

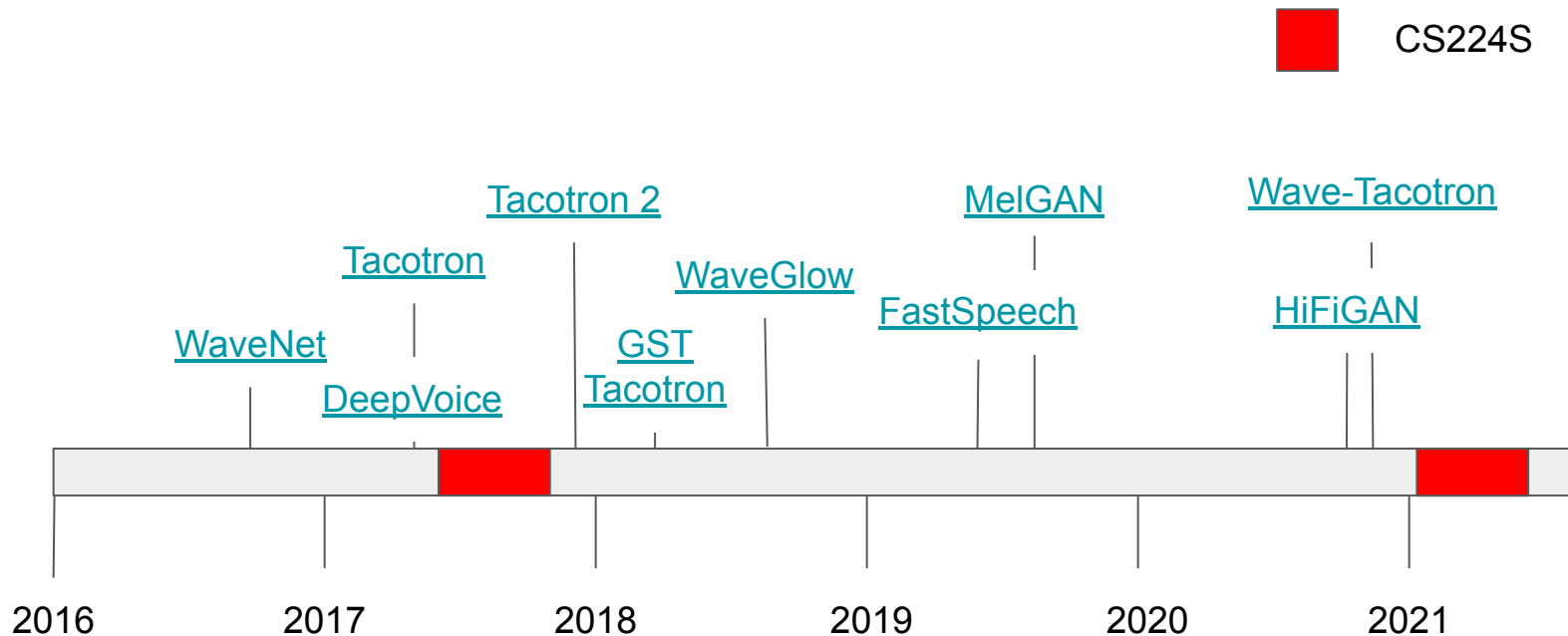


CS224S

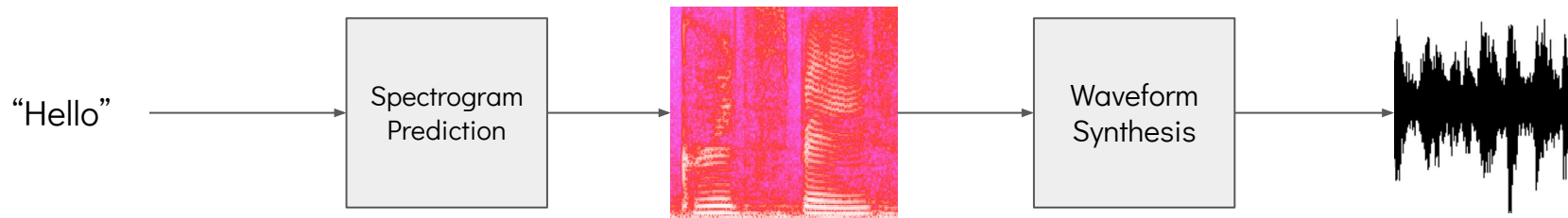
Lecture 14: Deep Learning for TTS

Alex Barron
Stanford University
Winter 2021

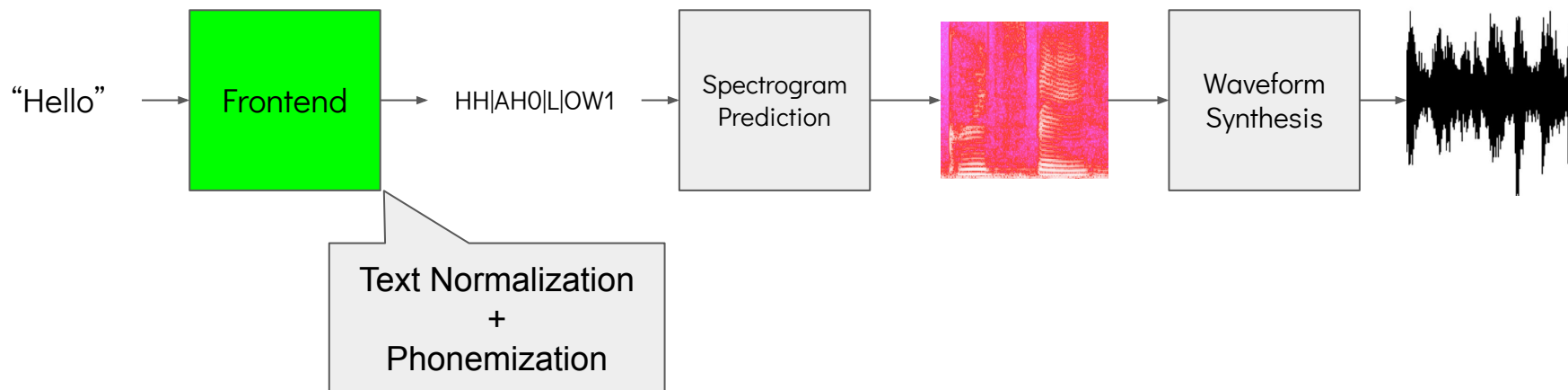
Rapid Progress in TTS in the last few years



“End to End” TTS Paradigm



“End to End” TTS Paradigm



Why “End to End” 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 concatenative TTS

Why “End to End” 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?

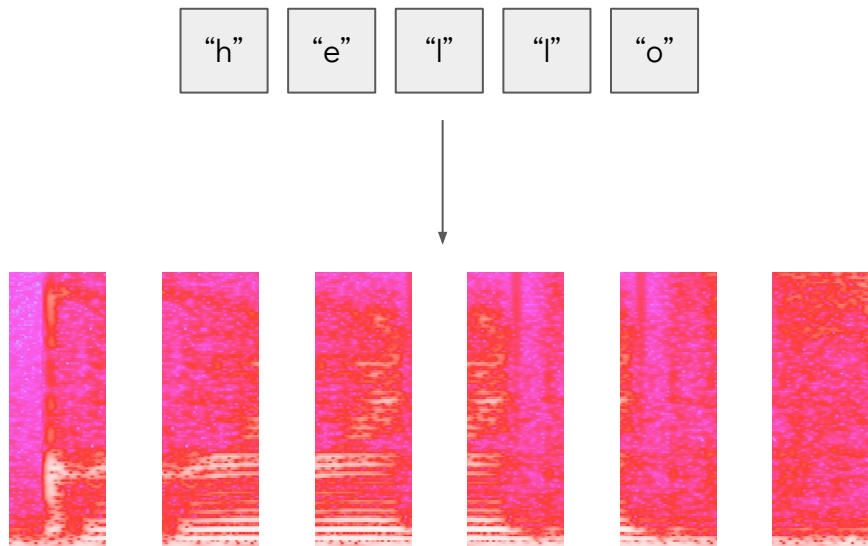
- Prosodic/phonemic aspects of speech can be modelled without phase information
- Allows focus on human speech frequency bands with mel filters
- STFT chunks speech into frames of a useful duration for phoneme and prosody modeling
- Fast to generate thanks to FFT
- Separate model can be used to fill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

Overview of Model Types

1. Text to Spectrogram Models
 - a. Attention-based (e.g. Tacotron)
 - b. Duration-based (e.g. FastSpeech)
2. Spectrogram to Waveform Models
 - a. Autoregressive (e.g. WaveNet, WaveRNN)
 - b. Flows (e.g. WaveGlow, Parallel WaveNet)
 - c. GANs (e.g. MelGAN, Parallel WaveGAN)
3. Speaker and Style Embeddings
 - a. One hot encoded labels
 - b. Learned speaker embeddings
 - c. Learned style embeddings

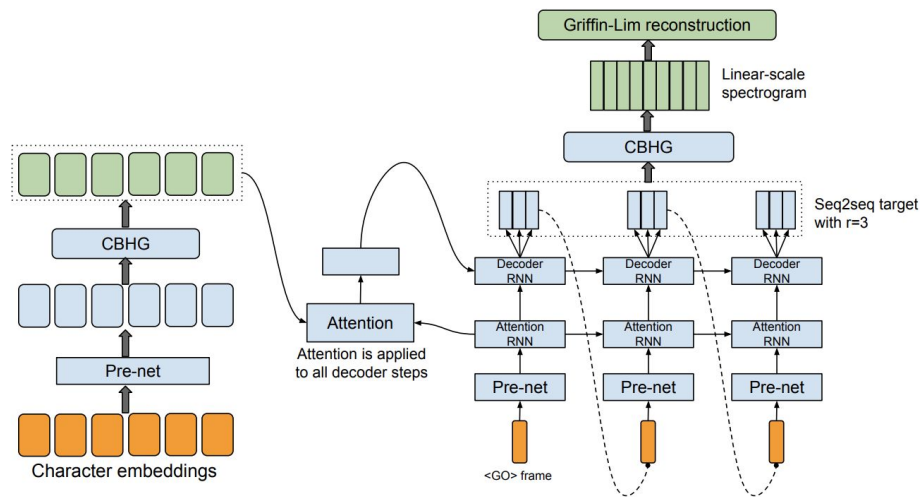
Text to Spectrogram Models

Sequence to Sequence Problem



Tacotron

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



Tacotron

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

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

$$\boldsymbol{\alpha}_i = \text{Attention}(\mathbf{s}_i, \dots) \quad \mathbf{c}_i = \sum_j \alpha_{i,j} \mathbf{h}_j \quad (3)$$

$$\mathbf{d}_i = \text{RNN}_{\text{Dec}}(\mathbf{d}_{i-1}, \mathbf{c}_i, \mathbf{s}_i) \quad \mathbf{y}_i = f_o(\mathbf{d}_i) \quad (4)$$

$$\{h_j\}_{j=1}^L = \text{Encoder}(\{x_j\}_{j=1}^L)$$

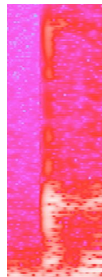
“g” “r” “a” “c” “e”

Optionally takes previous alignment,
encoder states

$$\alpha_i = \text{Attention}(s_i, \dots)$$

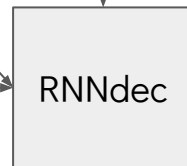
$$c_i = \sum_j \alpha_{i,j} h_j$$

y_{i-1}

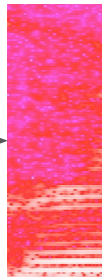


s_{i-1}

c_{i-1}



y_i



$$s_i = \text{RNN}_{\text{Att}}(s_{i-1}, c_{i-1}, y_{i-1})$$

$$d_i = \text{RNN}_{\text{Dec}}(d_{i-1}, c_i, s_i)$$

Can be 1-5 mel frames (reduction
factor)

$$i = 0$$

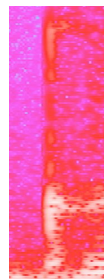
$$\{h_j\}_{j=1}^L = \text{Encoder}(\{x_j\}_{j=1}^L)$$

“g” “r” “a” “c” “e”

$$\alpha_i = \text{Attention}(s_i, \dots)$$

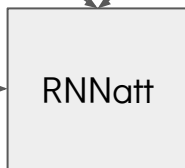
$$c_i = \sum_j \alpha_{i,j} h_j$$

y_{i-1}

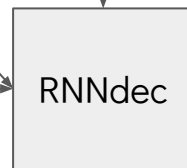


s_{i-1}

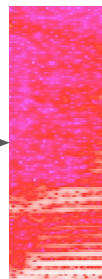
c_{i-1}



d_{i-1}



y_i

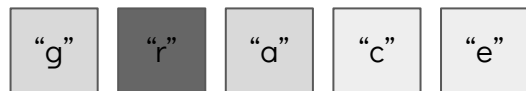


$$s_i = \text{RNN}_{\text{Att}}(s_{i-1}, c_{i-1}, y_{i-1})$$

$$d_i = \text{RNN}_{\text{Dec}}(d_{i-1}, c_i, s_i)$$

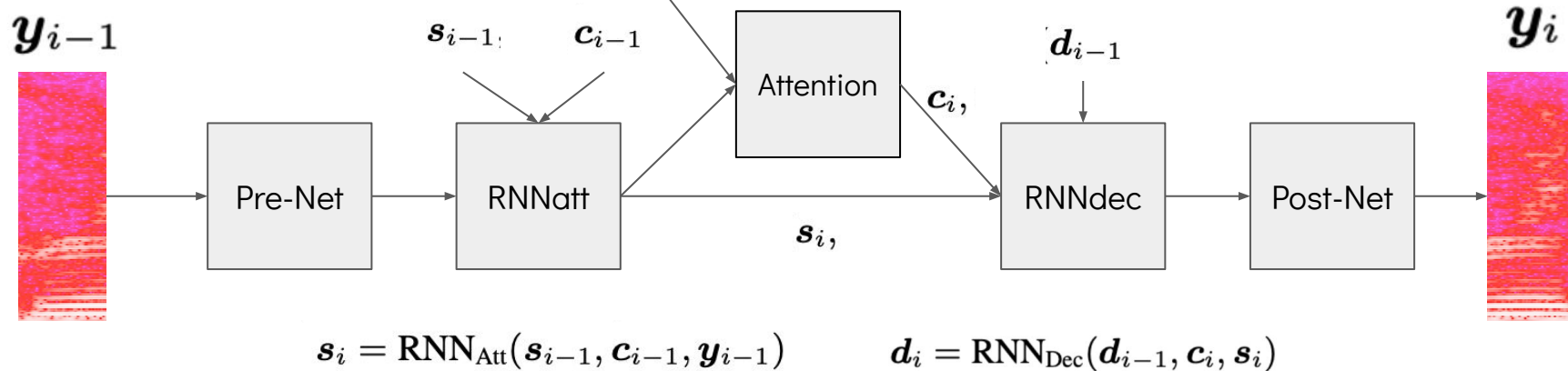
$i = 1$

$$\{h_j\}_{j=1}^L = \text{Encoder}(\{x_j\}_{j=1}^L)$$



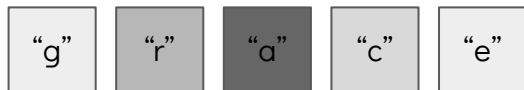
$$\alpha_i = \text{Attention}(s_i, \dots)$$

$$c_i = \sum_j \alpha_{i,j} h_j$$



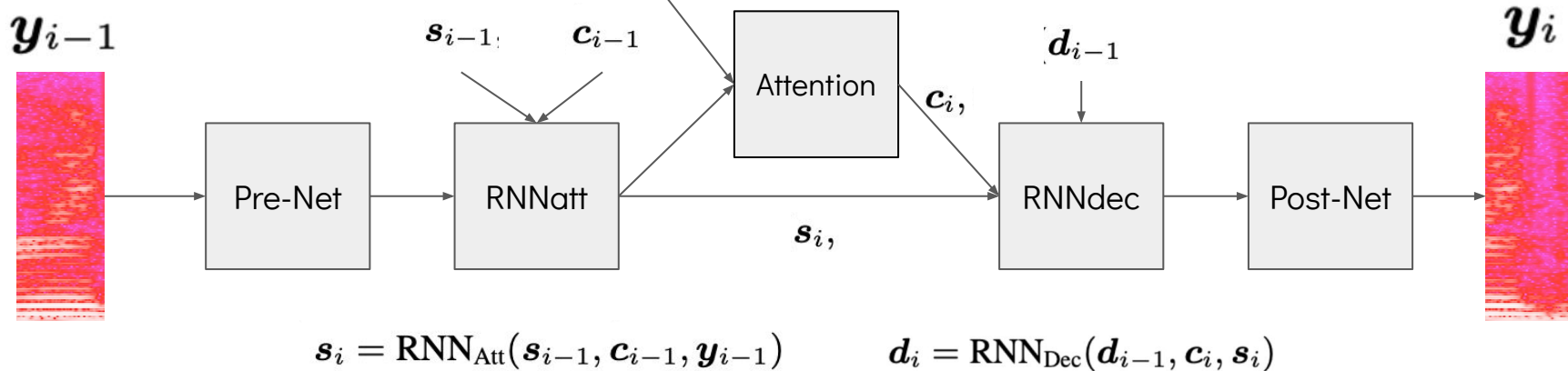
$i = 2$

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



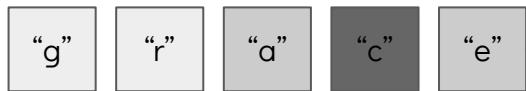
$$\alpha_i = \text{Attention}(\mathbf{s}_i, \dots)$$

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



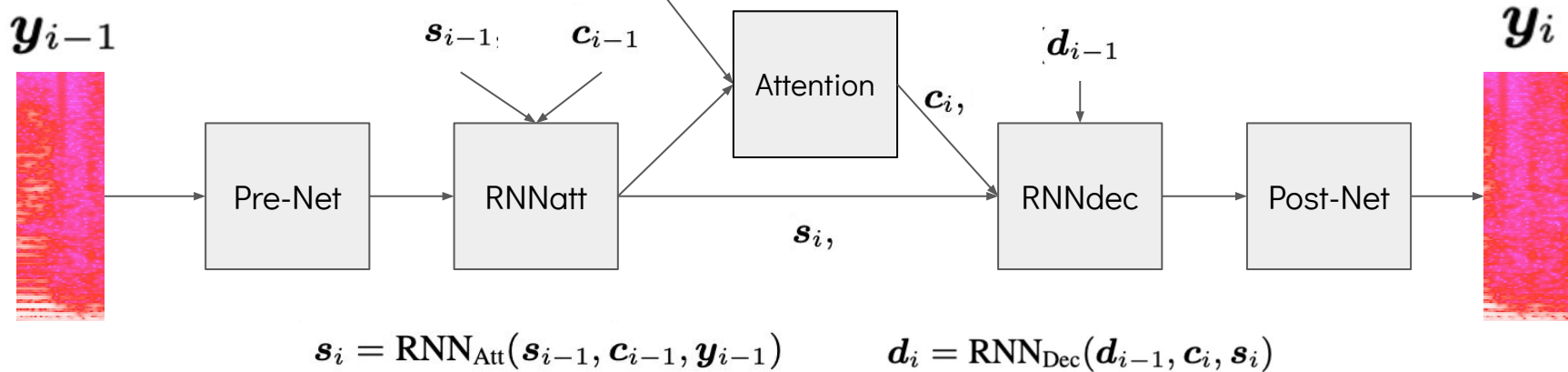
$i = 3$

$$\{h_j\}_{j=1}^L = \text{Encoder}(\{x_j\}_{j=1}^L)$$



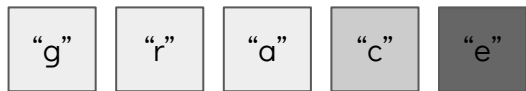
$$\alpha_i = \text{Attention}(s_i, \dots)$$

$$c_i = \sum_j \alpha_{i,j} h_j$$



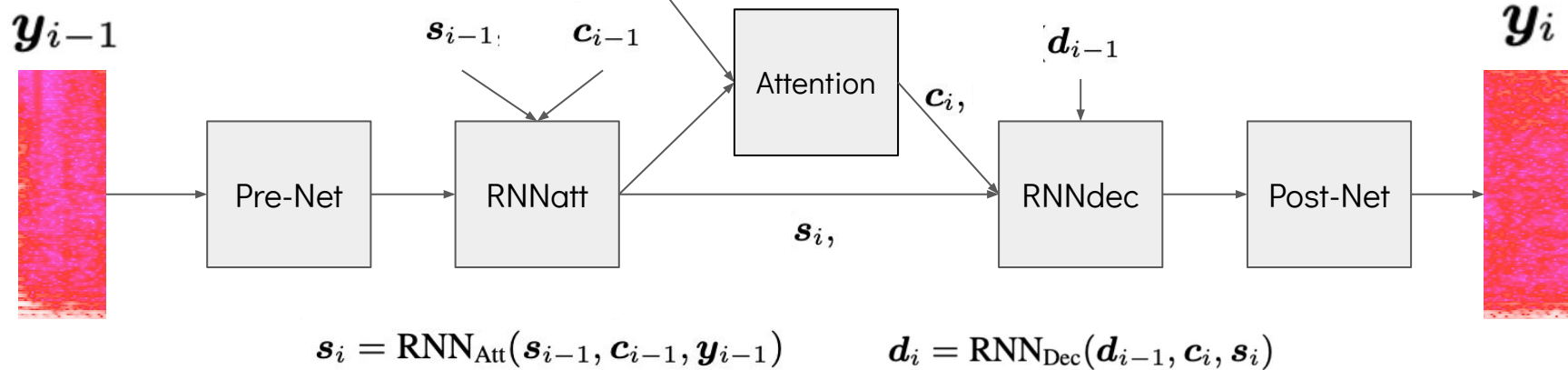
$i = 4$

$$\{h_j\}_{j=1}^L = \text{Encoder}(\{x_j\}_{j=1}^L)$$

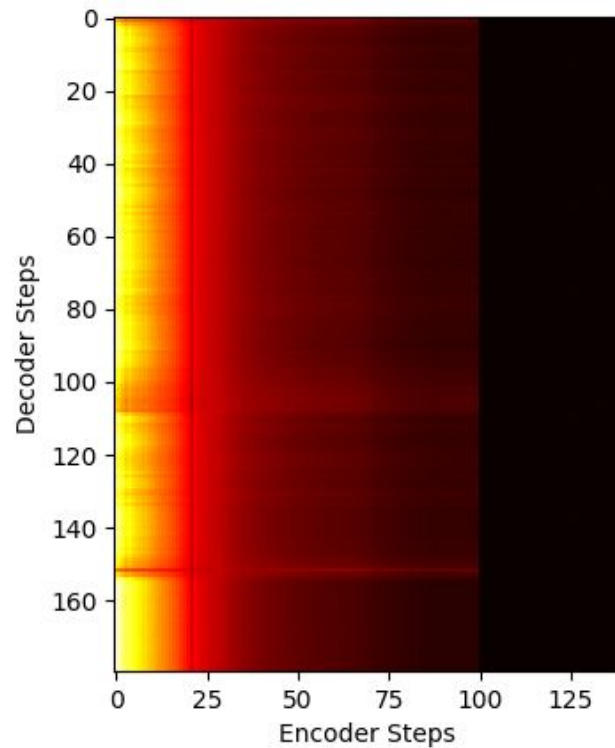


$$\alpha_i = \text{Attention}(s_i, \dots)$$

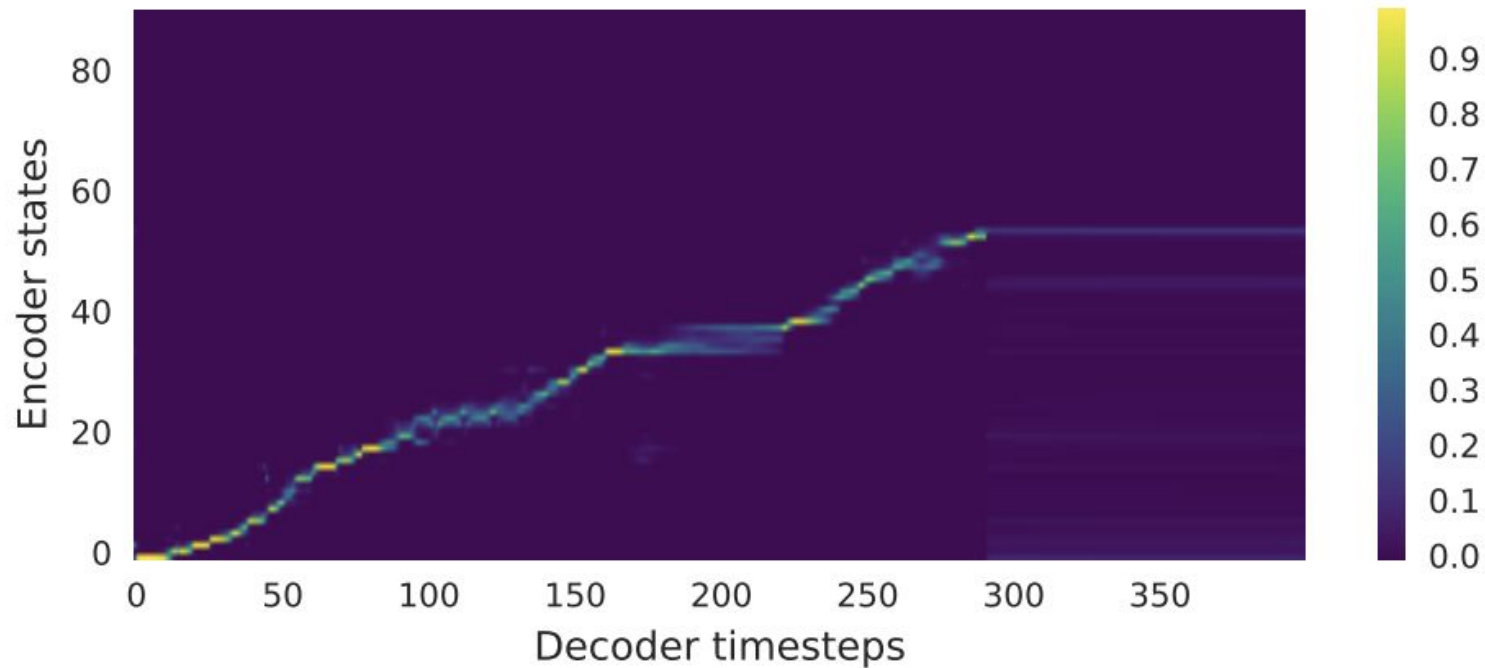
$$c_i = \sum_j \alpha_{i,j} h_j$$



Learning an Alignment



Attention can be fickle



Many Forms of Attention

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

Content Based Attention

RNN_att
Query

Encoder Keys

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

$$\alpha_i = \text{softmax}(e_i)$$

Monotonic Attention

- Replace softmax with “monotonic” probability function
- Mask all states in the softmax before the previously attended one
- Easier to get an alignment in training, but typically reduces maximum quality
- Can be useful at inference time to prevent losses of alignment

$$e_{i,j} = v^T \tanh(Ws_i + Vh_j)$$

$$\alpha_i = \textit{monotonic}(e_i, \alpha_{i-1})$$

Location Sensitive Attention

Convolution
with previous
alignment

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

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

$$\alpha_i = \text{softmax}(e_i)$$

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

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

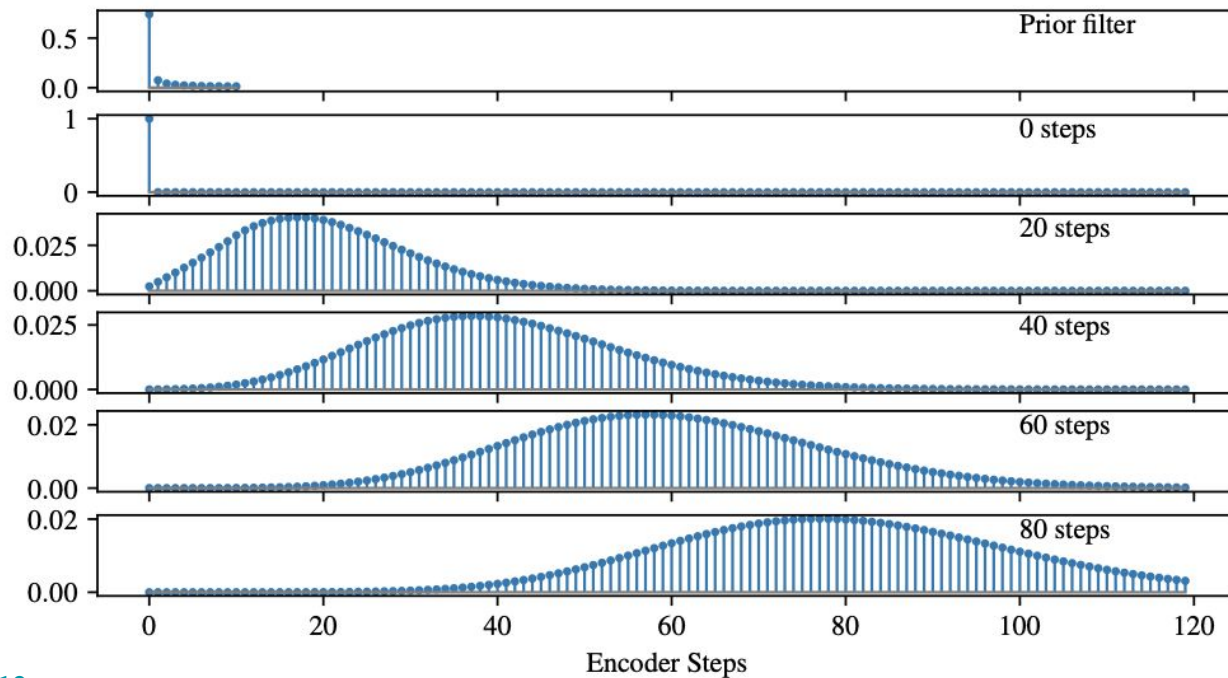
$$p_i = \log(P * \alpha_{i-1})$$

Prior bias to
encourage
monotonicity

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

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

Initial Alignment Via Repeated Application of Prior Filter



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

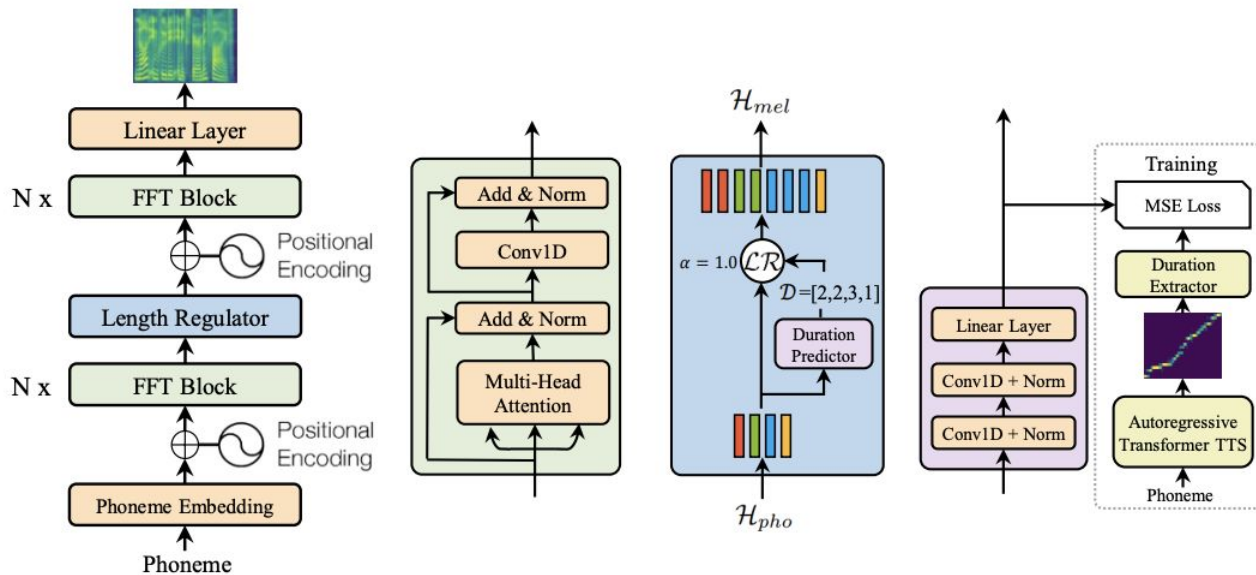
Tips for training attention TTS models

- 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



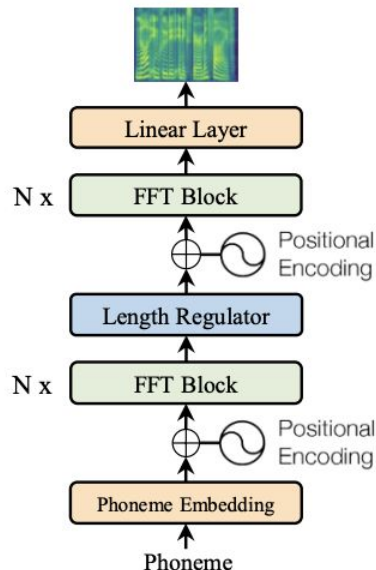
FastSpeech 1/2

- Similar to earlier DNN TTS systems
- Explicitly predict phoneme durations, f_0 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

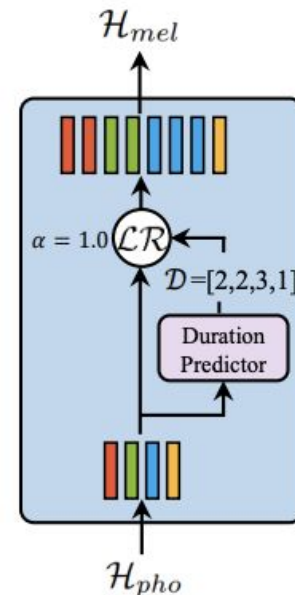
FastSpeech Architecture

Encoder and Decoder are both fully parallel transformer blocks

(FFT = Feed Forward Transformer not Fast Fourier Transform)

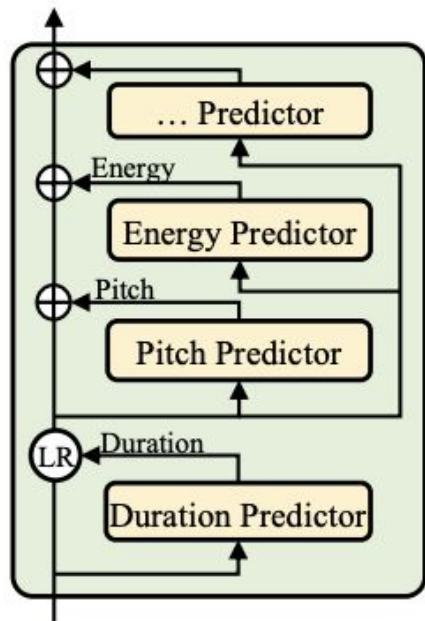


Upsample by repeating encoder states by the predicted duration in frames

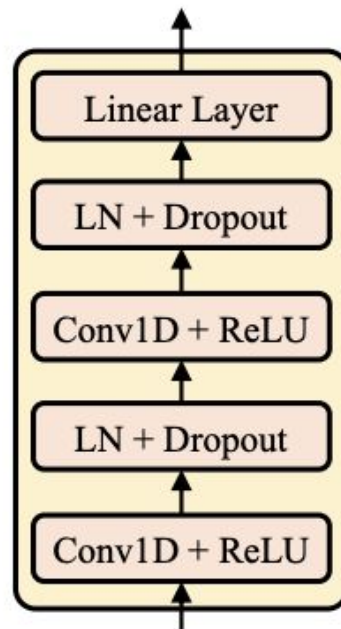


FastSpeech 2 Variance Predictors

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



Variance Predictor Structure:



Attention vs Duration based models

Attention-based

- No alignments needed
- Adaptable to diverse, noisy datasets
- Slightly higher maximum quality

Duration-based

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

Non-Attentive Tacotron -- a compromise?

- Explicitly predicts duration but uses an autoregressive decoder
- Gaussian upsampling to allow each decoder step access to multiple encoder states
- Unsupervised duration modelling with a VAE for noisy datasets where HMM alignments are difficult to obtain

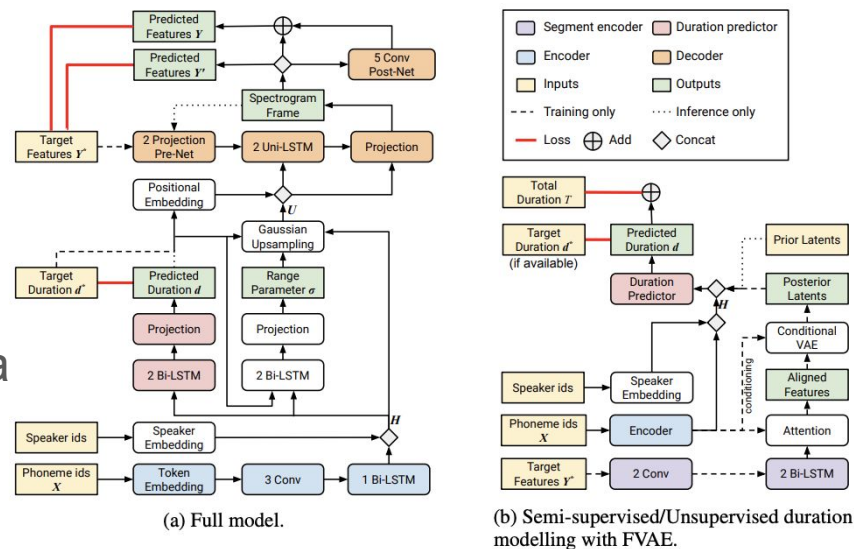


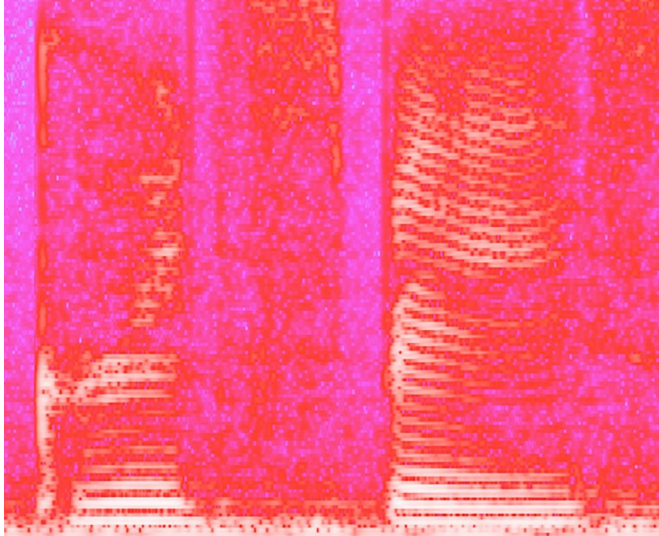
Figure 1: Architecture of Non-Attentive Tacotron.

Summary

- If you have a professionally recorded dataset with good forced alignments available, a FastSpeech style model will have very fast inference and be easy to train
- Varied datasets with conversational speech are more amenable to Tacotron style models which can learn their own alignments

Spectrogram to Waveform Models

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

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

```
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
```

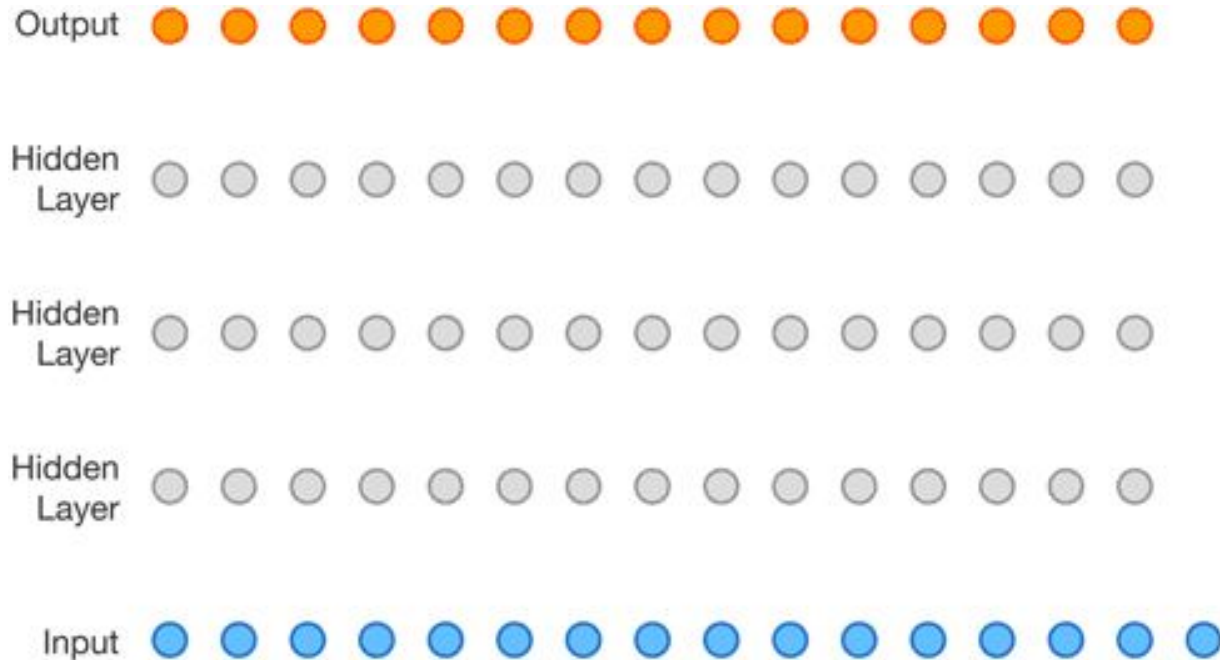
Griffin-Lim Limitations

- 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

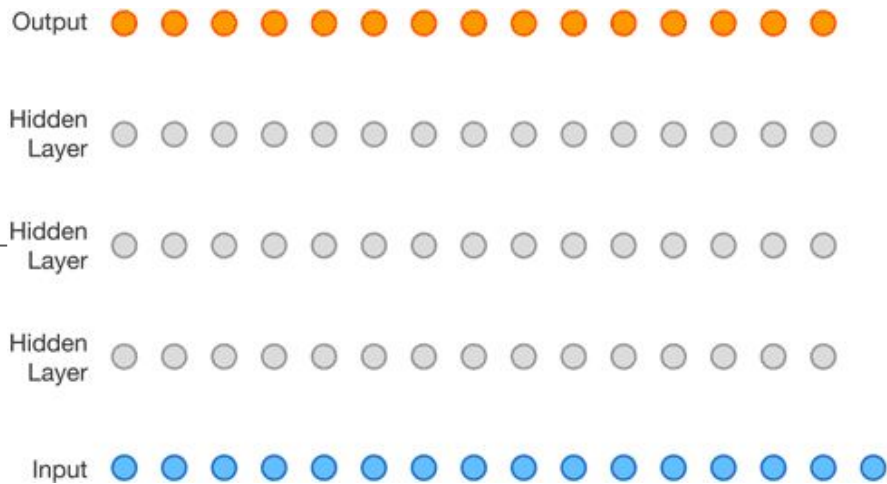
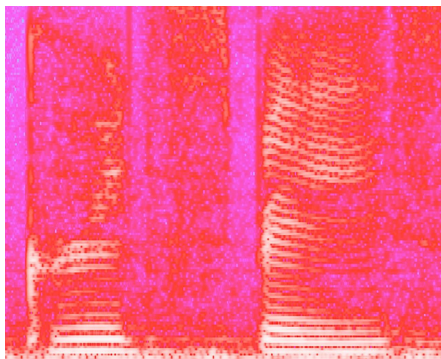
- Extremely long time series in the output
- Endless training data
- A test piece for modern generative models

WaveNet



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

WaveNet



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

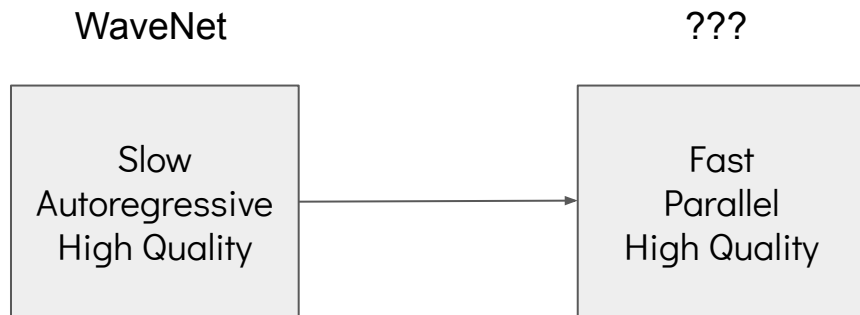
WaveNet

- 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

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)

Parallelizing WaveNet



Parallelizing WaveNet

WaveNet

Parallel Training
Sequential Inference

???

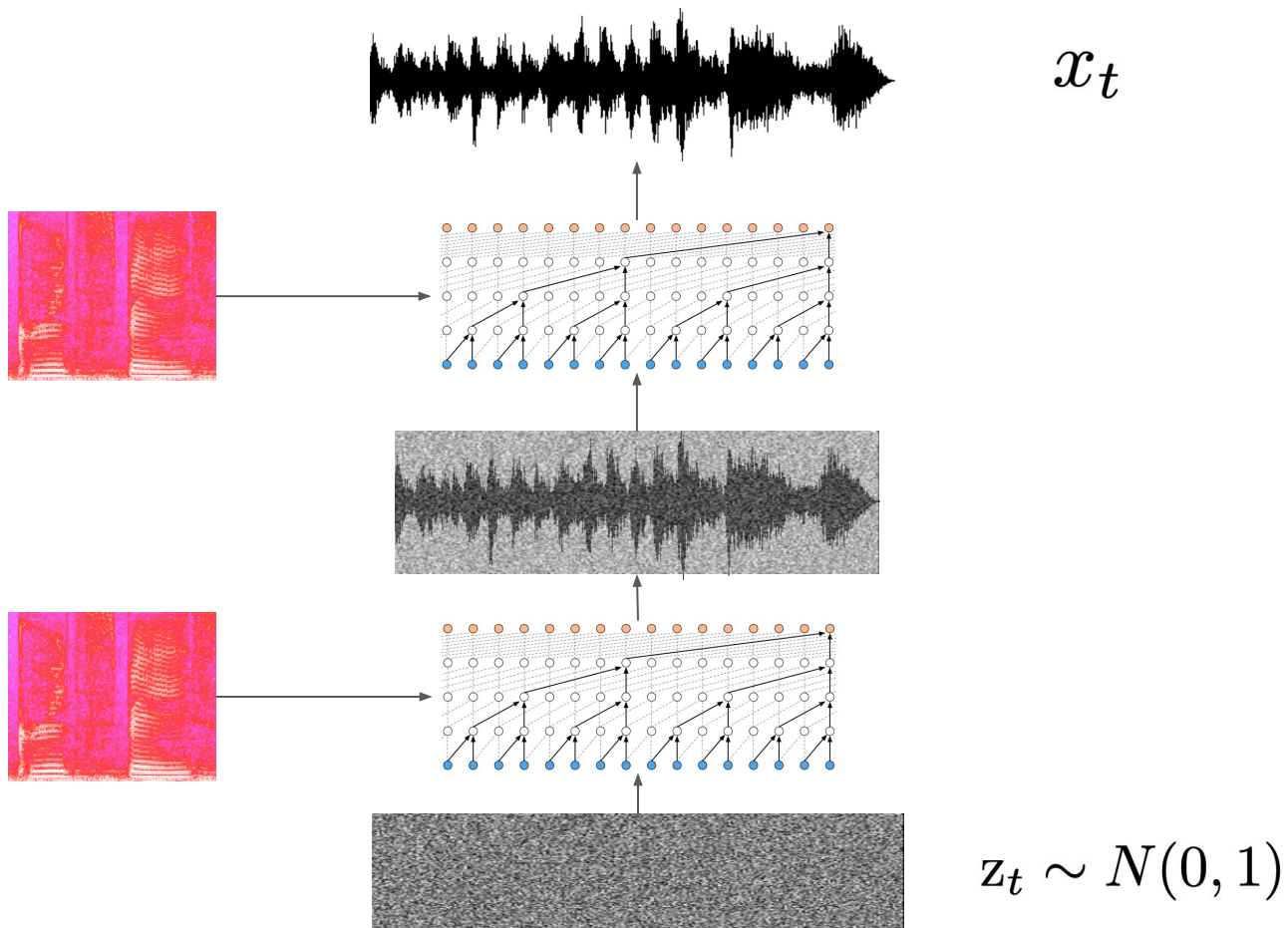
Sequential Training
Parallel Inference

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:

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

where s and μ are produced by running a WaveNet on \mathbf{z}



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

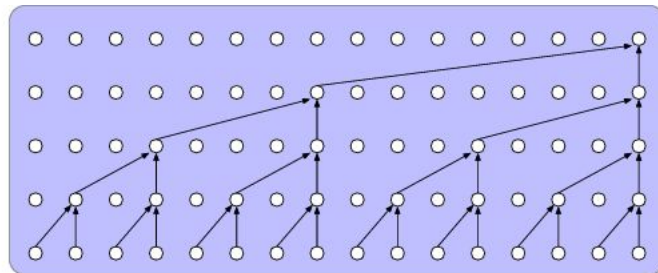
Parallel WaveNet: Student and Teacher

- 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

WaveNet Teacher

Linguistic features \dashrightarrow

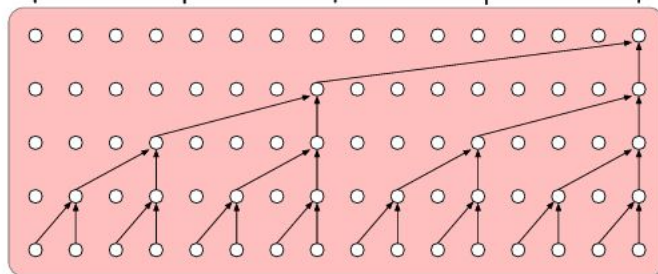


Teacher Output

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

WaveNet Student

Linguistic features \dashrightarrow



Generated Samples

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

Student Output

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

Input noise

$$z_i$$

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

WaveNet

Parallel Training
Sequential Inference

IAF

Sequential Training
Parallel Inference

Parallelizing WaveNet

WaveNet

Parallel Training
Sequential Inference

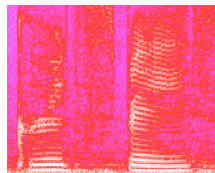
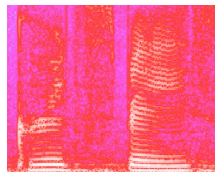
IAF

Sequential Training
Parallel Inference

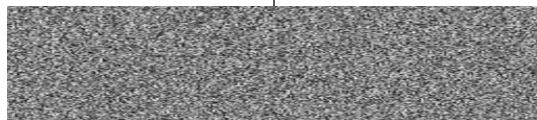
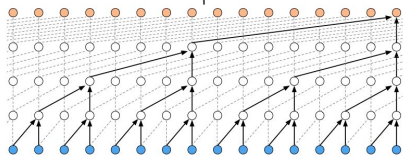
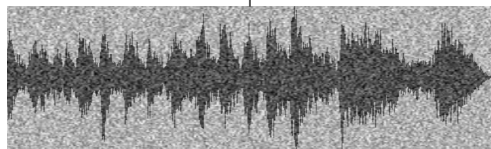
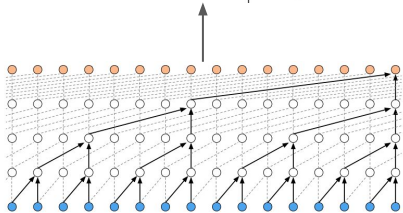
???

Parallel Training
Parallel Inference

What if this was
invertible?

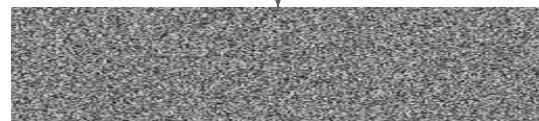
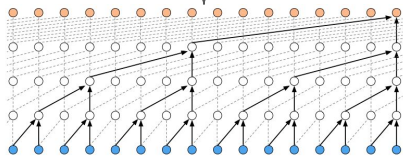
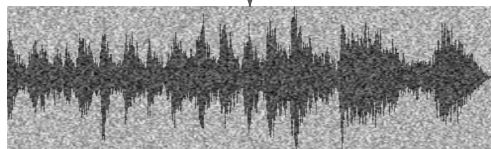
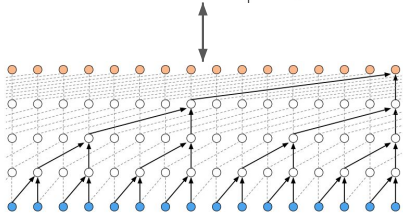
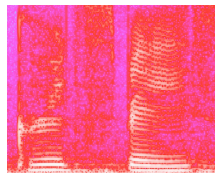
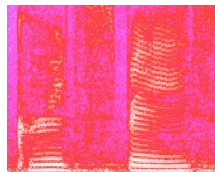


x_t



$z_t \sim N(0, 1)$

What if this was
invertible?



x_t

Inference:
Sample z and
transform to x

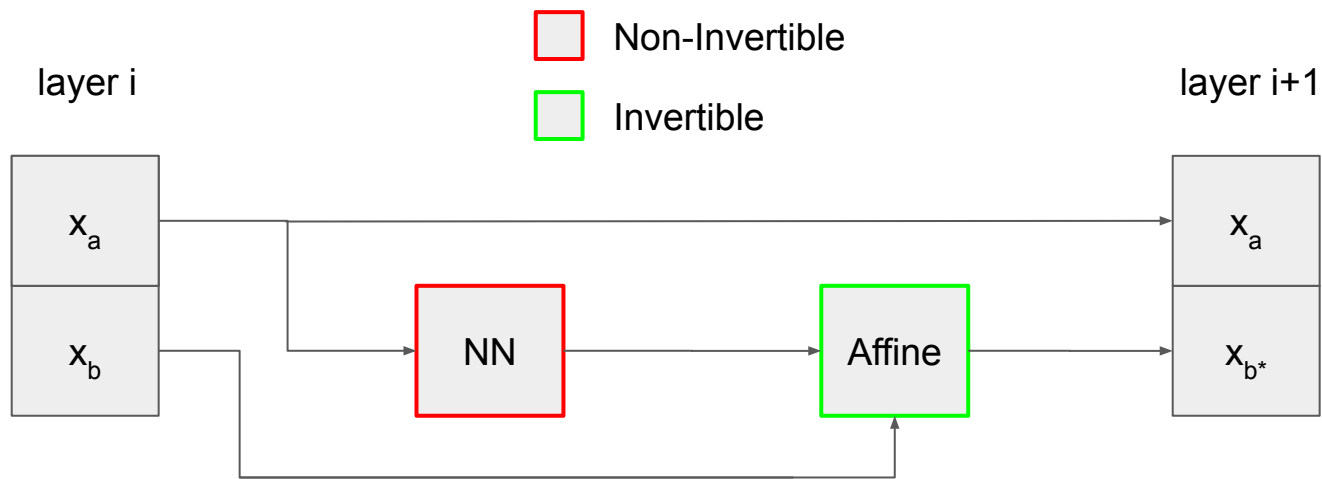
Training:
Transform x to z and
enforce a normal
distribution on z

$z_t \sim N(0, 1)$

WaveGlow

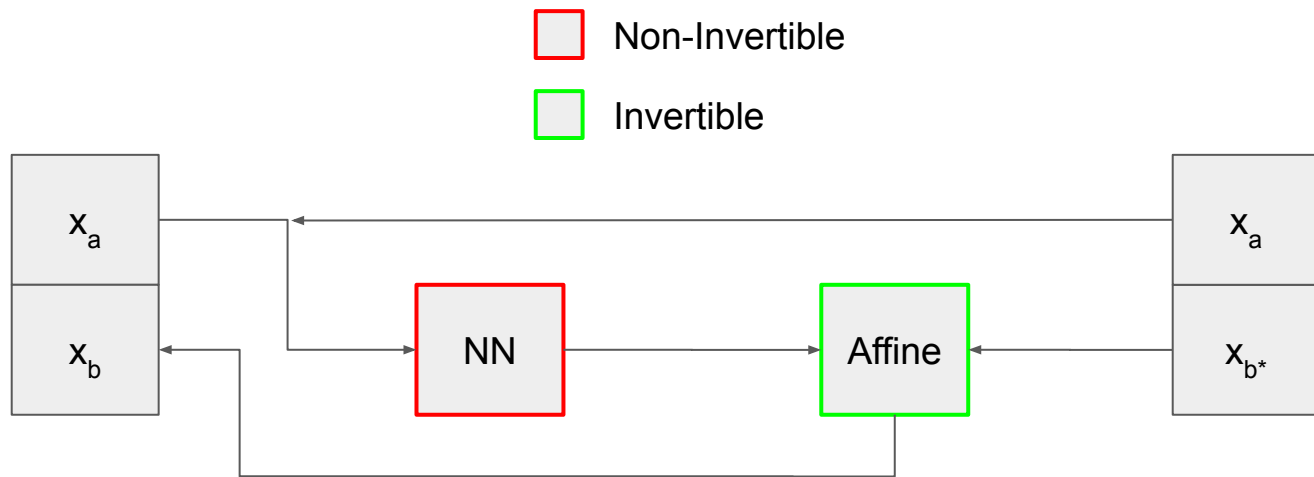
- Invertible flow based model
- Originally applied to image generation by OpenAI
- Quickly repurposed the architecture for audio generation

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

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

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

$$\mathbf{x}_{b'} = \mathbf{s} \odot \mathbf{x}_b + \mathbf{t}$$

$$\mathbf{f}_{\textit{coupling}}^{-1}(\mathbf{x}) = \textit{concat}(\mathbf{x}_a, \mathbf{x}_{b'})$$

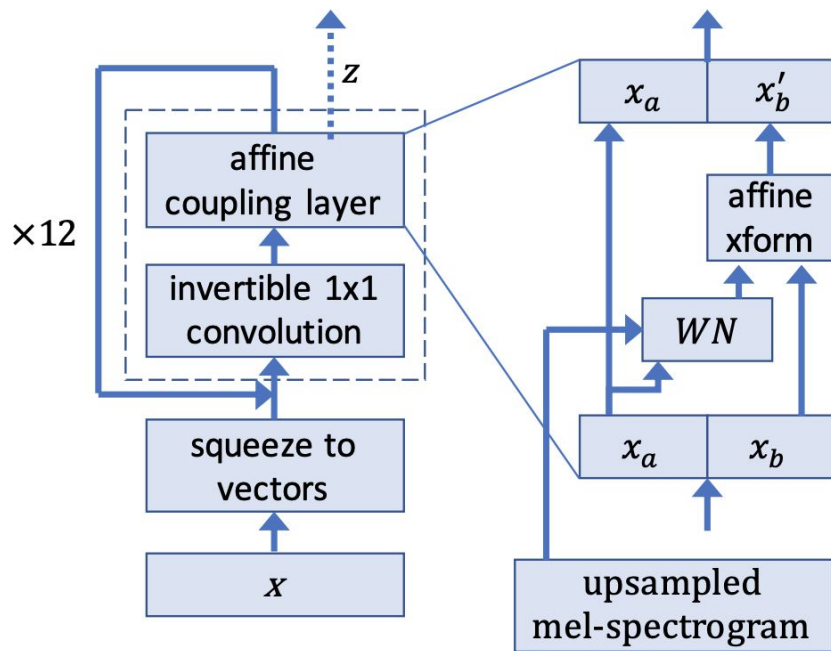
Mixing Channels

- 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



WaveGlow Loss Function

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

Fit \mathbf{z} to a unit
Gaussian
Distribution

Change of
variables from
coupling

$$+ \sum_{j=0}^{\#coupling} \log s_j(\mathbf{x}, mel-spectrogram)$$

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

Ensure 1x1 conv
kernels remain
invertible

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

Parallel Training
Sequential Inference

IAF

Sequential Training
Parallel Inference

WaveGlow

Parallel Training
Parallel Inference

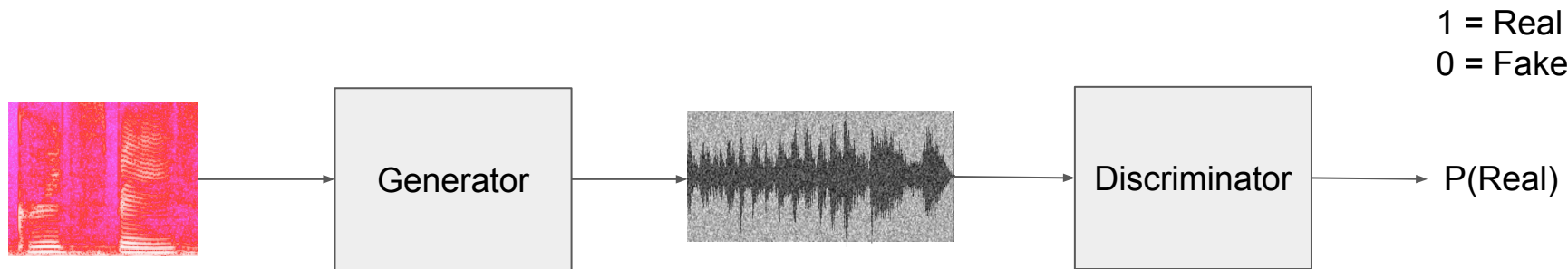
Can we go faster?

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

GAN-based Spectrogram->Waveform Models

- 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

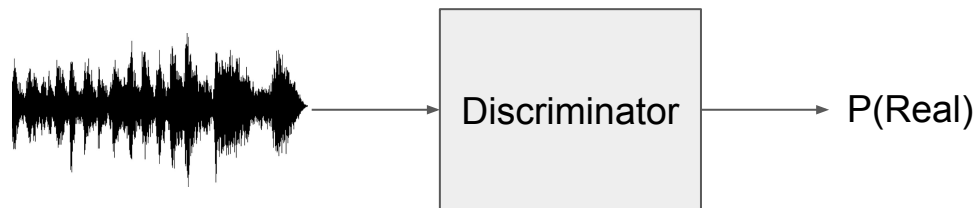


Generator Loss

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

Discriminator Loss

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



Additional Losses for Audio GANs

- Direct reconstruction loss on mel spectrograms

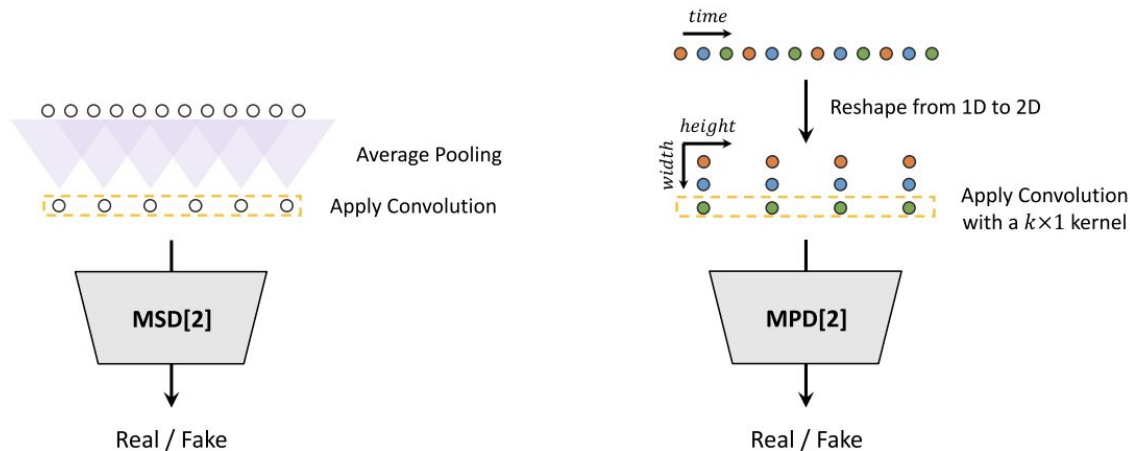
$$\mathcal{L}_{Mel}(G) = \mathbb{E}_{(x,s)} \left[||\phi(x) - \phi(G(s))||_1 \right]$$

- Discriminator feature map L1 loss

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

Multi-scale/multi-period Discriminators

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



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
- WaveRNN is great for embedded systems but high engineering cost to optimize
- WaveGlow based vocoders are slightly less complicated to train and perform well on GPUs

True End to End Models

- Full text to waveform models are still in their infancy
- [ClariNet](#), [FastSpeech 2](#) and [WaveTacotron](#) are examples

Speaker and Style Embeddings

Expanding the “text” in TTS

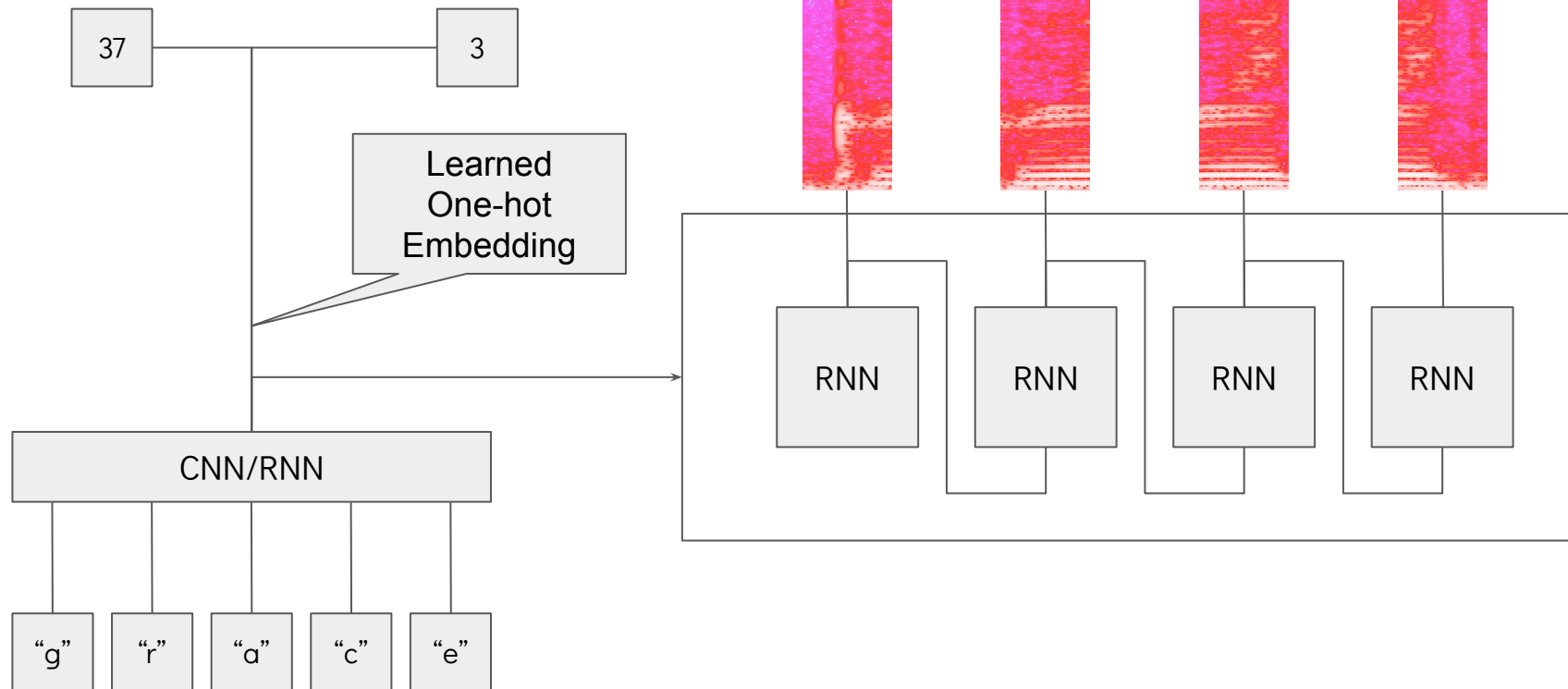
- TTS is fundamentally a one-to-many mapping
- The same text has infinitely many voicings
- Controllable speaker and prosody is very useful in dialog systems and elsewhere

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

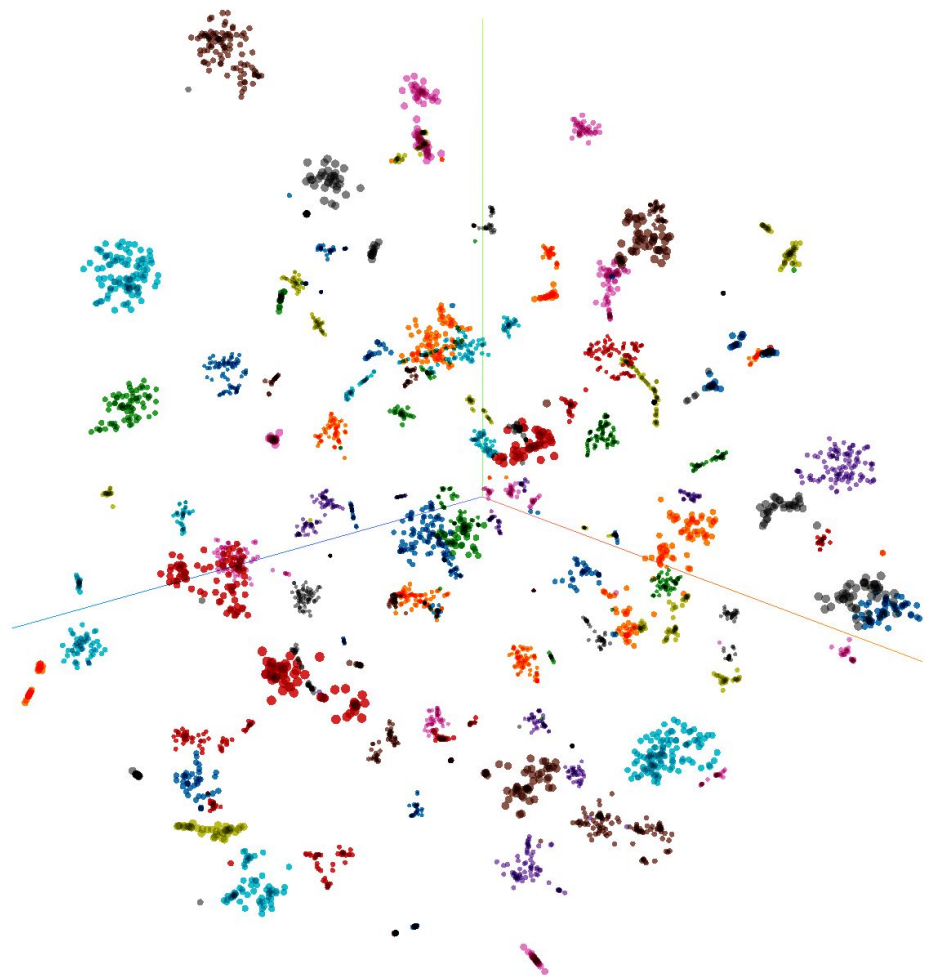
Speaker ID

Prosodic Style



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](#)



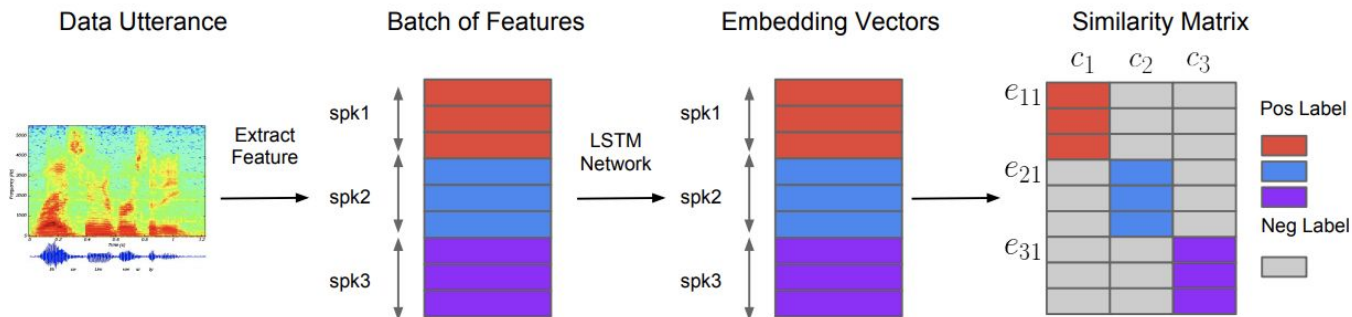


Fig. 1. System overview. Different colors indicate utterances/embeddings from different speakers.

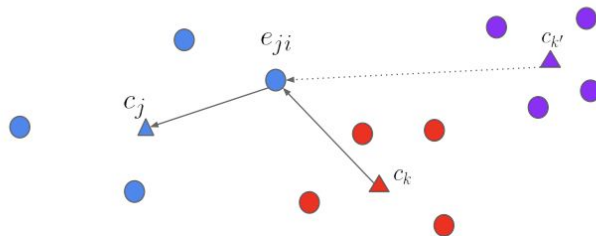
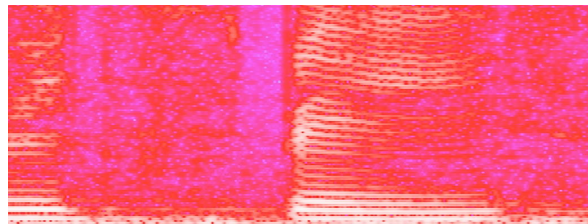


Fig. 2. GE2E loss pushes the embedding towards the centroid of the true speaker, and away from the centroid of the most similar different speaker.

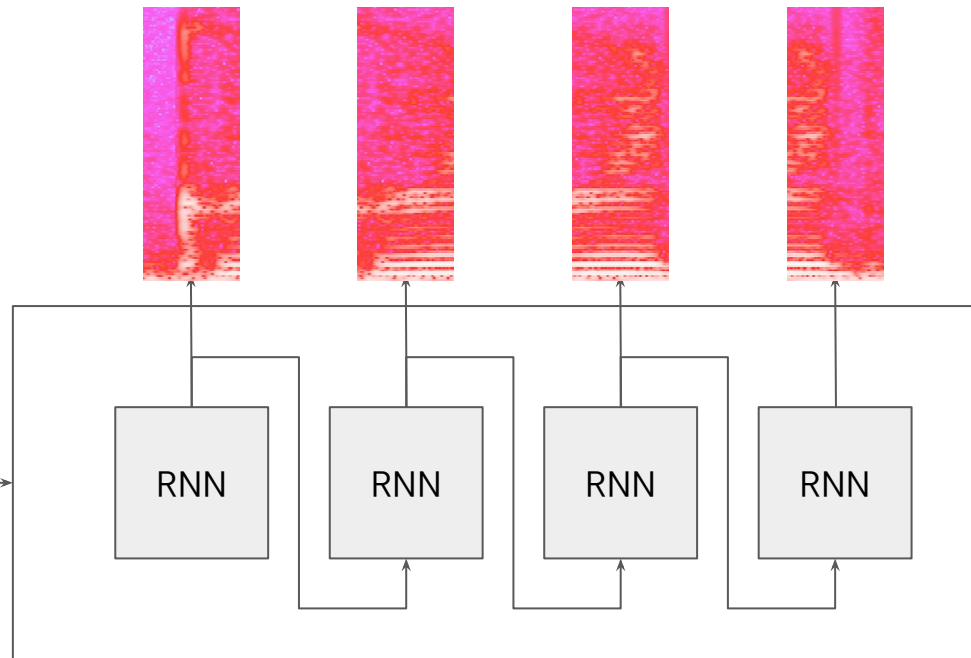
Speaker Spectrogram



CNN/RNN

CNN/RNN

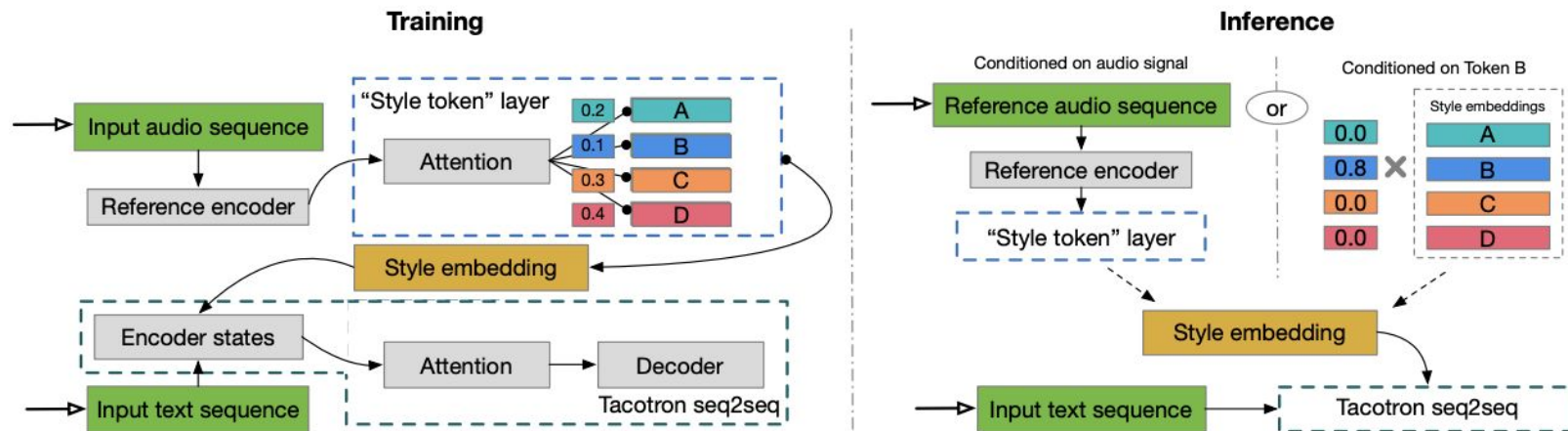
“g” “r” “a” “c” “e”



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
- Substantially compress the spectrogram before back prop or the task will become trivial
- At inference time feed in a reference mel spectrogram or sample from the latent space
- Can be achieved with token embeddings or a VAE

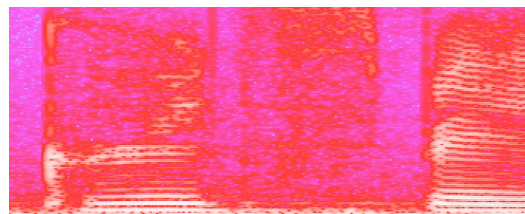
GST Tacotron



*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.*

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



CNN/RNN

VAE Latent Space

$$z_t \sim N(0, 1)$$

CNN/RNN

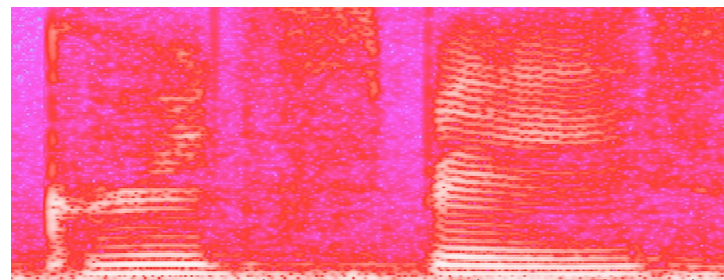
“g”

“r”

“a”

“c”

“e”

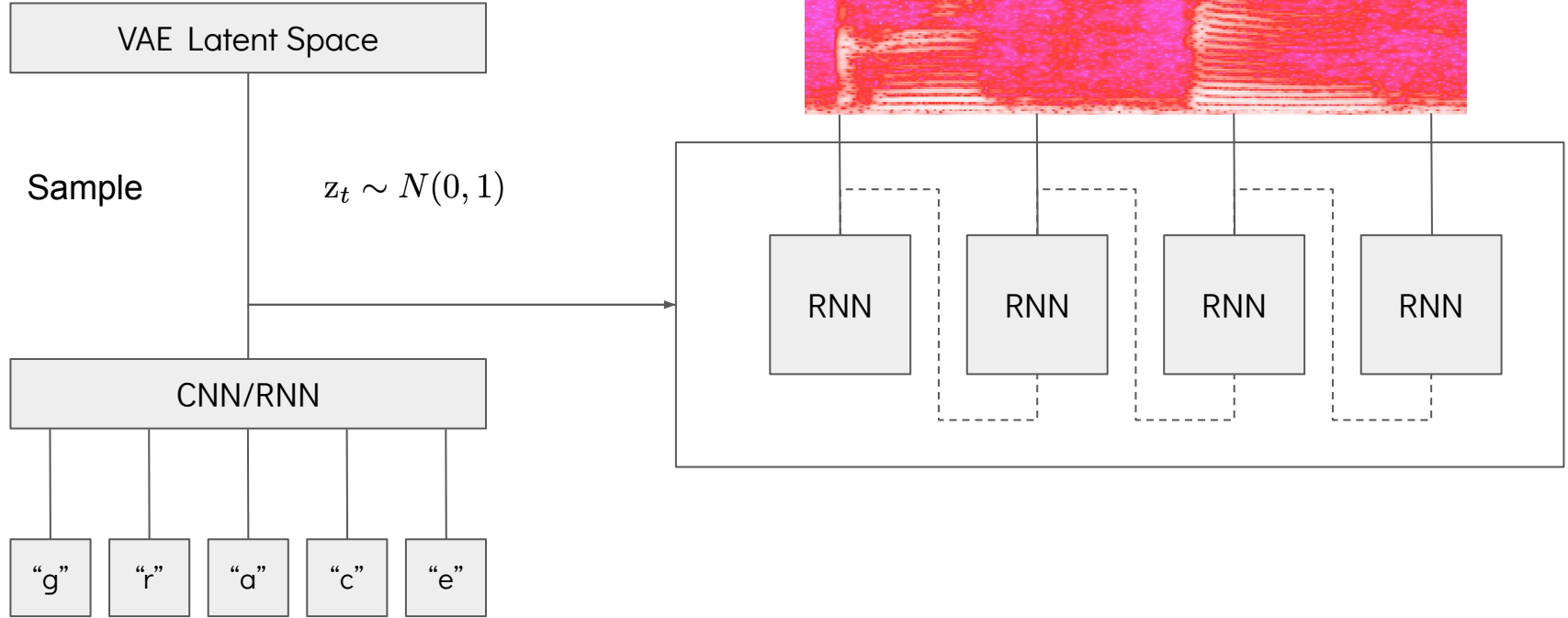
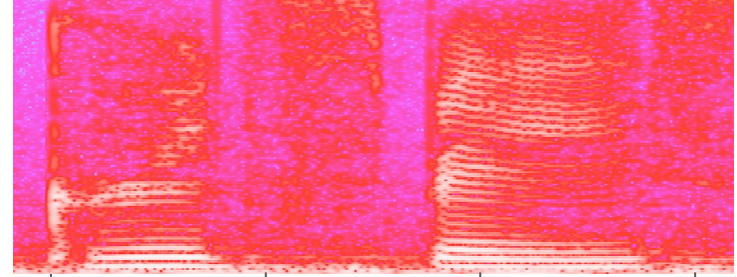


RNN

RNN

RNN

RNN



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

TTS at Gridspace: Grace Demo

Thank You!

- Look forward to seeing all the final projects!
- Please reach out if you have interest in working on cutting edge TTS, ASR, dialog systems and infrastructure at Gridspace

Email: alex@gridspace.com

Information Session: 10am-1pm Friday March 5th

Zoom link on Piazza