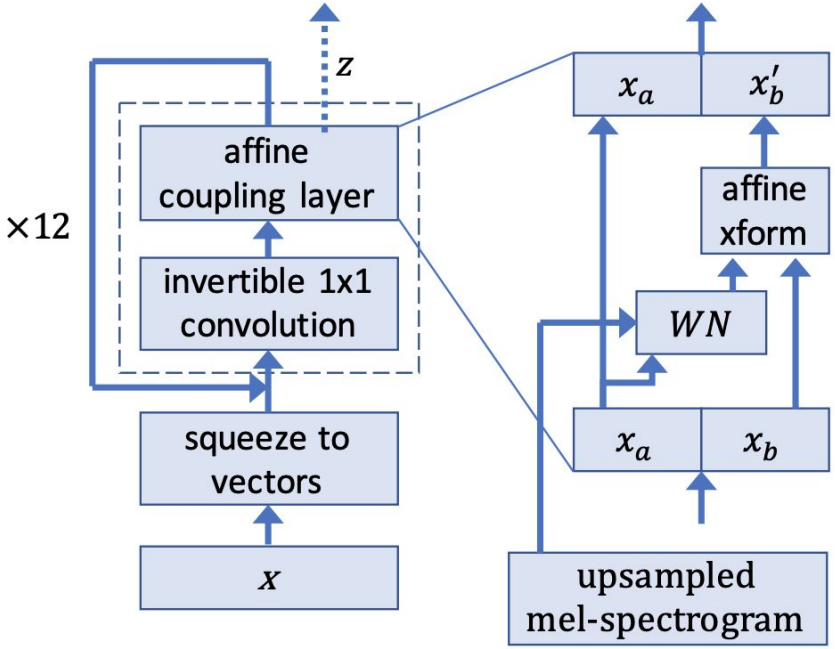
Mixing Channels

- Affine Coupling Layers can only transform half the input at a time
- Need a way to mix the channels between coupling layers

Invertible 1x1 Convolution

- 1x1 Convolution with a square kernel
- Initialize the kernel to be an invertible, orthonormal matrix
- Add a term to the loss to ensure it stays invertible in training
- For the backwards pass we just invert the kernel
- Now the channels are mixed between coupling layers

WaveGlow Architecture



Prenger et al. 2018

WaveGlow Loss Function

Fit z to a unit Gaussian Distribution

$$\log p_\theta(\boldsymbol{x}) = -\frac{\boldsymbol{z}(\boldsymbol{x})^T \boldsymbol{z}(\boldsymbol{x})}{2\sigma^2}$$

Change of variables from coupling

$$+ \sum_{j=0}^{\#coupling} \log \boldsymbol{s}_j(\boldsymbol{x}, \textit{mel-spectrogram})$$

Ensure 1x1 conv kernels remain invertible

$$+ \sum_{k=0}^{\#conv} \log\det|\boldsymbol{W}_k|$$

[Prenger et al. 2018]

WaveGlow

- Directly maximising likelihood makes training much more stable
- Eliminates the needs for perceptual losses
- Only have to train one model
- Quality equal to WaveNet
- Synthesize audio in parallel

Parallelizing WaveNet

WaveNet	IAF	WaveGlow
Parallel Training Sequential Inference	Sequential Training Parallel Inference	Parallel Training Parallel Inference

Can we go faster?

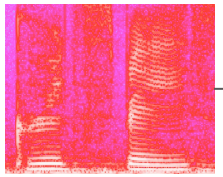
- WaveGlow requires a powerful GPU for fast inference
- WaveRNN requires heavy optimization to run real time on CPUs
- Is there an alternative?

GAN-based Vocoders

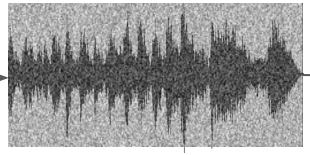
- Generative adversarial networks applied to audio generation
- Simultaneously train two networks: a generator and a discriminator
- Generator produces audio from the spectrograms to be as close as possible to the ground truth audio
- Discriminator trained to distinguish generator outputs from real audio
- Examples include MelGAN, Parallel WaveGAN, HiFiGAN

LSGAN Architecture

1 = Real
0 = Fake



Generator



Discriminator

P(Real)

Discriminator Loss

$$\mathcal{L}_{Adv}(D;G) = \mathbb{E}_{(x,s)}\left[(D(x)-1)^2 + (D(G(s)))^2\right]$$

Discriminator

P(Real)

Generator Loss

$$\mathcal{L}_{Adv}(G;D) = \mathbb{E}_{s}\left[(D(G(s))-1)^2\right]$$

Mao et al. 2016

Additional Losses for Audio GANs

- Direct reconstruction loss on mel spectrograms

$$\mathcal{L}_{Mel}(G) = \mathbb{E}_{(x,s)}\left[||\phi(x) - \phi(G(s))||_1\right]$$

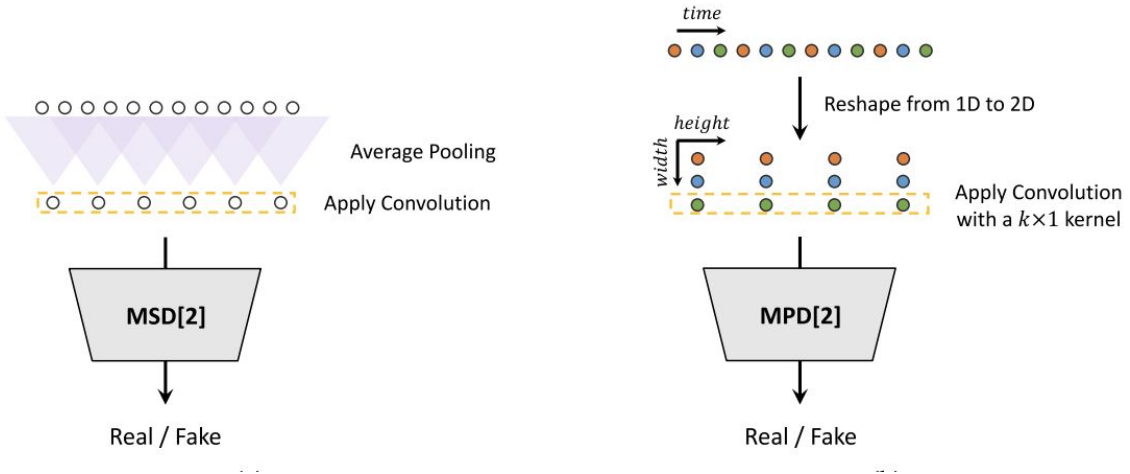
- Discriminator feature map L1 loss

$$\mathcal{L}_{FM}(G;D) = \mathbb{E}_{(x,s)}\left[\sum_{i=1}^{T}\frac{1}{N_i}||D^i(x) - D^i(G(s))||_1\right]$$

Kumar et al 2019, Kong et al. 2020

Multi-scale/multi-period Discriminators

- Multiple discriminators at different scales/periods are helpful
- Capture long term dependencies



Kong et al. 2020, Kumar et al 2019

GANs

- Very fast parallel GPU and CPU synthesis
- Quality approaching or matching WaveNet/WaveGlow/WaveRNN
- Require carefully designed additional losses to perform well
- Good open source implementations

Summary

- GAN based vocoders have the best quality/latency trade offs currently
- HiFiGAN is a great choice – high performance and high quality

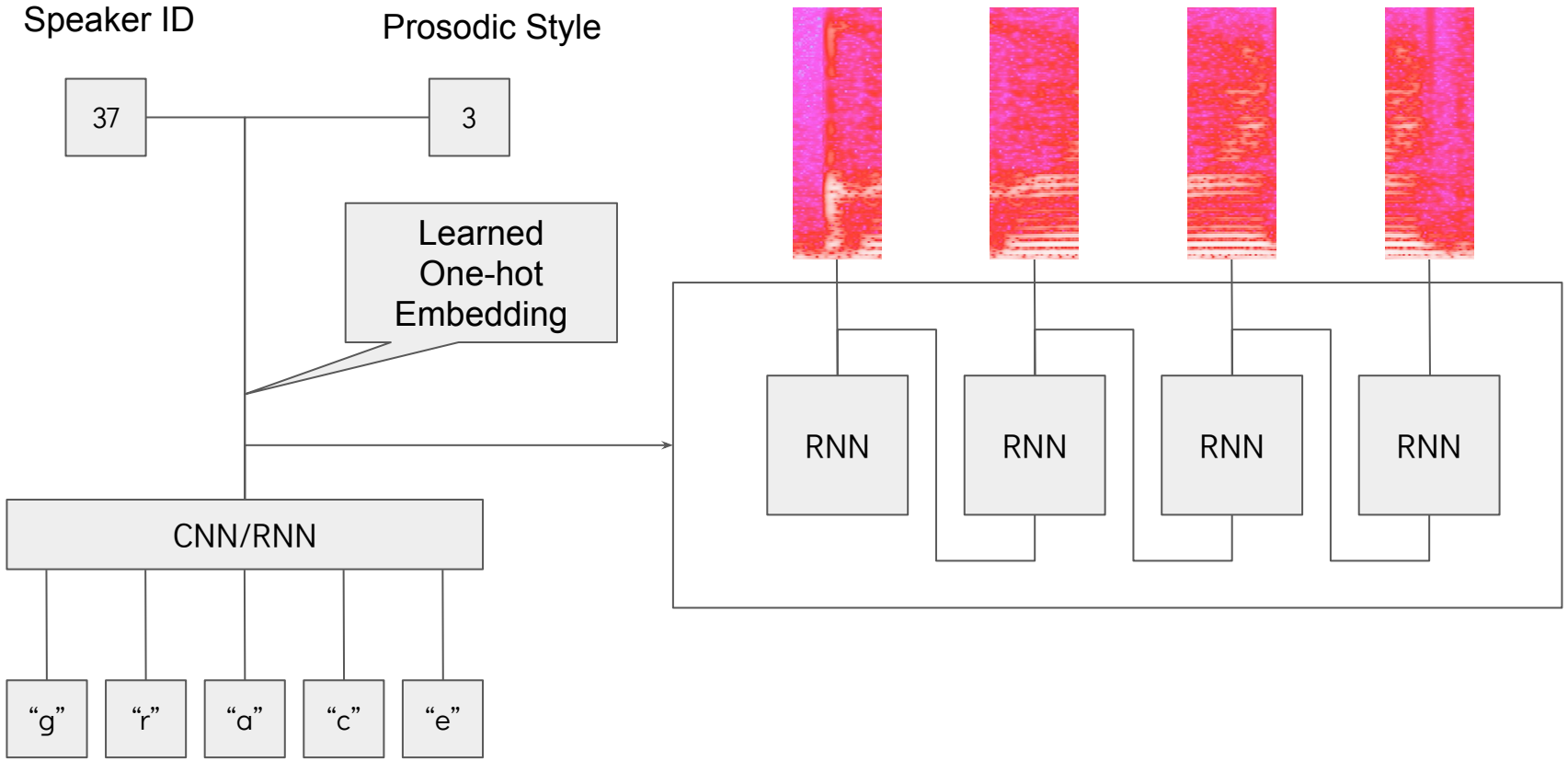
Speaker and Style Embeddings

Expanding the “text” in TTS

- TTS is fundamentally a one-to-many mapping
- The same text has infinitely many voicings
- Controllable speaker and prosody is very useful in dialog systems and elsewhere

Speaker/Style with One Hot Labels

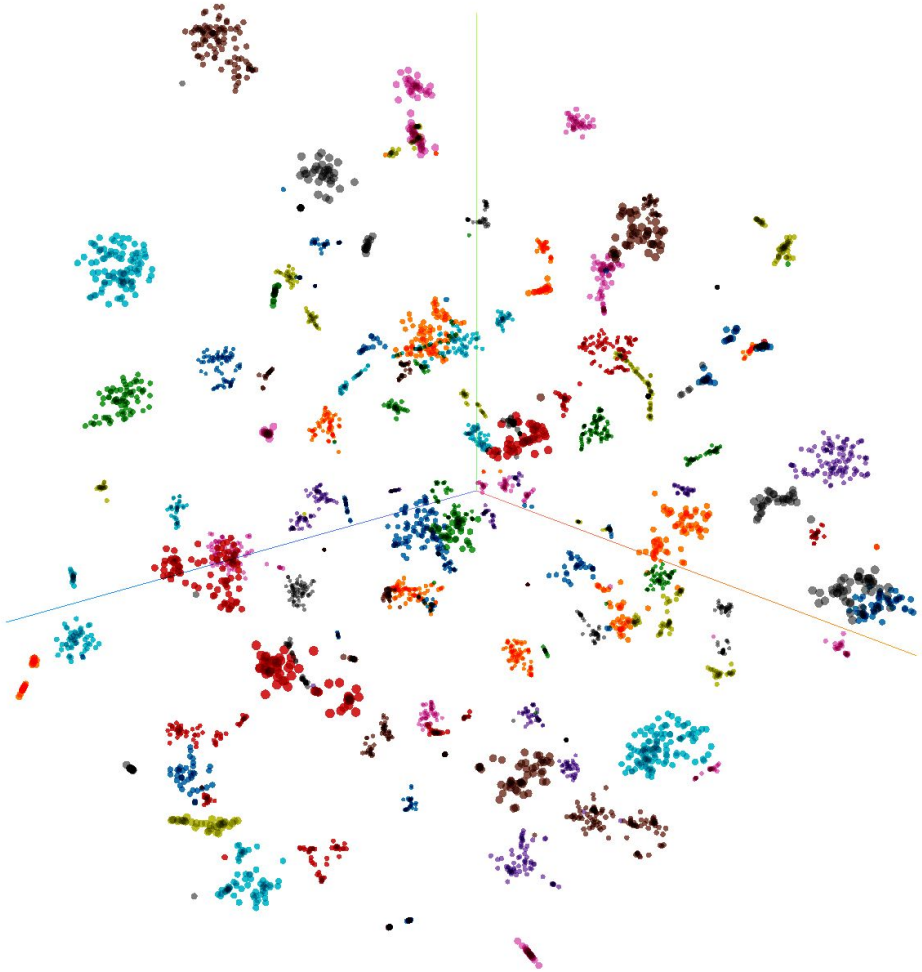
- Enumerate your speakers and/or styles and label the training data with them
- During training, learn an embedding for each speaker/style by passing a one hot encoding to the encoder
- At inference, pass in the corresponding speaker/style embedding
- Simple and easy to train but constrained by the breadth of your labels


Speaker ID
Prosodic Style
37
3
Learned
One-hot
Embedding
RNN
RNN
RNN
RNN
CNN/RNN
“g”
“r”
“a”
“c”
“e”

Learned Speaker Embeddings

- Train with large datasets of speaker-labelled audio
- Feed frozen embeddings to TTS model at training and inference time
- If the training dataset is sufficiently diverse, zero shot synthesis is possible for new speakers with a single utterance
- Audio Samples

Jia et al. 2018



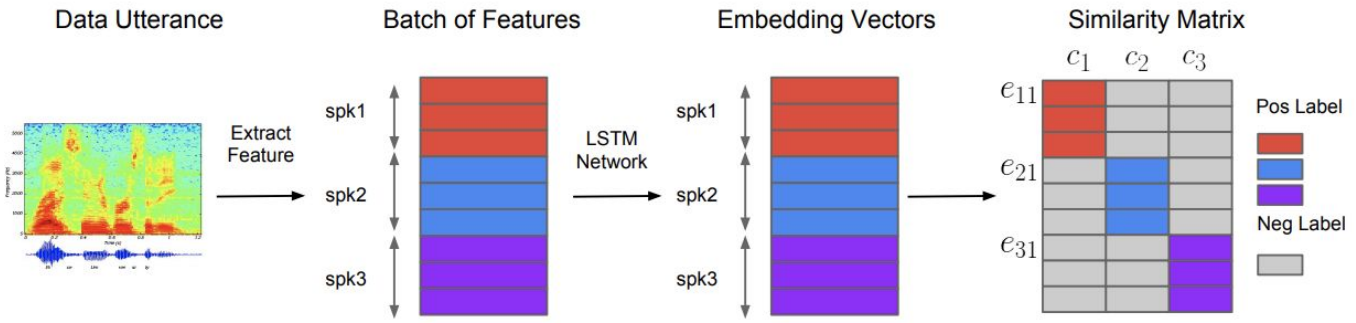


Fig. 1. System overview. Different colors indicate utterances/embeddings from different speakers.

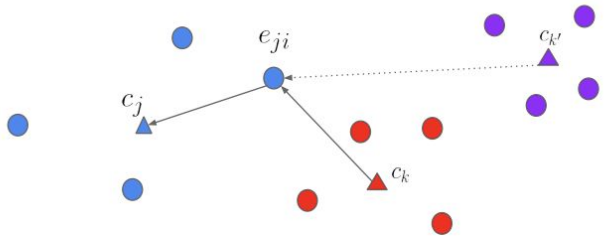
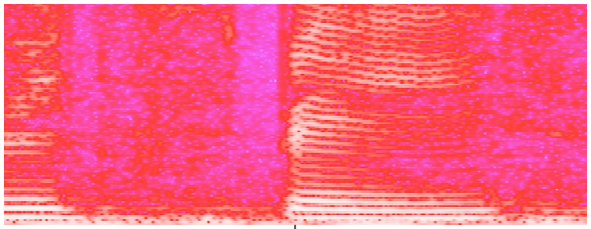


Fig. 2. GE2E loss pushes the embedding towards the centroid of the true speaker, and away from the centroid of the most similar different speaker.

Wan et al. 2017

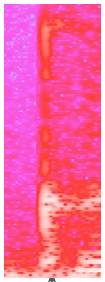
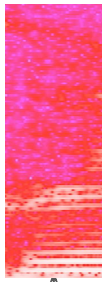
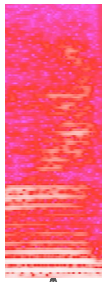
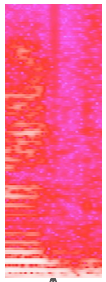
Speaker Spectrogram



CNN/RNN (frozen)

CNN/RNN

"g" "r" "a" "c" "e"

RNN RNN RNN RNN

Learned Style Embeddings

- Instead of explicitly labelling style, can we get the model to learn structure in the audio data organically?
- Feed in the mel spectrogram as an input to a style module at training time
- Compress with conv/lstm to prevent trivial reconstruction
- At inference time feed in a reference mel spectrogram or sample from the latent space
- Can be achieved with token embeddings or a VAE
- Known as a “reference encoder” in the literature

GST Tacotron

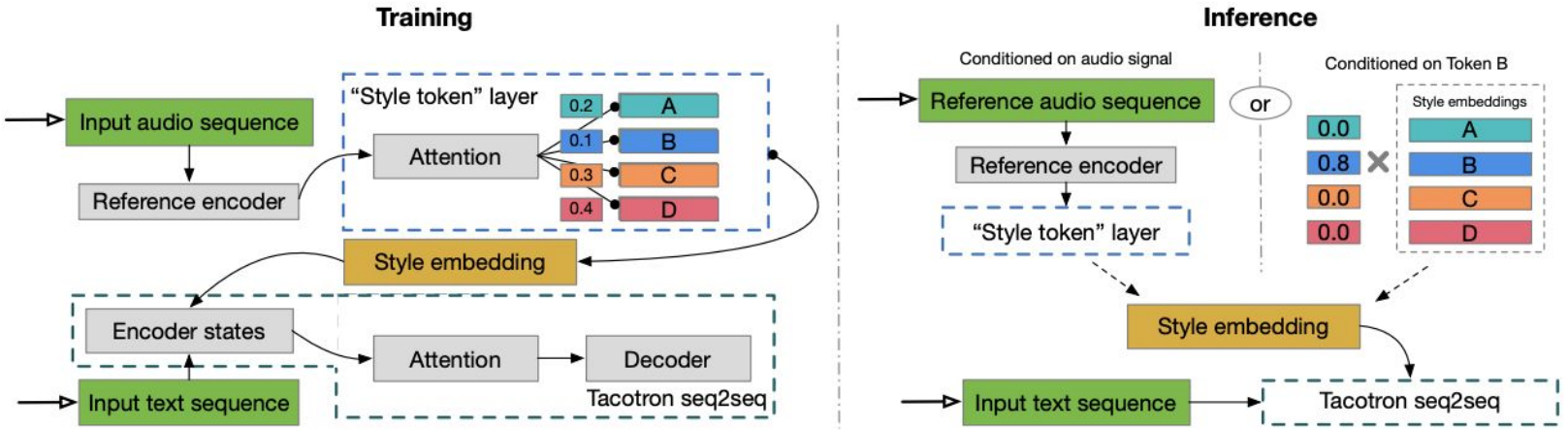


Figure 1. Model diagram. During **training**, the log-mel spectrogram of the training target is fed to the reference encoder followed by a style token layer. The resulting style embedding is used to condition the Tacotron text encoder states. During **inference**, we can feed an arbitrary reference signal to synthesize text with its speaking style. Alternatively, we can remove the reference encoder and directly control synthesis using the learned interpretable tokens.

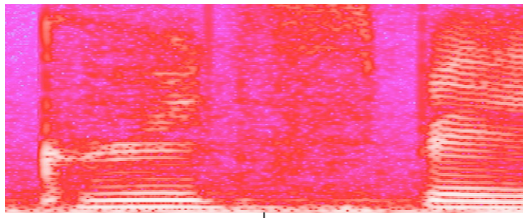
Wang et al. 2018

Samples

VAE Tacotron

- Use variational auto encoder for style latent space
- Latent space then encouraged to follow a gaussian distribution
- Sample prosodies from latent space at inference time
- GMVAE Tacotron uses a hierarchical mixture of gaussians so each component learns a different prosodic component of the data
- Fine-grained VAEs learn the variability in the model's prosody. This can be useful when generating data for semi-supervised ASR

Hsu et al. 2018, Sun et al. 2020



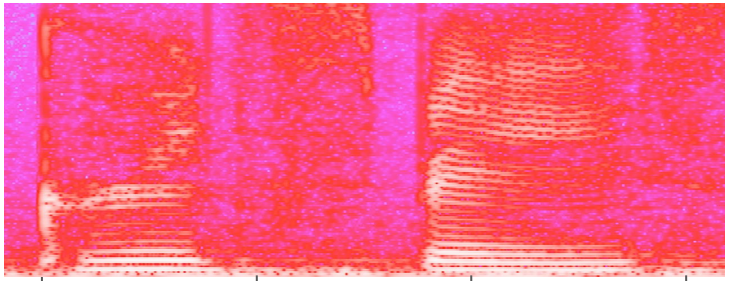
CNN/RNN

VAE Latent Space

$z_t \sim N(0,1)$

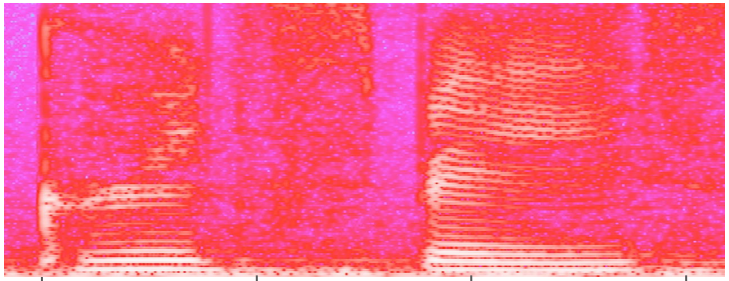
CNN/RNN

"g" "r" "a" "c" "e"



RNN RNN RNN RNN

Training



VAE Latent Space

Sample

$z_t \sim N(0, 1)$

CNN/RNN

"g" "r" "a" "c" "e"

RNN RNN RNN RNN

Inference

End to End Models

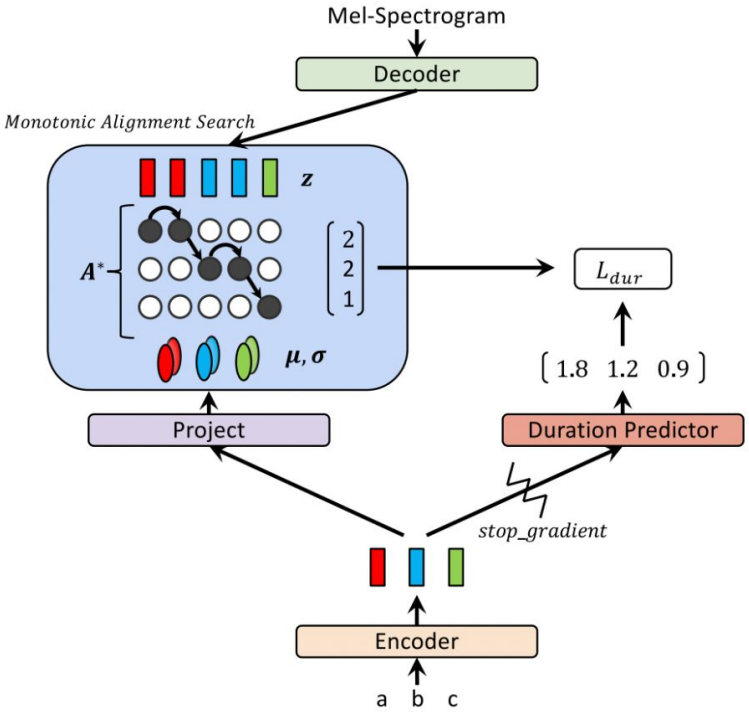
The Ideal TTS Model

- Expressive, flexible duration modeling like Tacotron
- Fast parallel inference like FastSpeech
- Reference encoder to account for one-to-many mapping
- Trained end to end – no separate vocoder

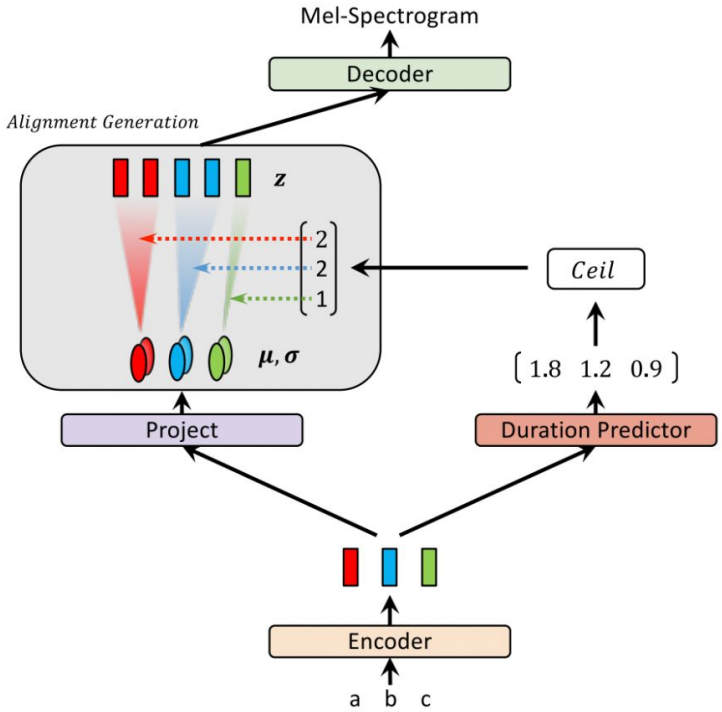
Glow-TTS

- Use a flow model for posterior from mel spectrograms to text
- Use a transformer text encoder to parametrize the prior
- Train using maximum likelihood to match prior and posterior distributions
- Since we have maximum likelihood, use dynamic programming to find the most likely alignment during training
- For inference, train a separate duration predictor to match the most likely alignment

Glow-TTS



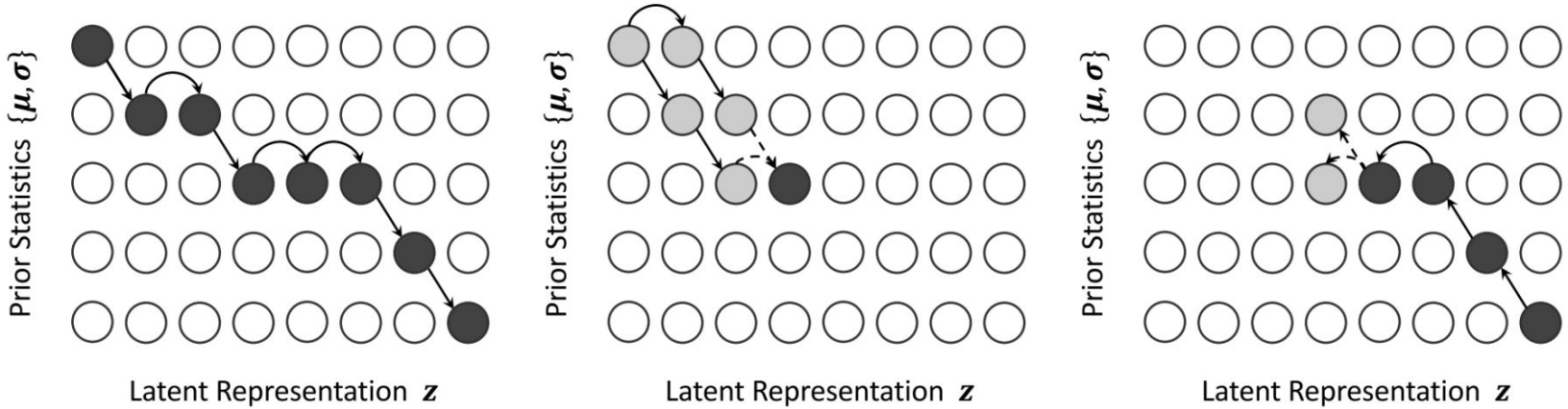
(a) An abstract diagram of the training procedure.



(b) An abstract diagram of the inference procedure.

Kim et al. 2020

Glow-TTS – Monotonic Alignment Search



(a) An example of monotonic alignments

(b) Calculating the maximum log-likelihood Q.

(c) Backtracking the most probable alignment A^*.

Figure 2: Illustrations of the monotonic alignment search.

Kim et al. 2020

Glow-TTS Shortcomings

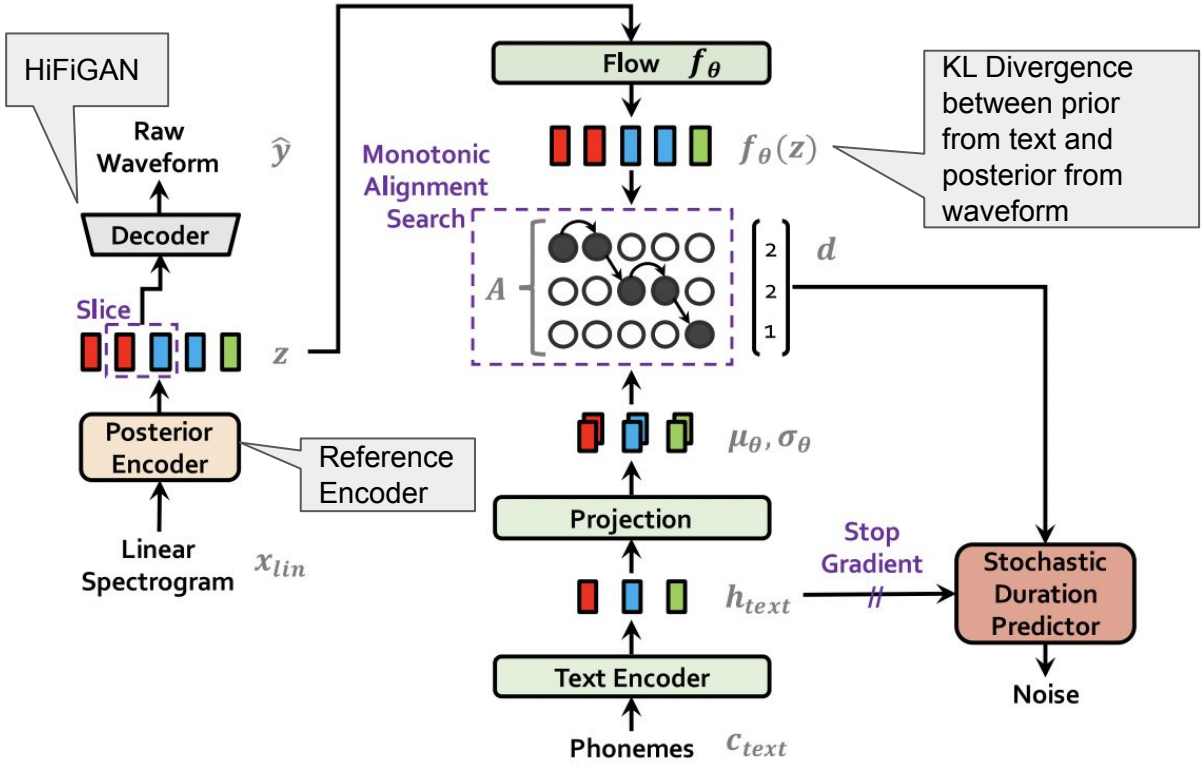
- Not trained end to end – still uses a mel spectrogram output
- No reference encoder – less prosodic variation/controllability
- Direct duration prediction – less natural prosody

VITS

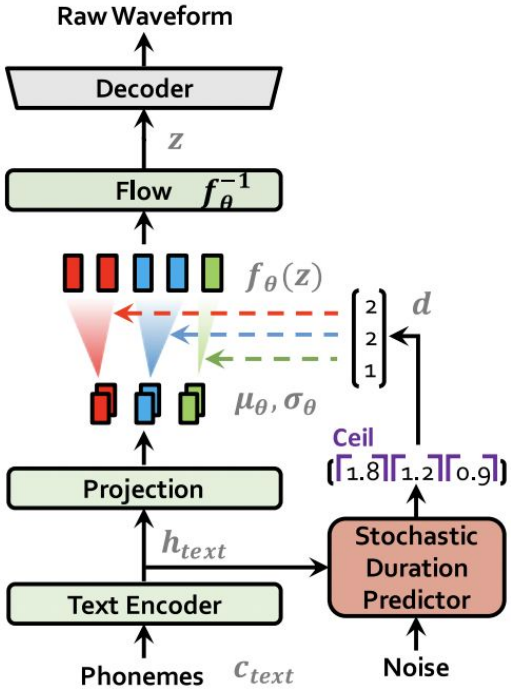
- Glow-TTS style flow model with monotonic alignment search
- Reference encoder with VAE latent space
- Flow model to produce varied duration modeling
- HifiGAN inspired waveform decoder
- Fully end to end training

Kim et al. 2021

VITS



(a) Training procedure



(b) Inference procedure

[Kim et al. 2021]

VITS Loss

$$L_{vae} = L_{recon} + L_{kl} + L_{dur} + L_{adv}(G) + L_{fm}(G)$$

Kim et al. 2021

VITS

- Combines the best features of flows, VAEs and GANs
- End to end training
- No alignments required
- Controllable prosody through VAE
- Fast parallel inference: 67 RTF on 1 V100
- Very high MOS scores

NaturalSpeech

- Similar structure to VITS with prior/posterior flow model
- Adds phoneme pretraining, differential duration modeling and a memory VAE
- Matches MOS of human speaker on LJSpeech dataset

Samples

Tan et al. 2022

Generative Spoken Language Modeling

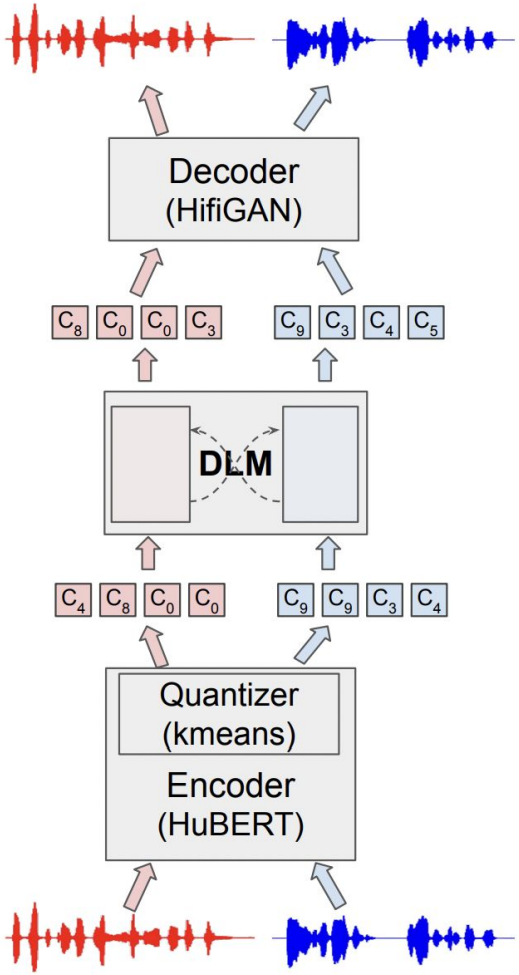
The Future of TTS

- On high quality datasets, TTS has reached parity with humans in MOS
- These systems operate on a single utterance at a time
- Systems that handle long form context and dynamically adjust their tone are the future

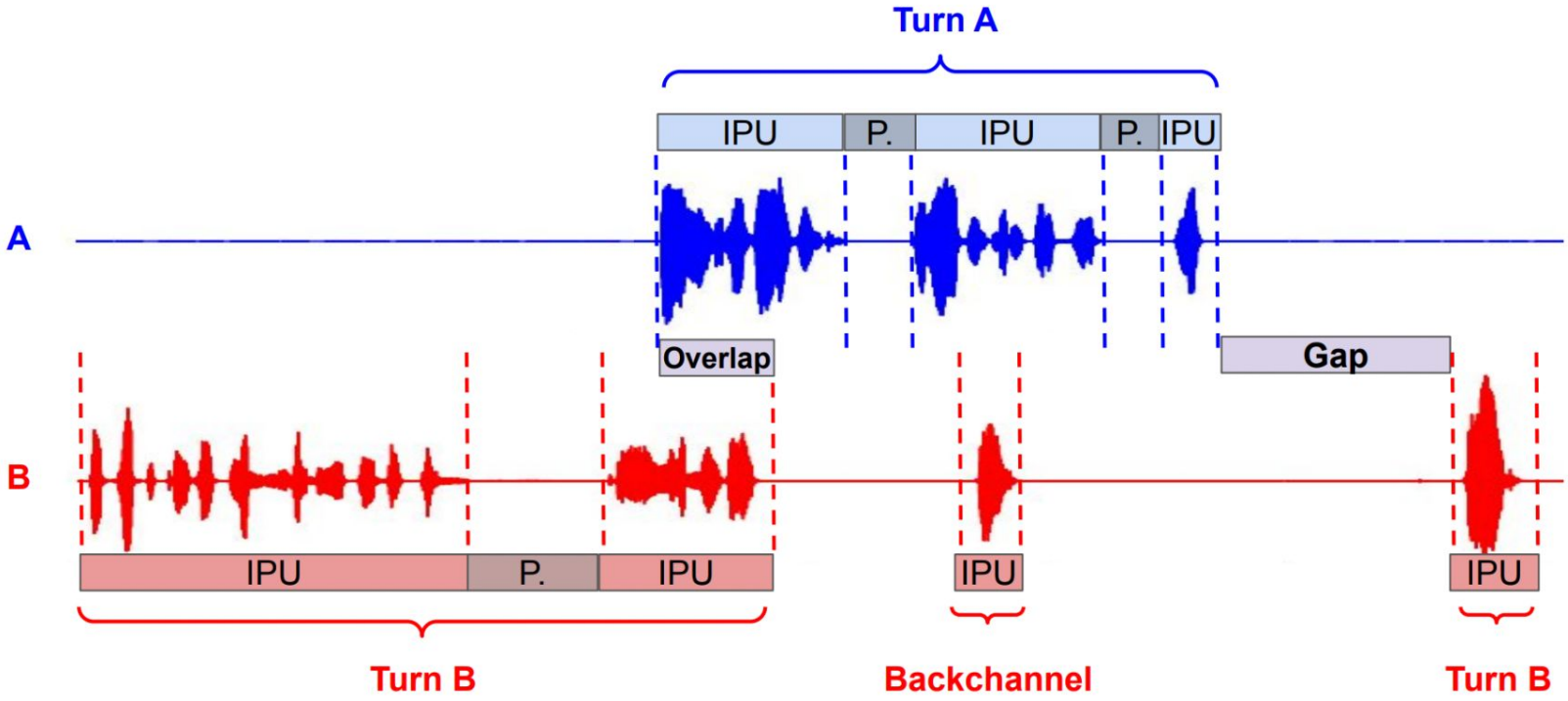
Generative Spoken Language Modeling

- Obtain discrete audio codes from Wav2Vec-2, HUBERT etc.
- Train a GPT style transformer LM on the codes
- Train a speech synthesis model to convert codes to speech
- Can simulate turn-taking and backchannels when training on two channels

Samples



Nguyen et al 2022



Nguyen et al 2022

Conclusion

Ethical TTS

- Modern TTS is a powerful tool
- People have and will continue to be fooled by great TTS
- Only synthesize someone’s voice with permission
- Disclose that your dialog system is a bot

Thank You!

Email: alex@gridspace.com