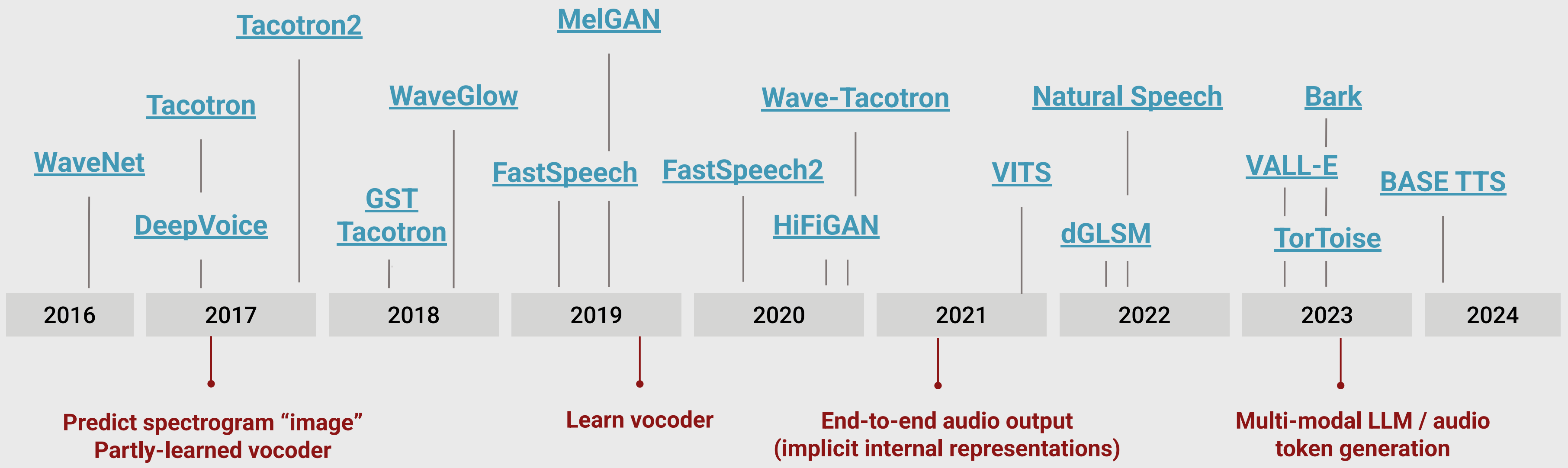
# CS 224S / Linguist 285
# Spoken Language Processing

Andrew Maas | Stanford University | Spring 2025

## Lecture 5: Deep Learning for TTS

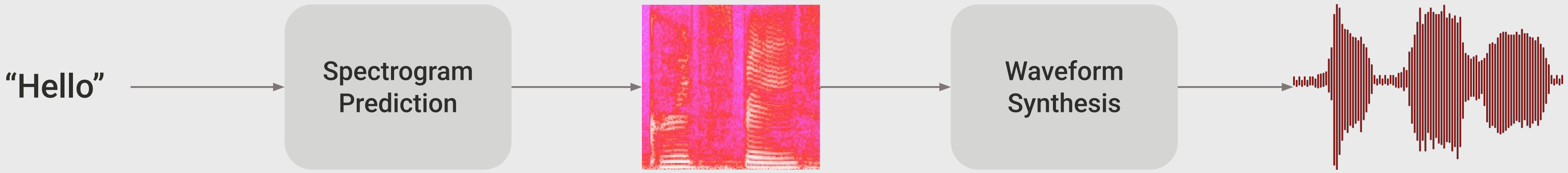
Slide contributions by Alex Barron & Mike Wu

# Rapid Progress in TTS Over the Last Few Years



## Neural TTS paradigm: Increasingly text to waveform

Decreasing reliance on explicit intermediate representations



## Outline

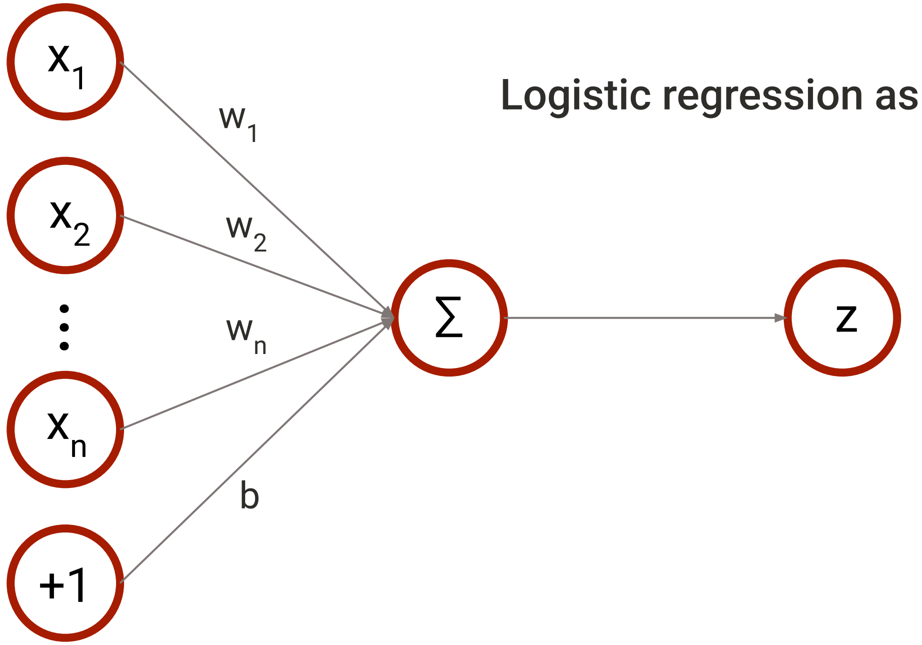
- Neural Network Basics: Hidden Layers and Gradient Computation
- Recurrent NNs
- Attention
- Transformers: stacking attention

*~35 minutes on these topics. If you need more introduction please see readings and CS124 + CS224N + CS229 etc. as well as online tutorials*

- Text to Spectrogram Models
- Speaker and Style Embeddings
- Spectrogram to Audio Models (Vocoders)

# Neural Network Basics: Hidden Layers and Gradient Computation

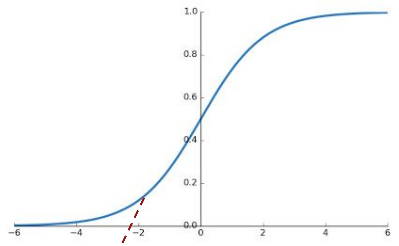
# Logistic Regression



Logistic regression as a "neuron"

$= \sum w_i x_i + b$

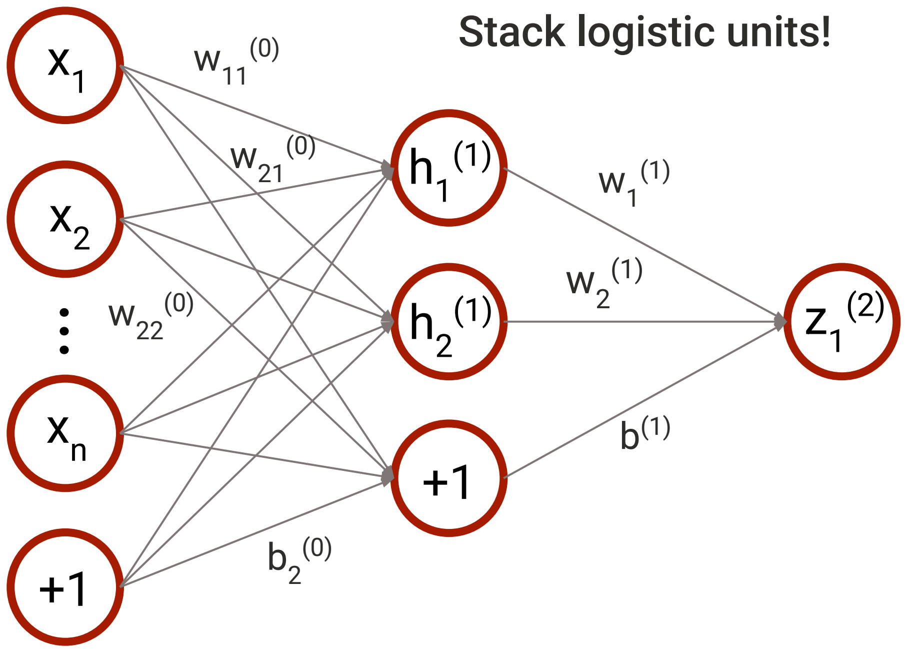
nonlinearity



Outputs $p(y=1|x_1 \ldots x_n) = \sigma(z)$, the probability of predicting class 1!

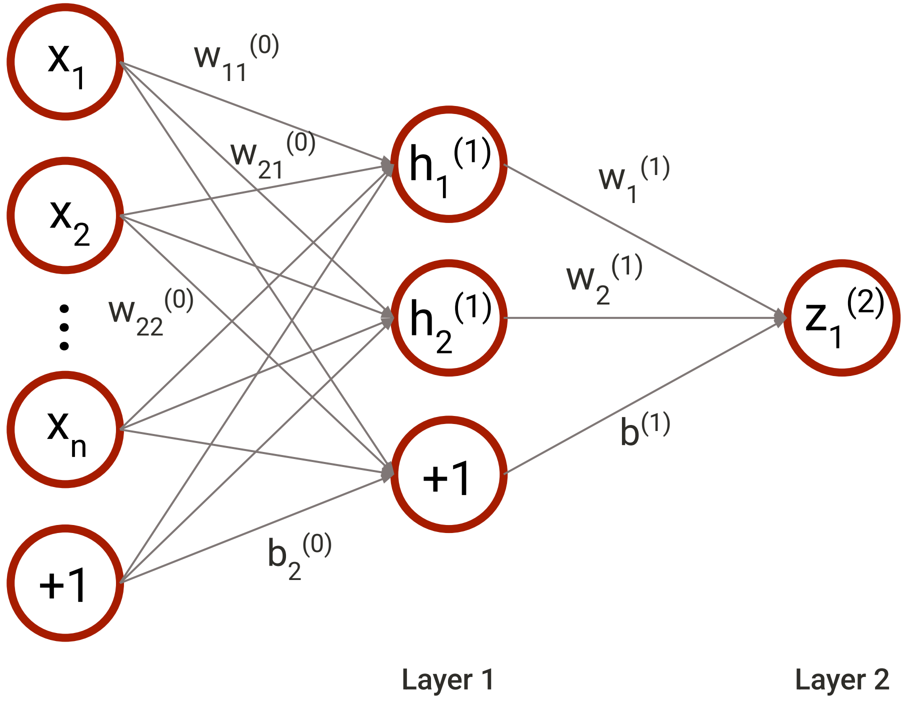
# Multi-layer Perceptrons (MLPs)

Stack logistic units!



"Hidden Layer"

# Multi-layer Perceptrons (MLPs)



## Forward Propagation

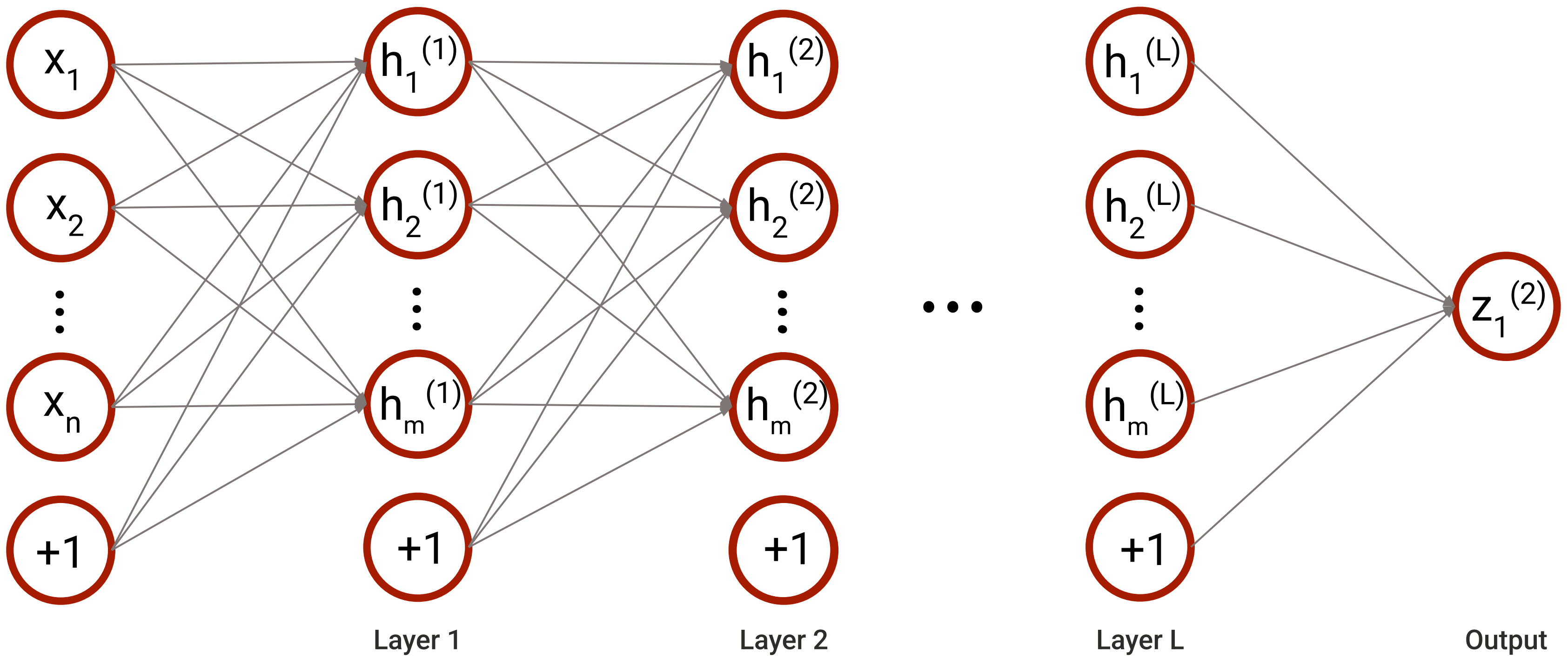
$z_j^{(l+1)} = \sum w_{ij}^{(l)} a_i^{(l)} + b_j^{(l)}$

$h_j^{(l)} = \sigma(z_j^{(l)})$ "*activation*"

$\theta = \{ w_{ij}^{(l)}, b_j^{(l)} \text{ for all } i,j,l \}$

^ "parameters"

# “Deep” Neural Networks



## Objective Function

- Depends on the task! Examples:

**Binary classification**

**Label:** $y \in \{0,1\}$

**Objective:**

p = sigmoid(z)

$J_\theta$ = y log p + (1-y) log(1-p)

**Multiclass classification**

**Label:** $y \in \{1,\ldots,K\}$

**Objective:**

p1:K = [p1, …, pk] = softmax(z1:K)

$J_\theta$ = -∑ yconehot log pc

**Regression**

**Label:** $y \in R^d$

**Objective:**

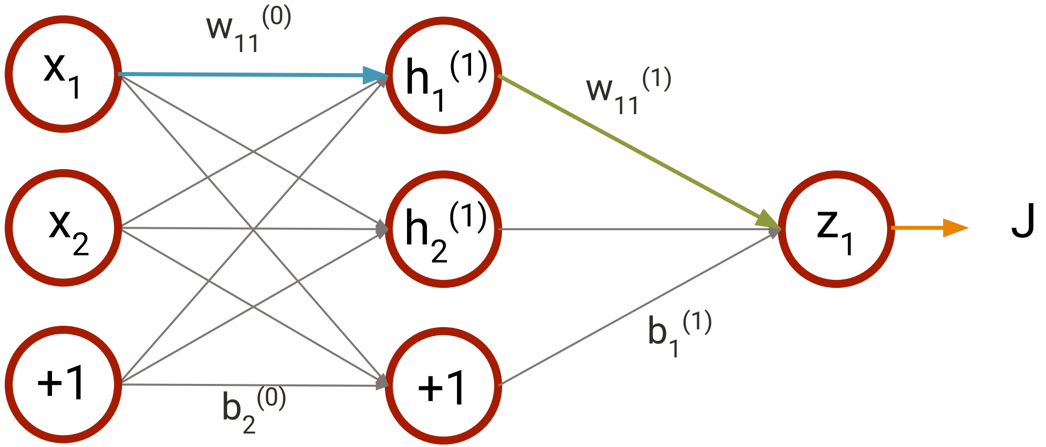
$J_\theta$ = sum(sqrt(z - y)) / d

Now we have to optimize!

# Backpropagation

Let's do $dJ/dw_{11}^{(0)}$ as an example:

$$\frac{dJ}{dw_{11}^{(0)}} = \frac{dJ}{dz_1}\frac{dz_1}{dw_{11}^{(0)}} = \frac{dJ}{dz_1}\left(\frac{dz_1}{dh_1}\frac{dh_1}{dw_{11}^{(0)}}\right)$$



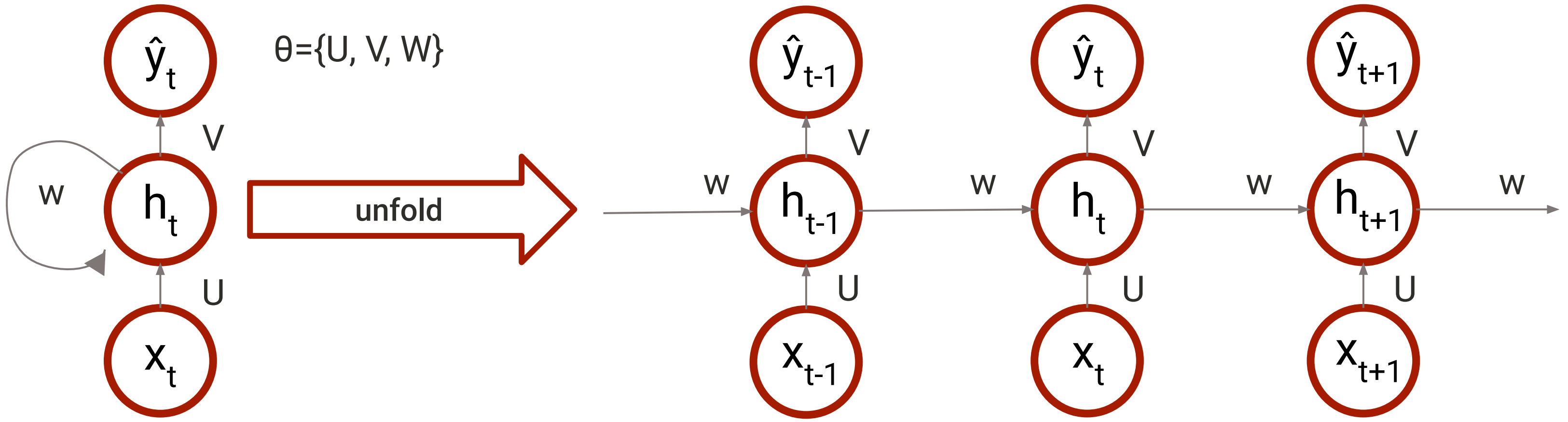
**Use chain rule!**

For a fixed objective J **and** a fixed architecture, you can compute a closed form for every derivative.

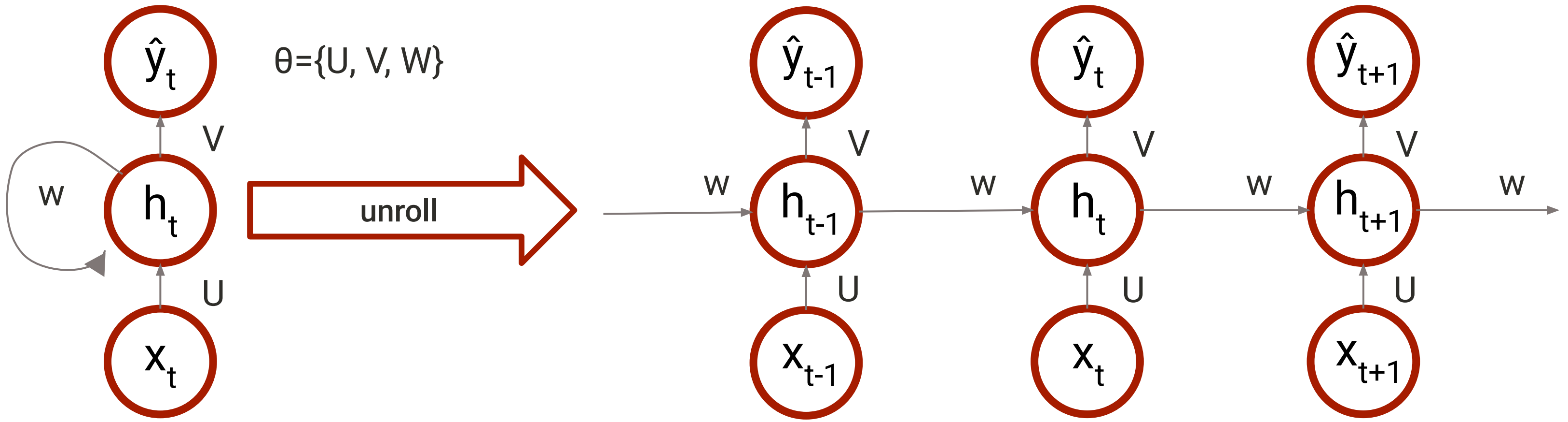
## Recurrent NNs

## Recurrent NNs

- Input is a sequence of tokens $(x_1, x_2, \ldots, x_t)$
- Output is a sequence of tokens $(y_1, y_2, \ldots, y_t)$
- Goal is map $x_t$ to a "hidden state" $h_t$ (a real-valued vector)
- Think of ht is a nonlinear summary of $(x_1, \ldots, x_t)$
- Use $h_{t-1}$ and xt to predict $y_t$



## Recurrent NNs

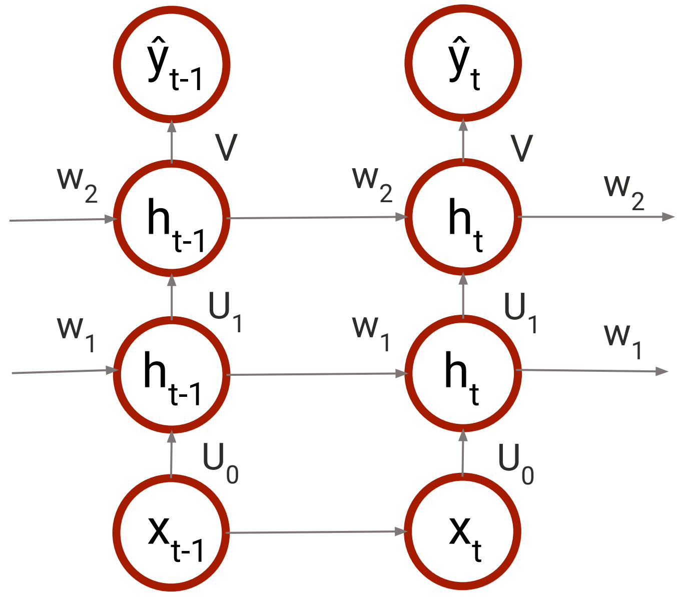


- U, V, W are shared over all timesteps (same ones!)
- The model does a forward pass for every single timestep: $f_\theta(x_t, h_{t-1})$
- Objective (assume x and y are discrete tokens)

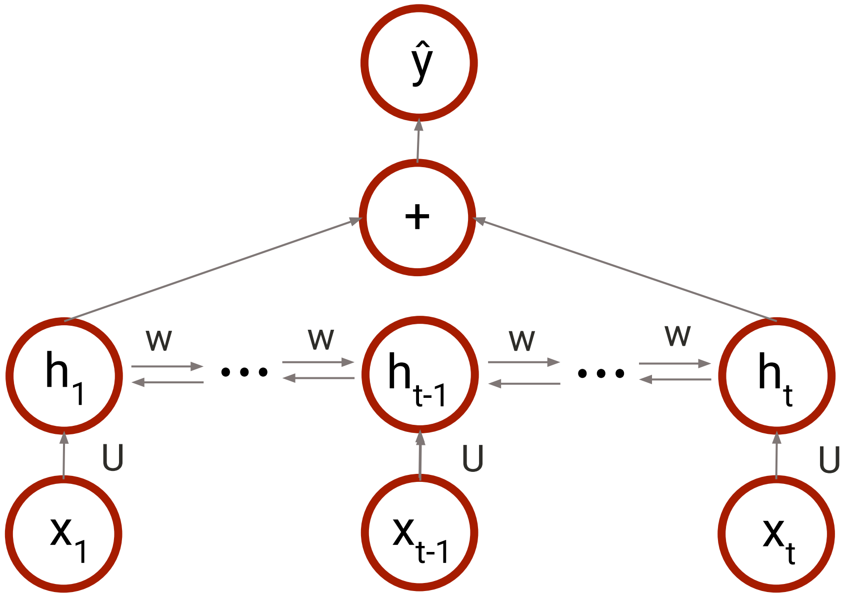
$$J(x,y,\theta) = - \sum_t \log p(y_t \mid x_1, \ldots, x_t) = - \sum_t \text{CrossEntropy}(y_t, f_\theta(x_t, h_{t-1}))$$

- Backpropagation through time (BPTT) [Rumelhart et. al. 1986, Werbos 1990]

# Recurrent NNs

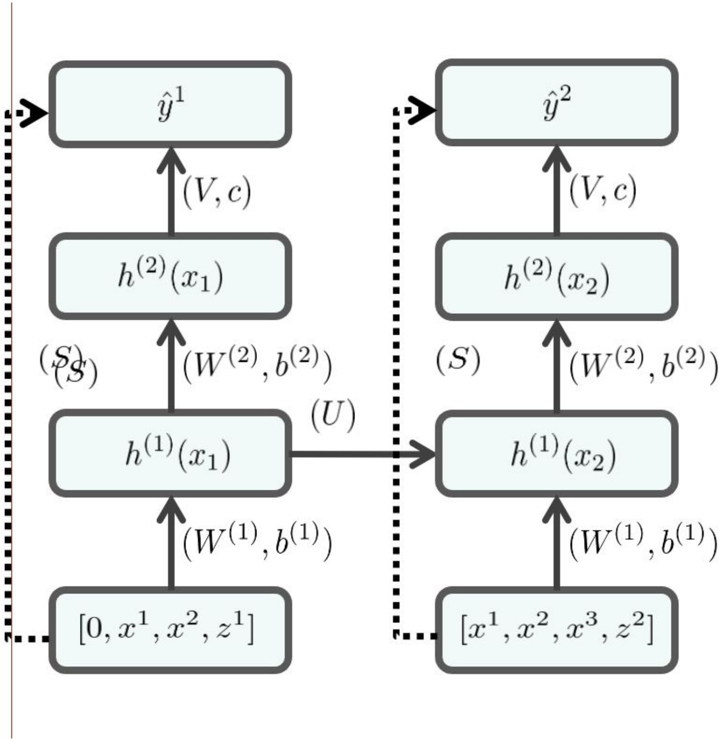


Stacked RNNs



Bi-directional RNNs

# RNN application: Speech spectrogram denoising



Output Layer

$$\hat{y}^2 = Vh^{(2)}(x_2) + Sx_2 + c$$

Hidden Layer

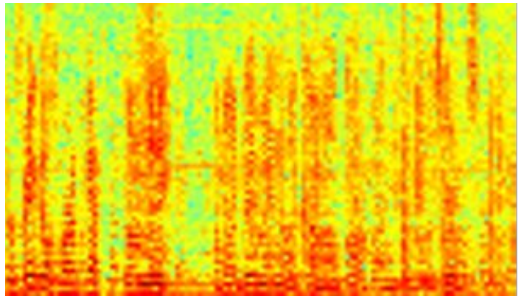
$$h^{(2)}(x_2) = \sigma(W^{(2)}h^{(1)}(x_2) + b^{(2)})$$

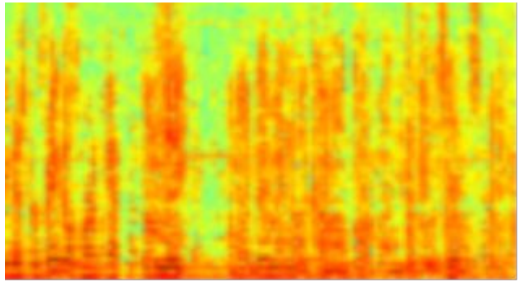
Hidden Layer

$$h^{(1)}(x_2) = \sigma(W^{(1)}x_2 + b^{(1)} + Uh^{(1)}(x_1))$$

Noisy Input **X**

Noise Estimate **Z**



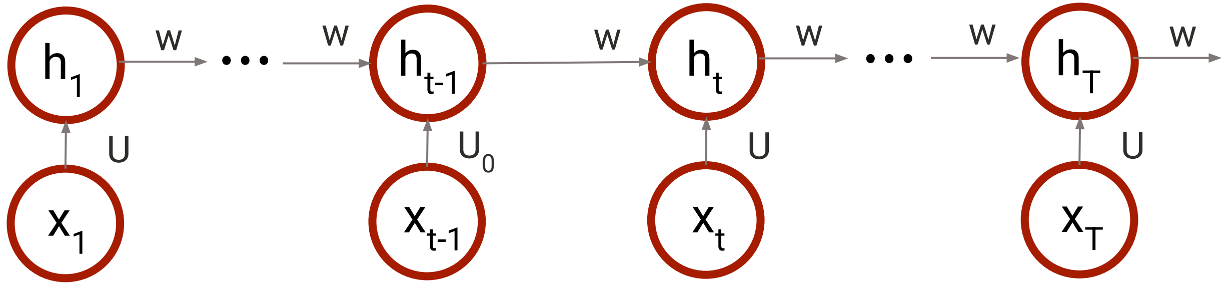


(Maas *et al*, 2013)

## Encoder-Decoder Models

# RNN Encoder

- Input is a sequence of tokens $(x_1, x_2, \dots, x_T)$
- Goal: Encode sequence information into single, final timestep hidden layer representation

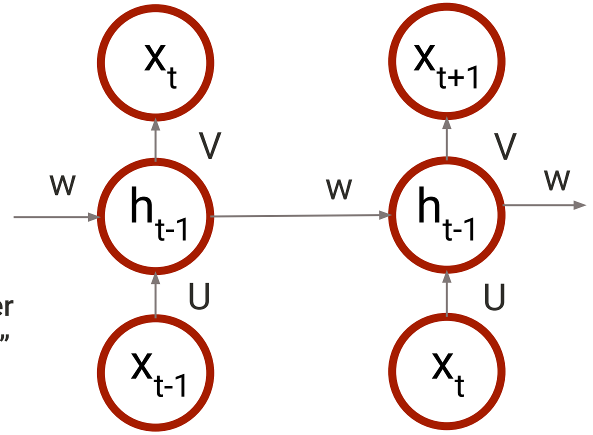


## RNN Decoder

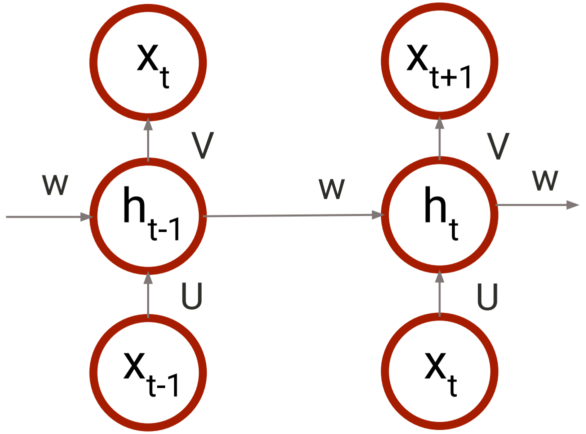
- Input is a sequence of tokens $(x_1, x_{2,} \ldots, x_T)$ , no output sequence.
- The model should learn $p(x_t \mid x_1, \ldots, x_{t-1})$.

We don't know the true $x_t$!

“Teacher Forcing”



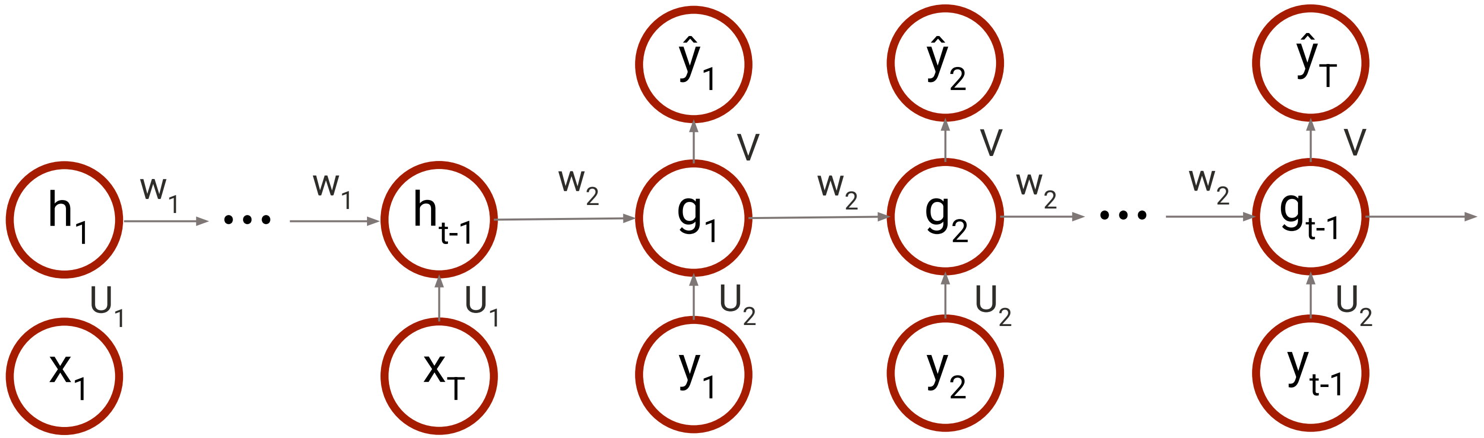
Training



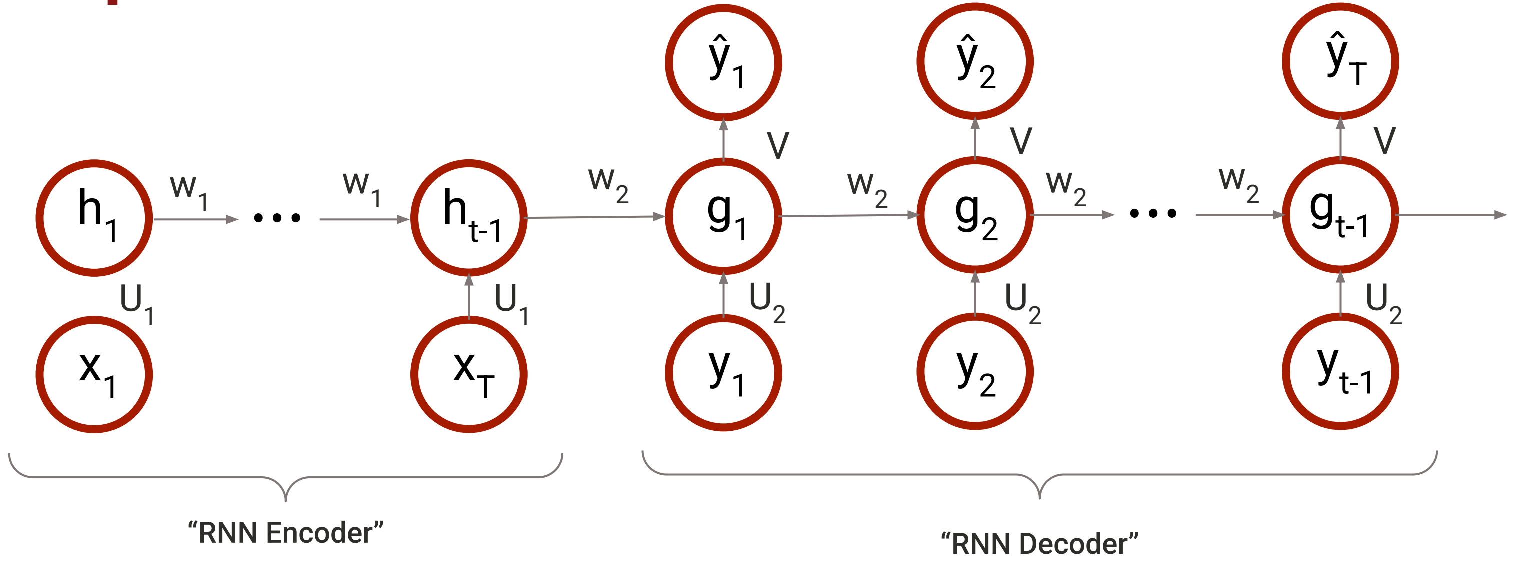
Generation

## Sequence to sequence RNN models (Seq2Seq)

- All that work and we have our first encoder-decoder model!
- Given an input and output sequence $(x_1, x_2, \ldots, x_t), (y_1, y_2, \ldots, y_{t'})$, the model should capture $p(y_t \mid y_1, \ldots, y_{t-1}, x_1, \ldots, x_T)$. It gets to see all of x!



# Seq2Seq full model



- Two different RNNs glue'd together (separate parameters)
- One of them encodes $(x_1, \ldots, x_T)$ into a summary vector, $h_T$
- The other one uses $h_T$ to initialize a language model
- Train this just like an RNN language model (x = speaker 1, y = speaker 2)

## Limitations of RNNs

- If you have a really long sequence (e.g. T=1000), hard to believe $h_{854}$ will remember $x_2$
- $h_t$ captures "local" info since it has to predict $y_t$ (little incentive to remember $h_{t-100}$).

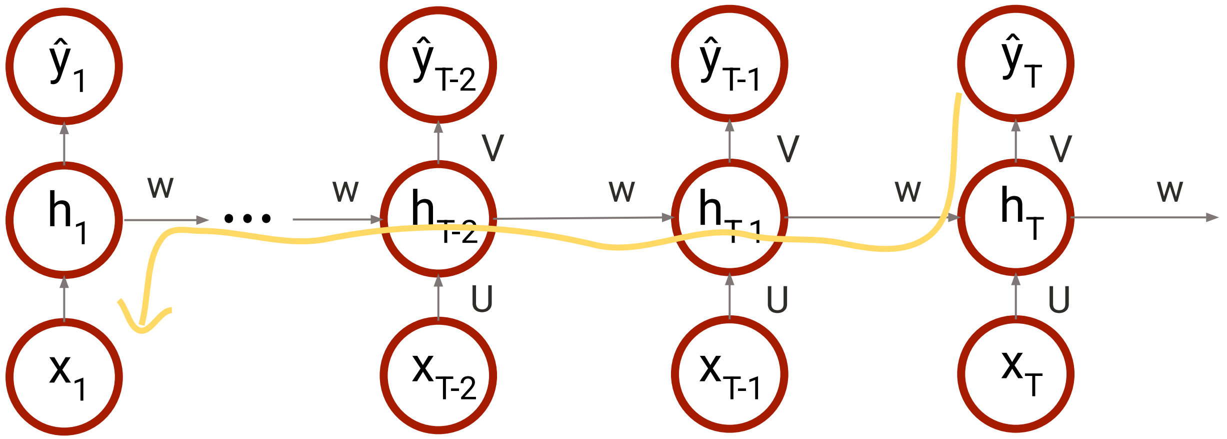
But long range dependencies are important

**Example:** … the flights the airline was cancelling were full.

What flights??? What airline???

If you have building a chatbot, you might need to remember things from long ago.

# Vanishing Gradients



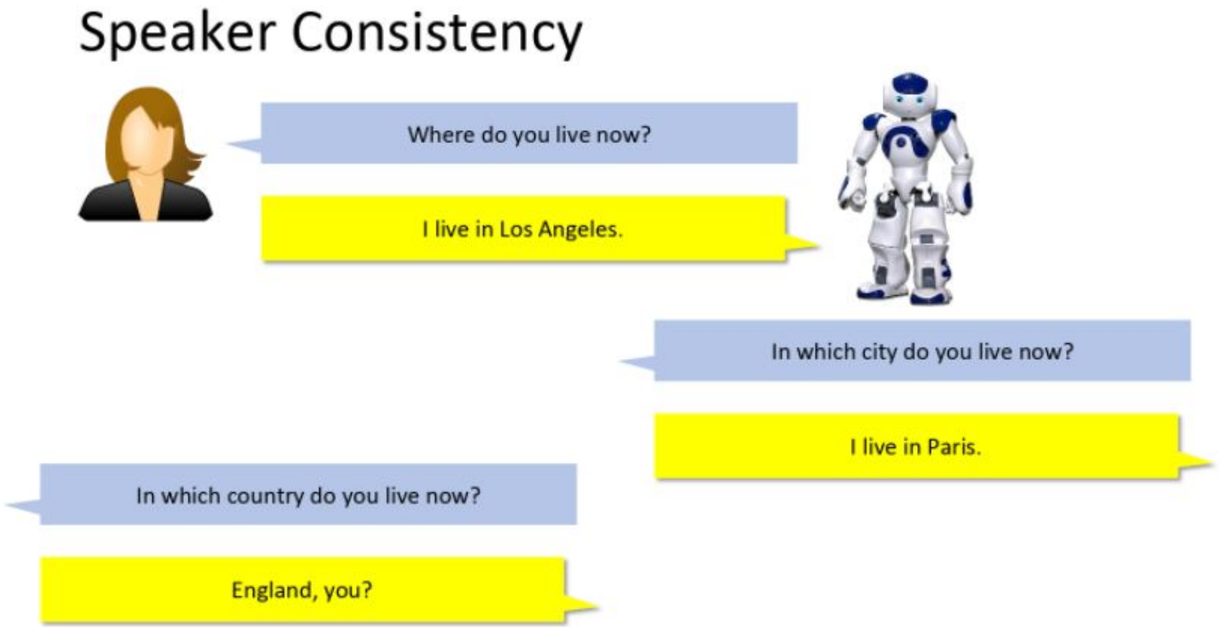
- Suppose T →∞. Say we want to calculate $dy_t / dx_1$
- What happens if $dh_t/dh_{t-1} < 1$ for all t?
- Impact of $x_{t-s}$ on $y_t$ decreases as s increases.

## Attention! And transformers

# Attention

**Holy grail:** capturing long term dependencies. Especially in NLProc and speech audio where long, context-rich sequences exist

- Vanishing gradient problem & local dependency of RNNs.
- Attention gets at this more directly (and simply).

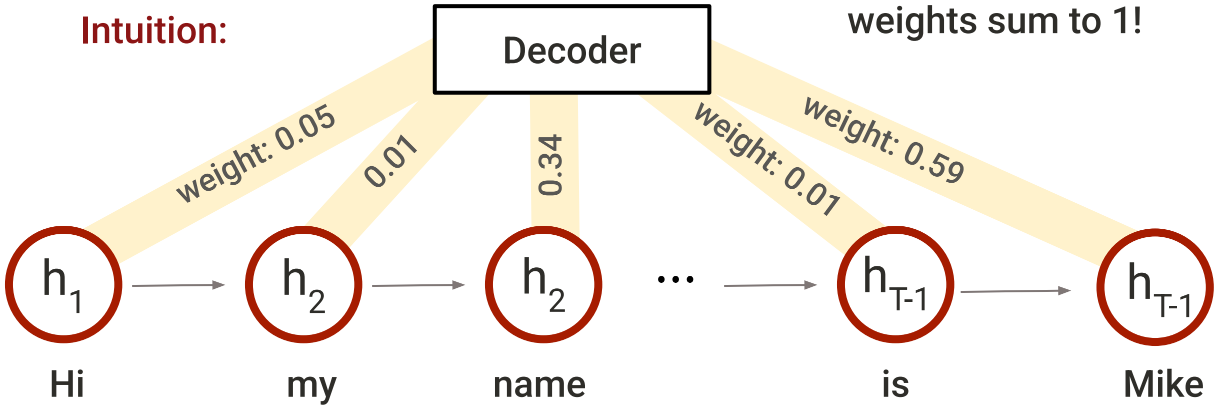


## Attention

**Holy grail:** capturing long term dependencies. Especially in NLProc and speech audio where long, context-rich sequences exist

- Vanishing gradient problem & local dependency of RNNs.
- Attention gets at this more directly (and simply).

Intuition:



## Attention Mechanism

$q \in \mathbb{R}^d$ : query ; $k_1, \ldots, k_T \in \mathbb{R}^d$ : keys ; $v_1, \ldots, v_T \in \mathbb{R}^d$ : values

## Attention Mechanism

$q \in R^d$ : query ; $k_1, \ldots, k_T \in R^d$ : keys ; $v_1, \ldots, v_T \in R^d$ : values

**Step 1: define a similarity function sim($q$, $k_t$).**

| | | |
|---|---|---|
| $\text{sim}(q, k_t) = w_2\text{relu}(w_1[q; k_t] + b_1) + b_2$ | [Bahdanau et. al. 2014] | (MLP) |
| $\text{sim}(q, k_t) = q^TWk_t$ | [Luong et. al. 2015] | (Bilinear) |
| $\text{sim}(q, k_t) = q^Tk_t$ | [[Luong et. al. 2015] | (dot-pdt) |
| $\text{sim}(q, k_t) = q^Tk_t \ / \ \text{sqrt}(d)$ | [Luong et. al. 2015] | (scaled-dot-pdt) |

## Attention Mechanism

$q \in R^d$ : query ; $k_1, \ldots, k_T \in R^d$ : keys ; $v_1, \ldots, v_T \in R^d$ : values

Step 1: define a similarity function $sim(q, k_t)$.

**Step 2:** compute attention weights ($a$).

$$a_t = \frac{\exp\{ sim(q, k_t) \}}{\sum_{s=1}^{T} \exp\{ sim(q, k_s) \}}$$

Note $a_t \in [0, 1]$ for all t
Also $\sum_t a_t = 1$

# Attention Mechanism

$q \in R^d$ : query ; $k_1, \ldots, k_T \in R^d$ : keys ; $v_1, \ldots, v_T \in R^d$ : values

Step 1: define a similarity function sim($q$, $k_t$).

Step 2: compute attention weights ($a$).

**Step 3**: attend to values vectors.

$c = \sum^T_{t=1} a_t \, v_t$ weighted linear combo of values!

## Multi-headed Attention

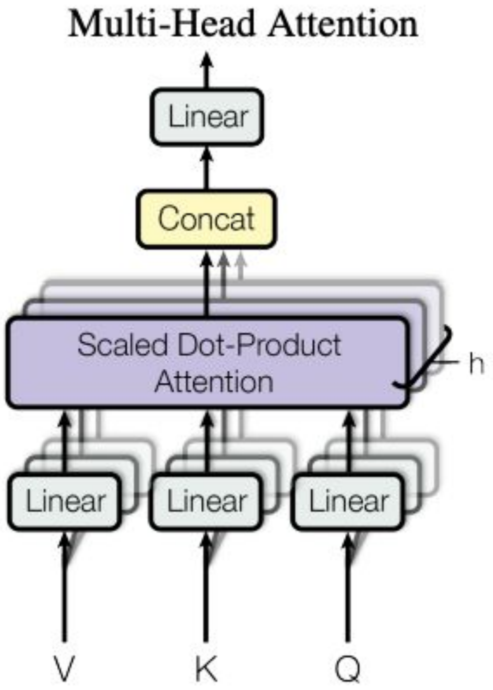
- What if I want to pay attention to different things at the same time!?

| | |
|---|---|
| | This is my big red dog, Clifford. |
| CONTENT- BASED | This is my big red dog, Clifford. |
| DESCRIPTION-BASED | This is my big red dog, Clifford. |
| REFERENCE-BASED | This is my big red dog, Clifford. |

- What's useful depends on the task. How do I pick what to do?

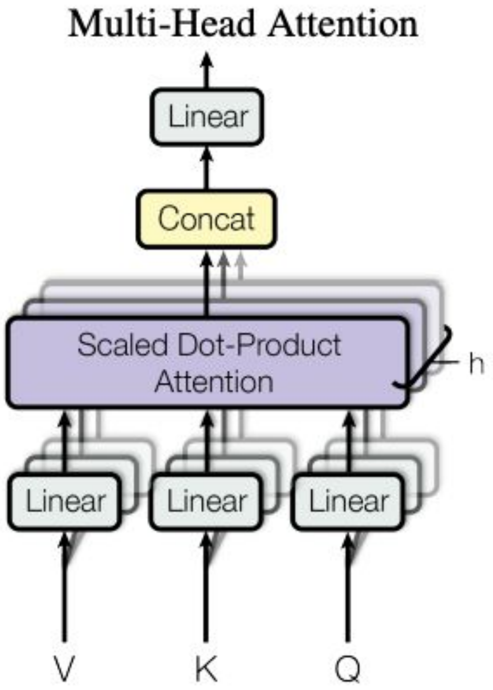
## Idea: Don’t pick. Pay Attention as if You Had Multiple Heads

Vaswani et. al. 2017



## Idea: Don't pick. Pay Attention as if You Had Multiple Heads

Vaswani et. al. 2017



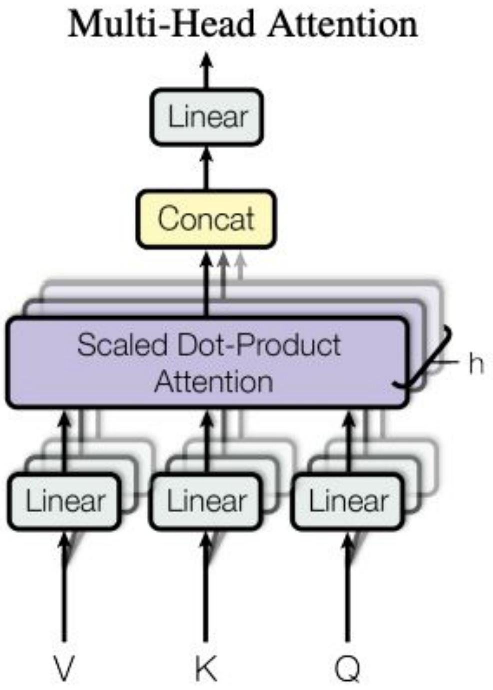
Pick H heads.

```
For h in range(H):

        $q^{(h)} = MLP_h(q)$
        $k^{(h)}_t = MLP_h(k_t)$ for all t
        $v^{(h)}_t = MLP_h(v_t)$ for all t
```

## Idea: Don't pick. Pay Attention as if You Had Multiple Heads

Vaswani et. al. 2017



Pick H heads.

For h in range(H):

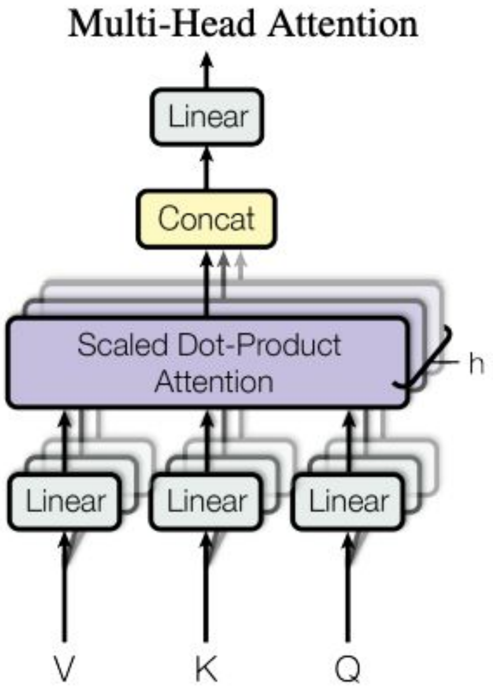
$q^{(h)} = \mathrm{MLP}_h(q)$
$k^{(h)}_t = \mathrm{MLP}_h(k_t)$ for all t
$v^{(h)}_t = \mathrm{MLP}_h(v_t)$ for all t

$$a^{(h)}_t = \frac{\exp\{ q^{(h)T} k_t^{(h)} / \mathrm{sqrt}(d) \}}{\sum^T_{s=1} \exp\{ q^{(h)T} k_s^{(h)} / \mathrm{sqrt}(d) \}}$$

$$c^{(h)} = \sum^T_{t=1} a^{(h)}_t v^{(h)}_t$$

## Idea: Don't pick. Pay Attention as if You Had Multiple Heads

Vaswani et. al. 2017



Pick H heads.

For h in range(H):

$q^{(h)} = \text{MLP}_h(q)$
$k^{(h)}_t = \text{MLP}_h(k_t)$ for all t
$v^{(h)}_t = \text{MLP}_h(v_t)$ for all t

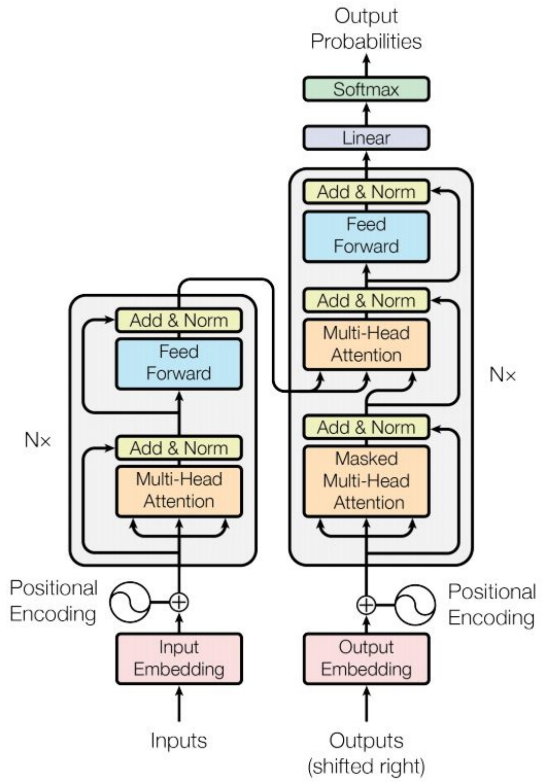
$$a^{(h)}_t = \frac{\exp\{ q^{(h)T} k_t^{(h)} / \text{sqrt}(d) \}}{\sum^T_{s=1} \exp\{ q^{(h)T} k_s^{(h)} / \text{sqrt}(d) \}}$$

$$c^{(h)} = \sum^T_{t=1} a^{(h)}_t v^{(h)}_t$$

$c_{all} = \text{concat}(c^{(1)}, \ldots, c^{(H)})$
$c = \text{linear}(c_{all})$ # project to smaller dimension

## Transformers (SOTA)

# Transformer: Self-attention Nets



- A transformer layer is composed of an encoder and a decoder.
- Both use the same building blocks.
- Similar to RNNs, here…
  - Encoder sees $(x_1, x_2, \ldots, x_T)$ and outputs a hidden sequence $(h_1, h_2, \ldots, h_T)$.
  - Decoder sees $(x_1, x_2, \ldots, x_{T-1})$ and outputs predictions for $(x_2, x_3, \ldots, x_T)$.

## Self Attention

Vaswani et. al. 2017

**Simple idea:** query, keys and values in attention are the same.

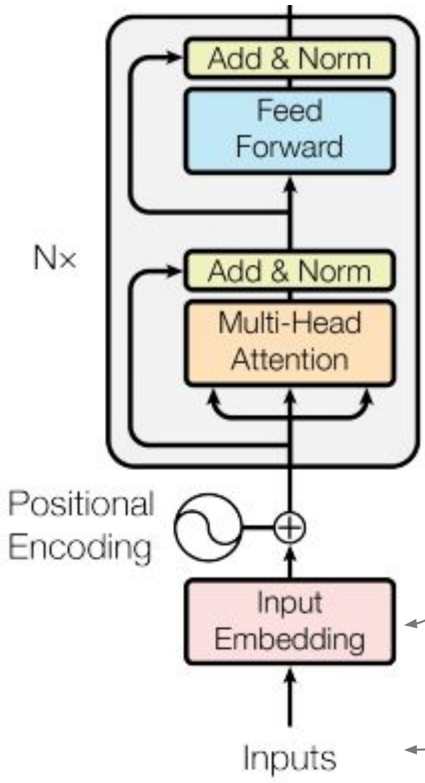
Take some sequence $(x_1, x_2, \ldots, x_T)$. For every t, build:

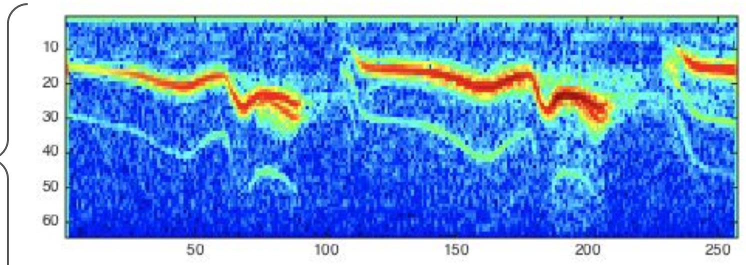
$$q_t = \mathrm{MLP}(x_t)\ ,\ k_t = \mathrm{MLP}(x_t)\ ,\ v_t = \mathrm{MLP}(x_t)$$

Then we can extract a sequence:

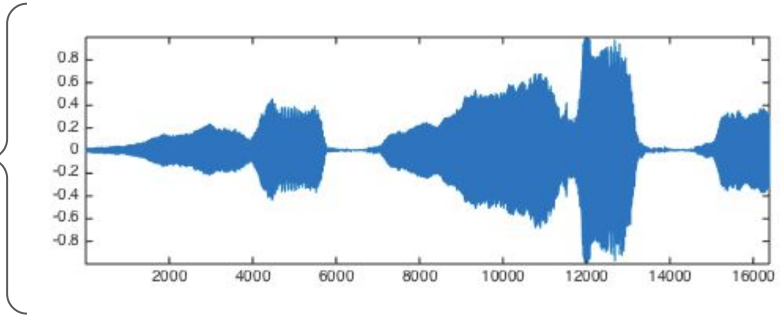
$$c_t = \sum^{T}_{t=1} a_t\, v_t \Longrightarrow (c_1, c_2, \ldots, c_T)$$

# Transformer Encoder





$(e_1, e_2, \ldots, e_T)$ e.g. mel spectrogram



$(x_1, x_2, \ldots, x_T)$ e.g. a waveform

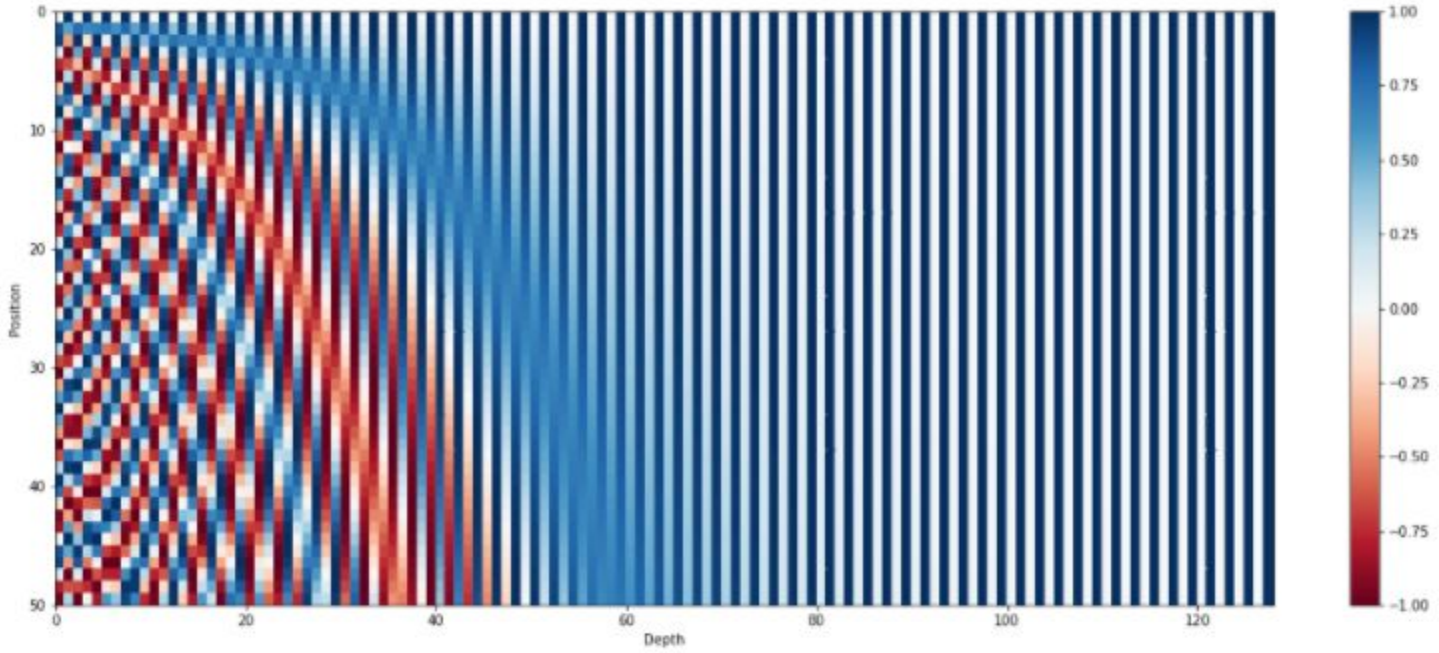
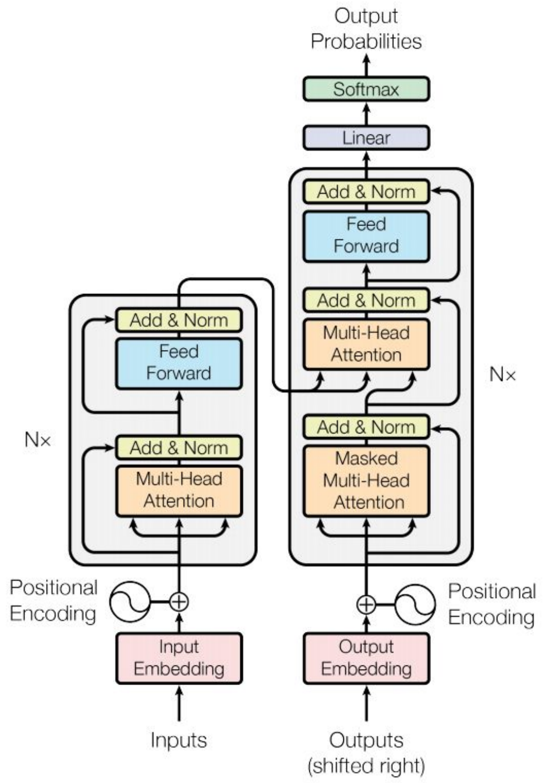


*Figure 2 – The 128-dimensional positonal encoding for a sentence with the maximum lenght of 50. Each row represents the embedding vector* $\overrightarrow{p_t}$

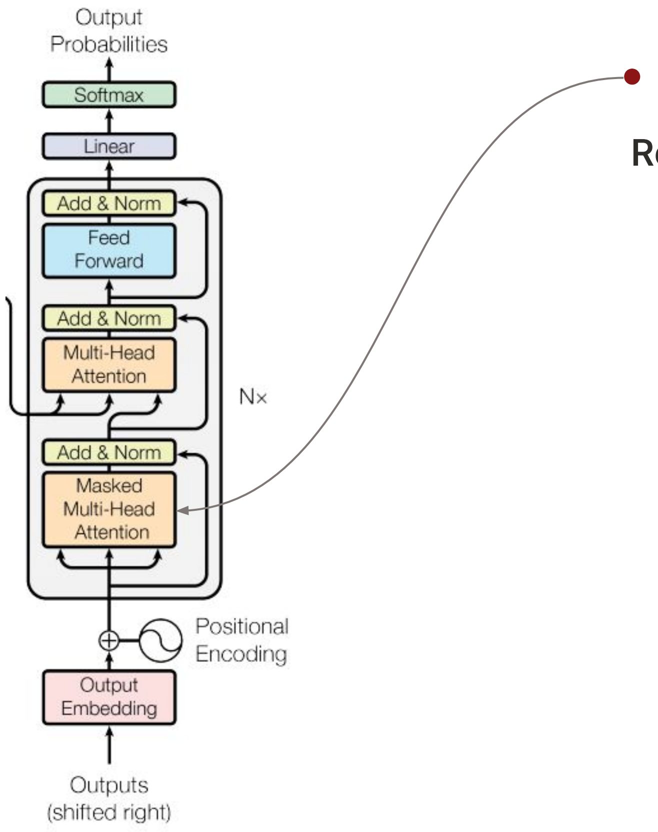
[Source post](Source post)

# Transformer Decoder



- Encoder Multi-head Attention:
- Output of encoder: $(h_1^{enc}, h_2^{enc}, \ldots, h_T^{enc})$.
- Use this for keys and values in attention.
- Query vectors come from decoder.
- This blends information from encoder into the decoder.
- Note: no bleeding problem here!
- SOTA speech recognition network is a variant of this idea for online+offline training with weight sharing

# Transformer Decoder

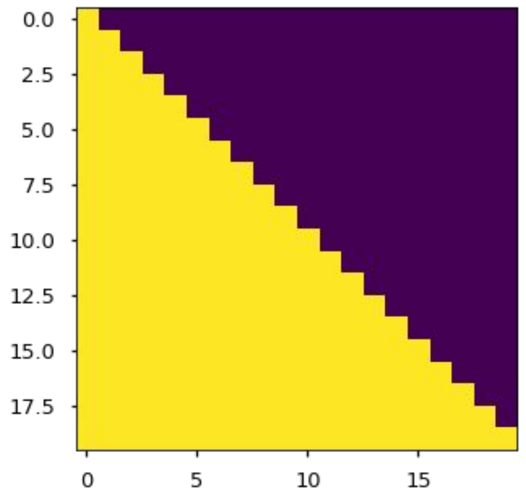


- Masked Multi-head Attention:

Recall, q : query ; $k_1, \ldots, k_T$: keys ; $v_1, \ldots, v_T$: values

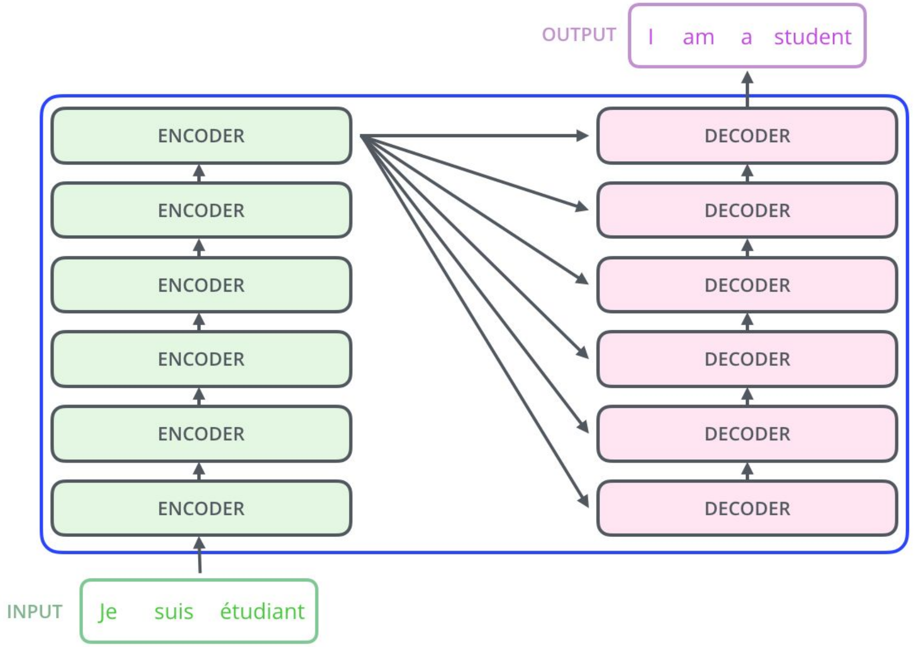
$$\text{maskedsim}(q, k_t, m) = m^T(q^T k_t) / \text{sqrt}(d)$$

m =



## Stacked Transformers

- The trend is make things deep. A single transformer encoder or decoder returns a sequence of the same signature as the input.



Github

## Residual stream view of transformer blocks

See Jurafsky & Martin chapter 9

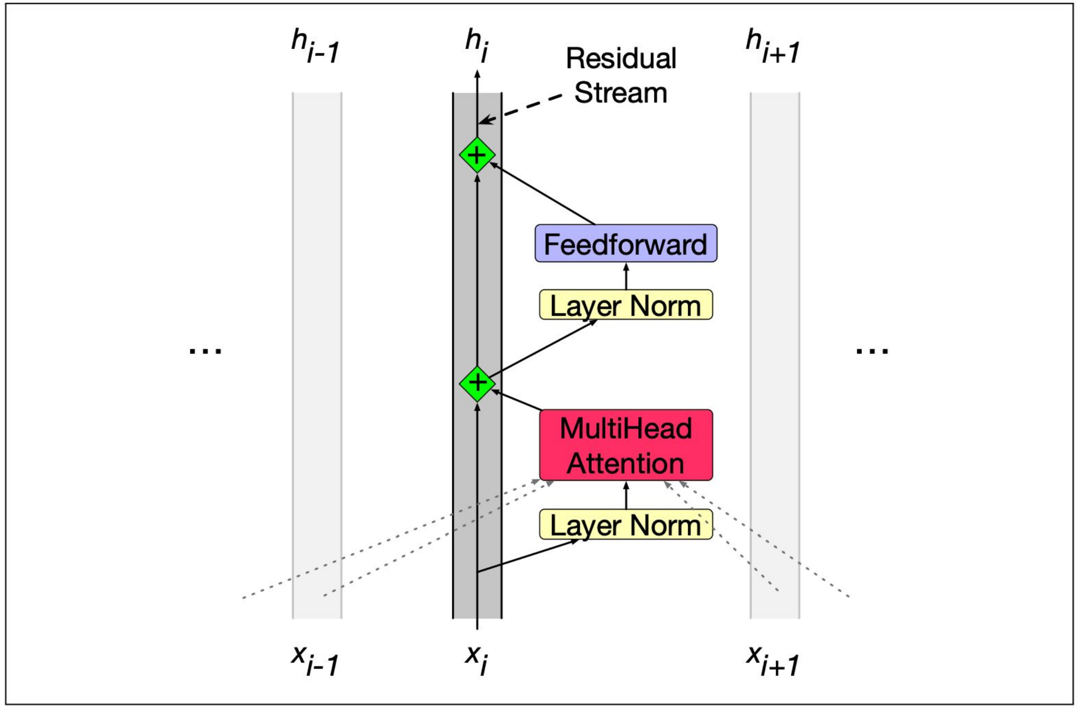
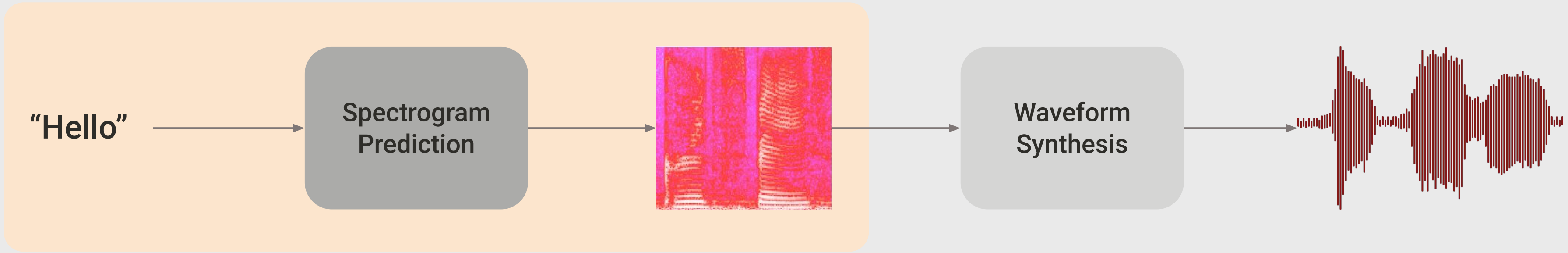


**Figure 9.6** The architecture of a transformer block showing the **residual stream**. This figure shows the **prenorm** version of the architecture, in which the layer norms happen before the attention and feedforward layers rather than after.

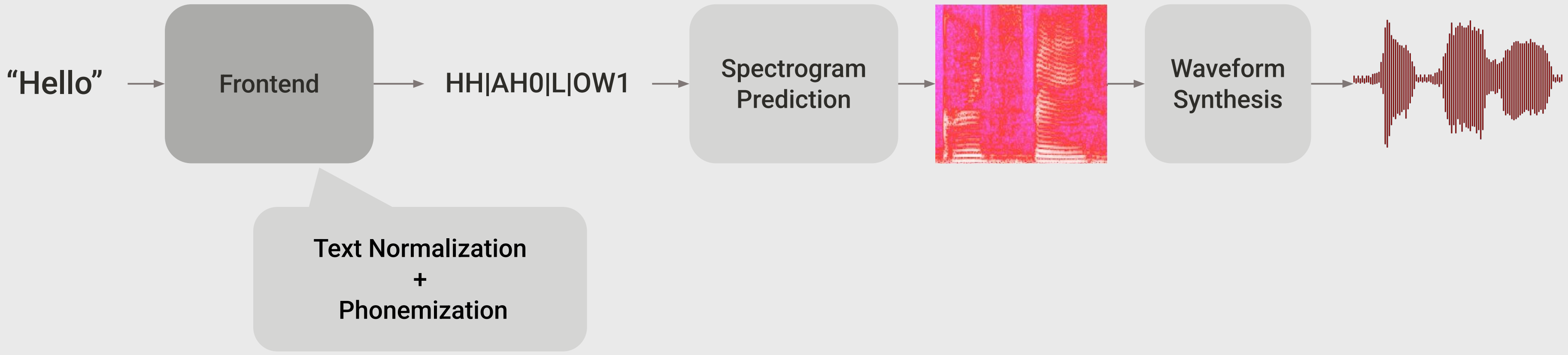
## Remaining topics for today

- Text to Spectrogram Models
- Speaker and Style Embeddings
- Spectrogram to Audio Models (Vocoders)

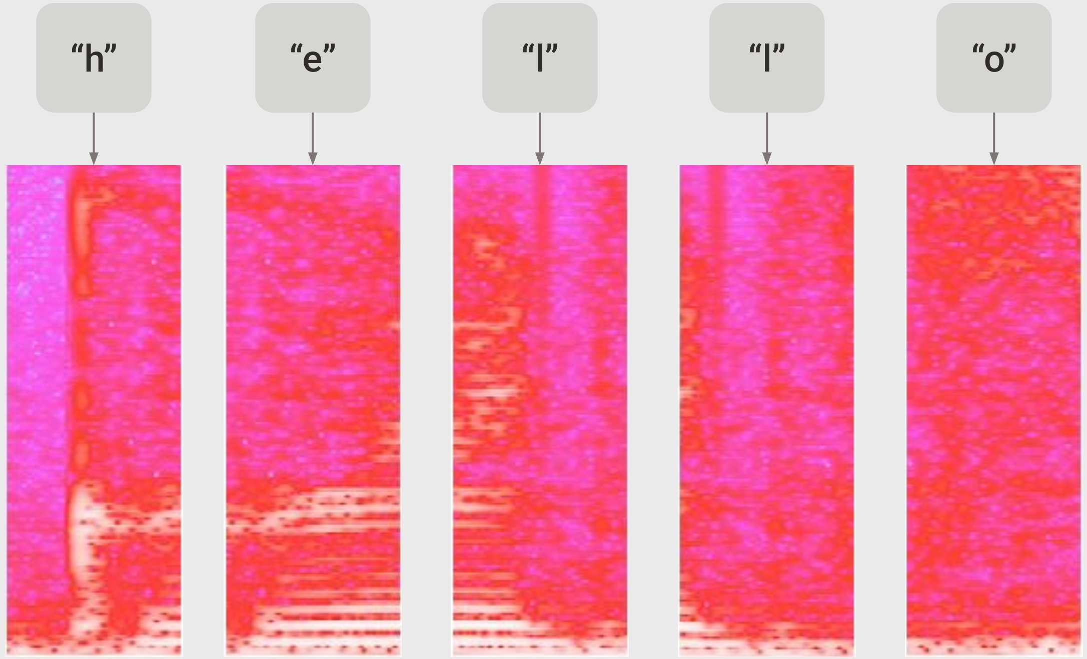
## Neural TTS Paradigm



## Neural TTS Paradigm



## Sequence to Sequence Problem

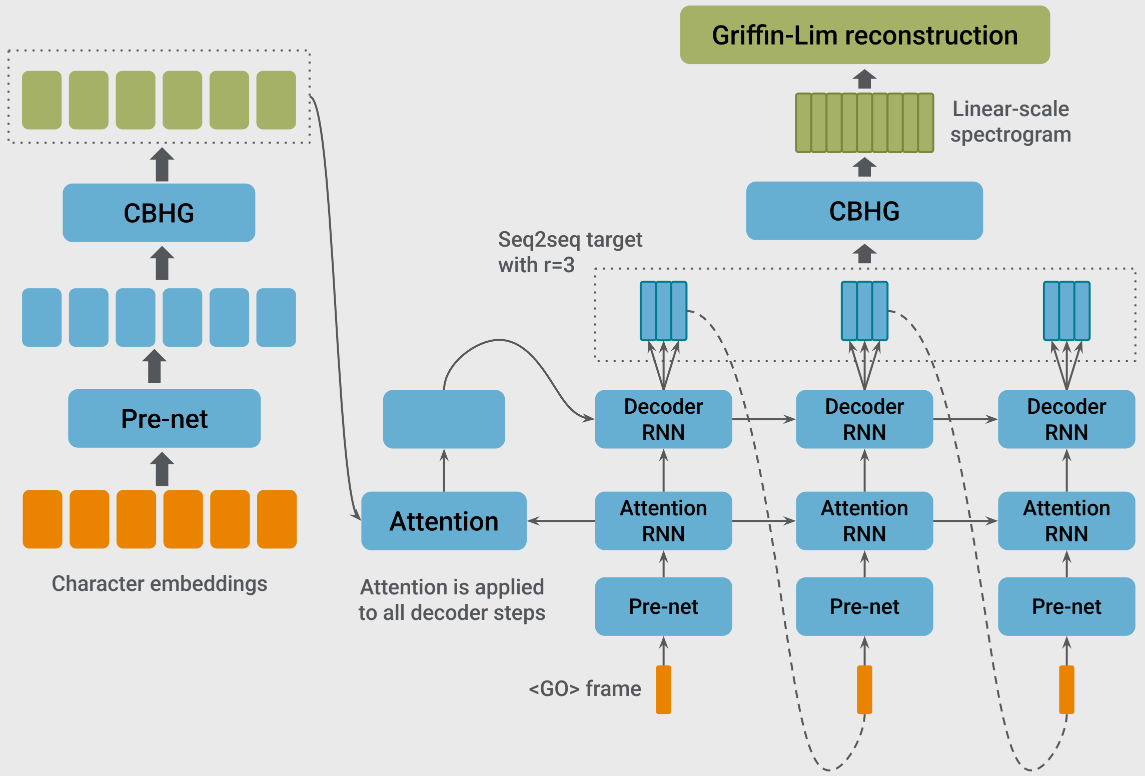


## Why use intermediate spectrograms?

- 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 – avoids deep learning model producing each wave sample value (8k/sec minimum!)

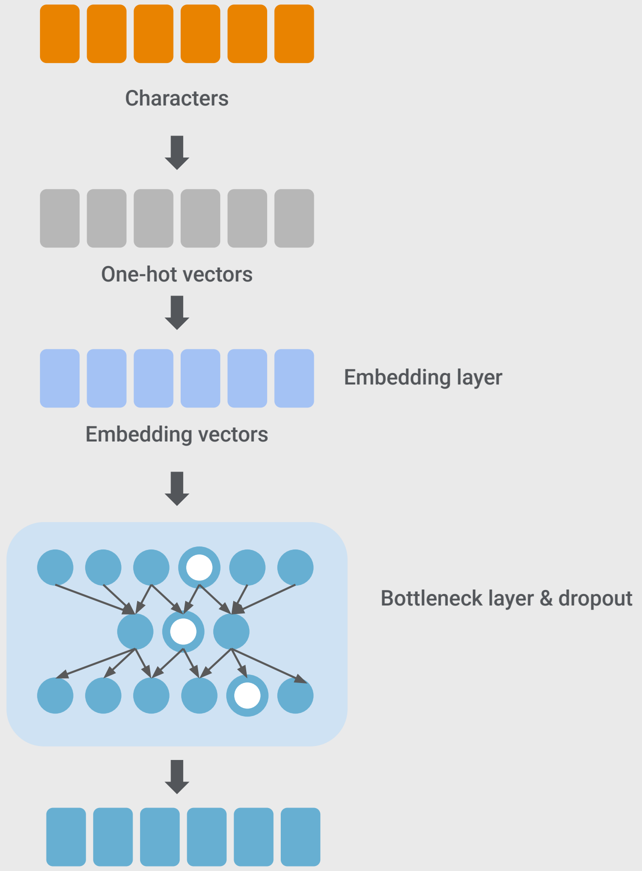
# Tacotron

- Encoder decoder model with attention
- Predicts mel spectrograms from character inputs
- “Information bottleneck” in pre-net crucial for regularization
- Older architecture. Details not relevant for modern TTS.



## Pre-Net

Maps characters into a continuous vector

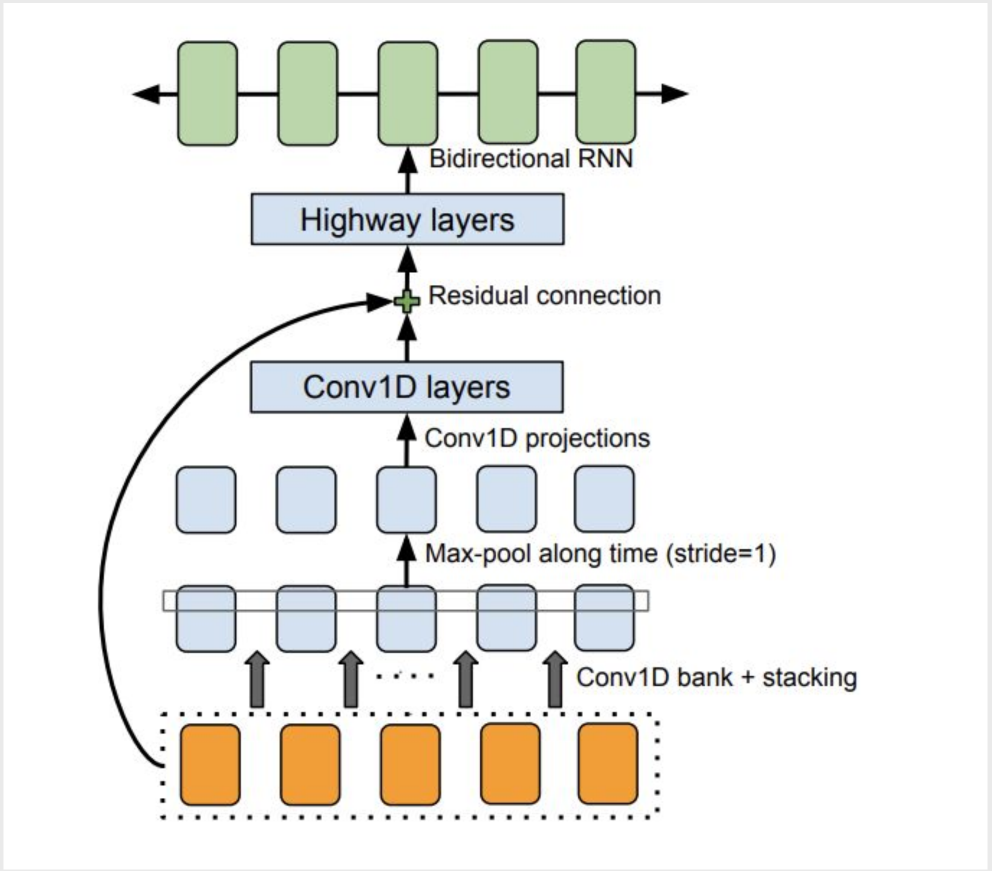


## CBHG Encoder

This is a special multi-layer RNN that includes a convolutional layer.

Mixes information 2 ways:

1. With neighboring characters via conv1d
2. Throughout entire sequence with GRU



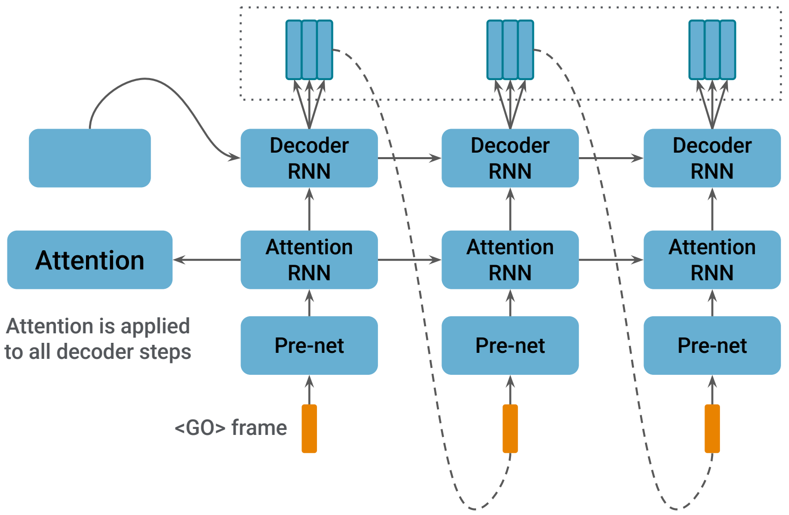
## Tacotron Decoder

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

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

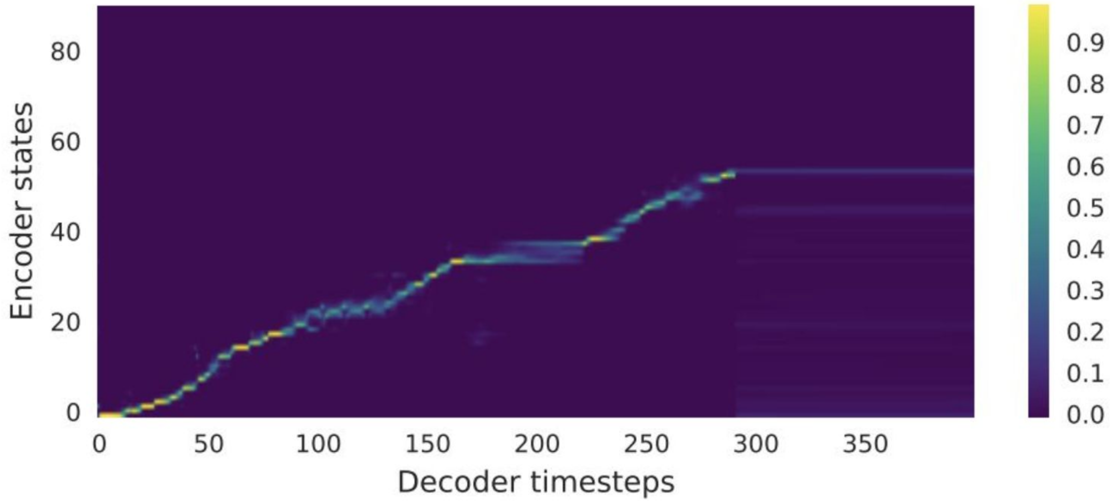
$$\boldsymbol{\alpha}_i = \text{Attention}(\boldsymbol{s}_i, \ \ldots) \qquad \boldsymbol{c}_i = \sum_j \alpha_{i,j} \boldsymbol{h}_j$$

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



Wang et al 2017

## Attention reveals alignment

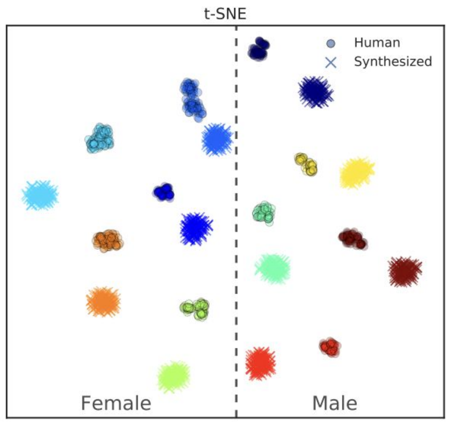


Wang et al 2017

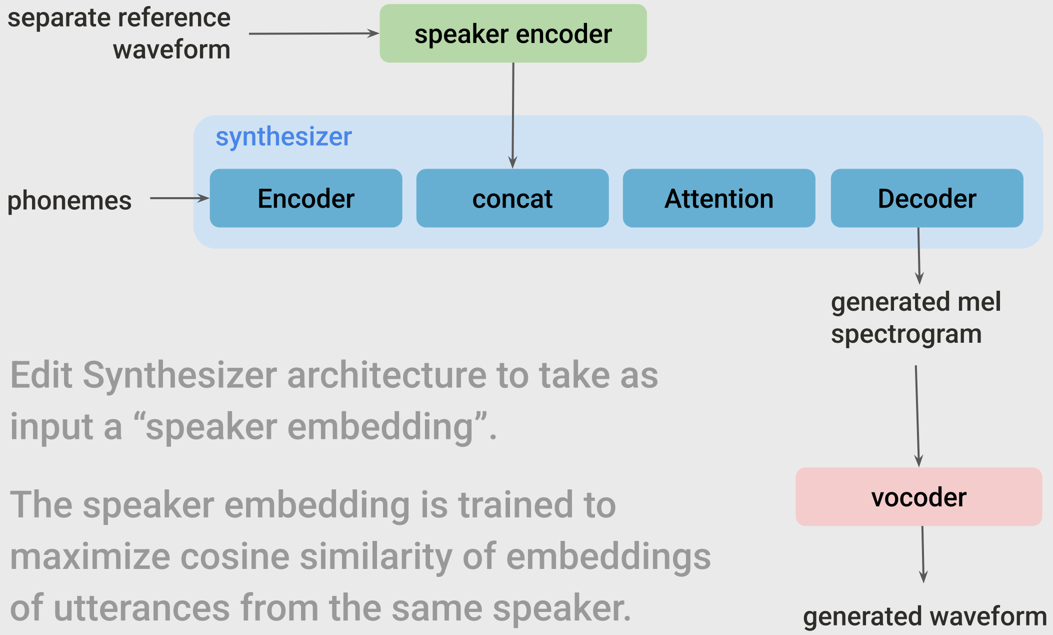
[Tacotron samples]

## Speaker and Style Embeddings

# Multispeaker TTS



Low dimensional projections of speaker embeddings



Edit Synthesizer architecture to take as input a "speaker embedding".

The speaker embedding is trained to maximize cosine similarity of embeddings of utterances from the same speaker.

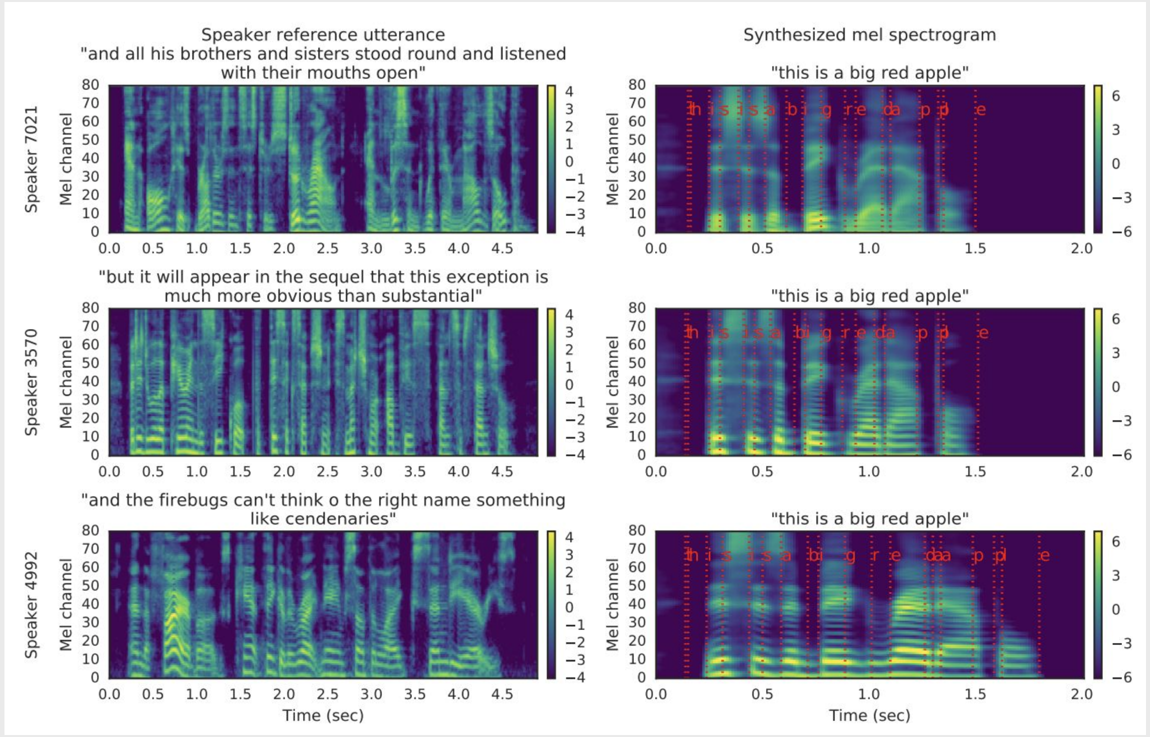
[Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis]

Speaker samples

# Multispeaker TTS

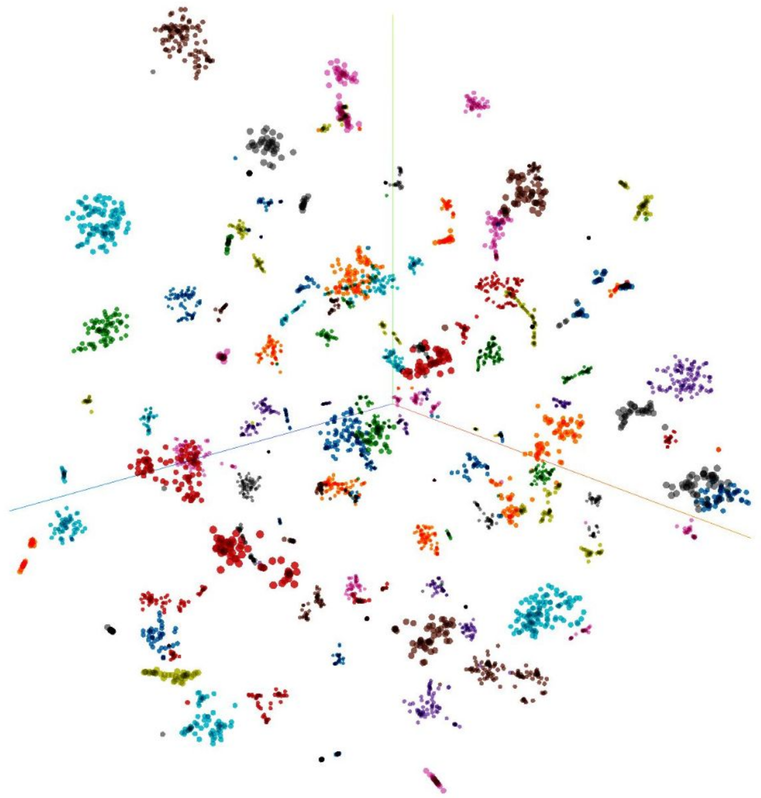
The same sentence produces different spectrograms based on the reference utterance.

Shape of the generated spectrograms are similar but some are stretched out more, representing slower speakers.



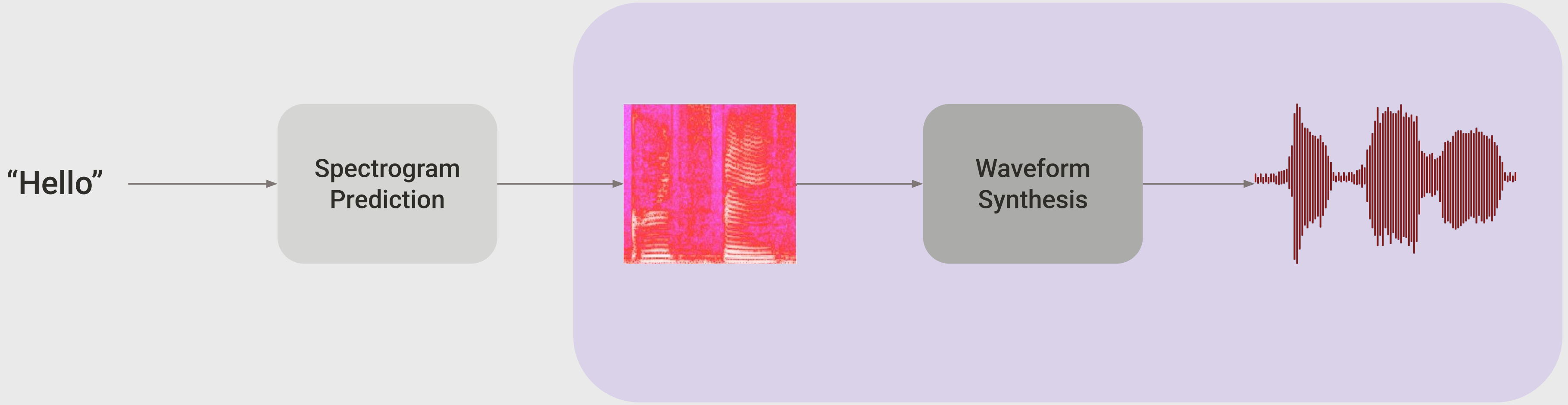
[Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis]

# Learned Speaker Embeddings

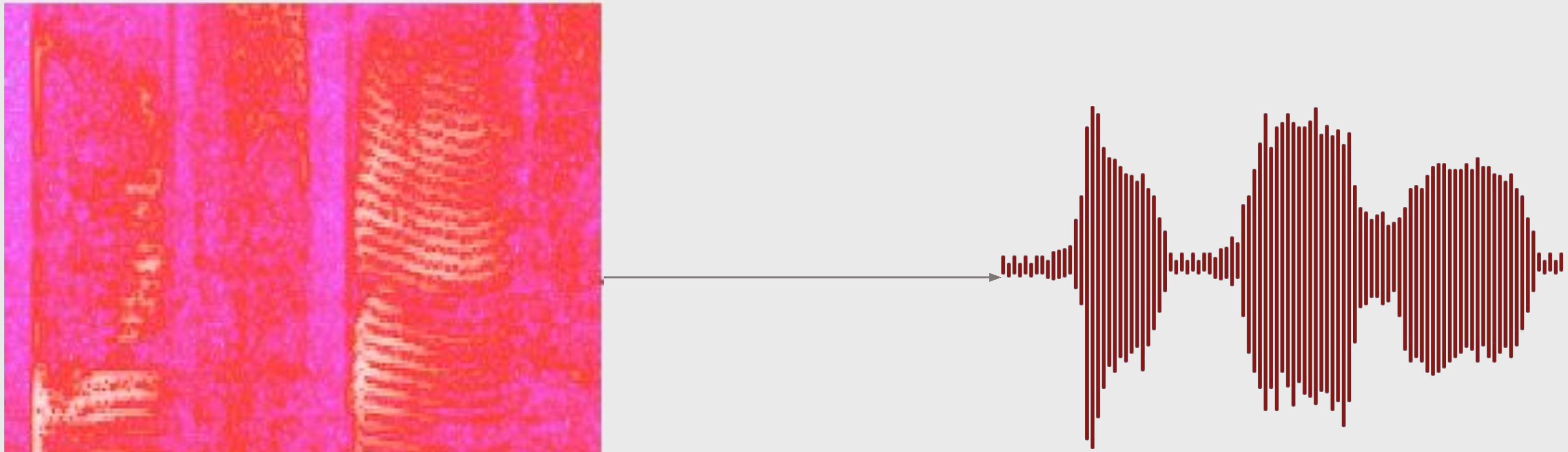


# Vocoders

## Neural TTS Paradigm: Vocoding creates waveforms



## 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, the spectrogram does not represent phase

## Griffin-Lim Wave Reconstruction

- 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 it works:
    - Start with a random prediction for the phase
    - Iteratively apply istft and stft to generate more “consistent” spectrograms
- These sound much clearer than random/zero phase in practice
- 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

## WaveNet

- One of the initial modern deep learning methods for neural net waveform synth
- Generating 16k audio samples per second is a challenge – specialized architectures often used

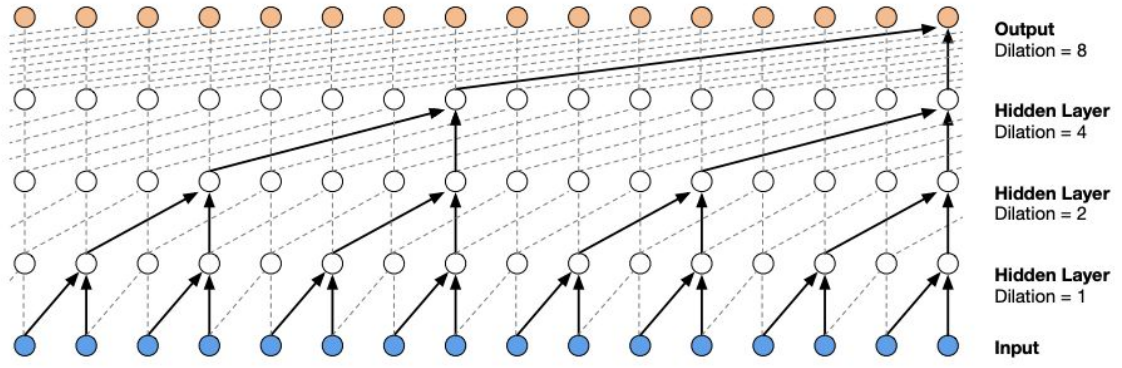
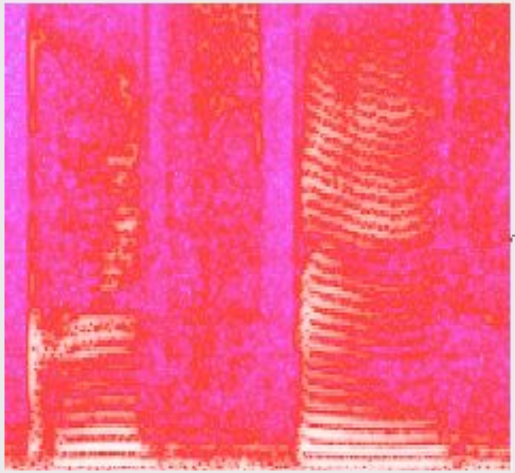


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

DeepMind Post

# WaveNet



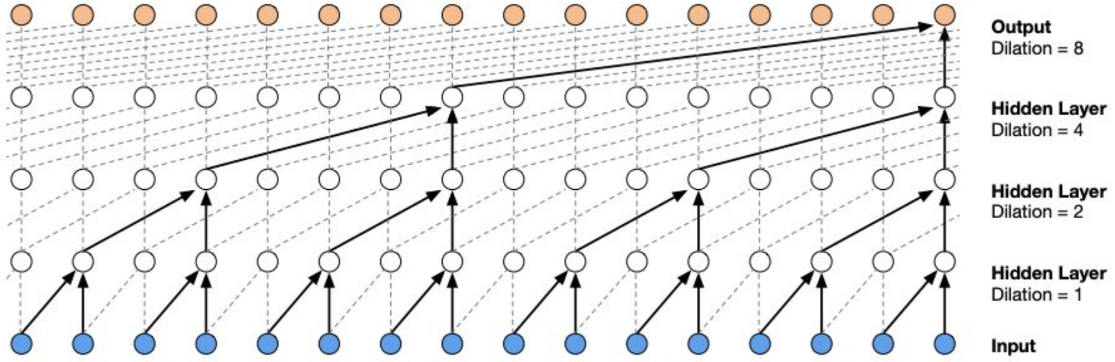
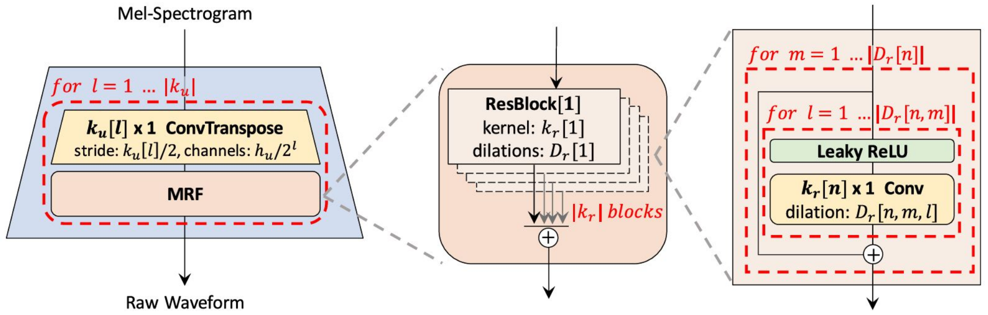


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

DeepMind Post

# HiFiGAN

- **GAN based vocoders have some of the best quality/latency trade offs currently**
  - Active development in this area as hardware and compute-efficient neural architectures improve
  - Ideally vocoders can run much faster than realtime, and on-device



Mel-Spectrogram
$for\ \ l = 1\ ...\ |k_u|$
$k_u[l]$ x 1 ConvTranspose
stride: $k_u[l]/2$, channels: $h_u/2^l$
MRF
Raw Waveform
ResBlock[1]
kernel: $k_r[1]$
dilations: $D_r[1]$
$|k_r|$ $blocks$
$for\ \ m = 1\ ...\ |D_r[n]|$
$for\ \ l = 1\ ...\ |D_r[n,m]|$
Leaky ReLU
$k_r[n]$ x 1 Conv
dilation: $D_r[n,m,l]$

Figure 1: The generator upsamples mel-spectrograms up to $|k_u|$ times to match the temporal resolution of raw waveforms. A MRF module adds features from $|k_r|$ residual blocks of different kernel sizes and dilation rates. Lastly, the $n$-th residual block with kernel size $k_r[n]$ and dilation rates $D_r[n]$ in a MRF module is depicted.

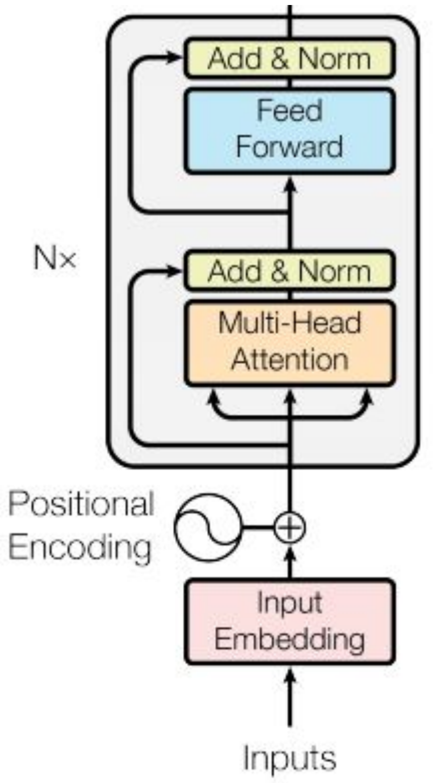
## Questions?

**Next time:** Deeper into wave generation + end-to-end text-audio models

# Appendix

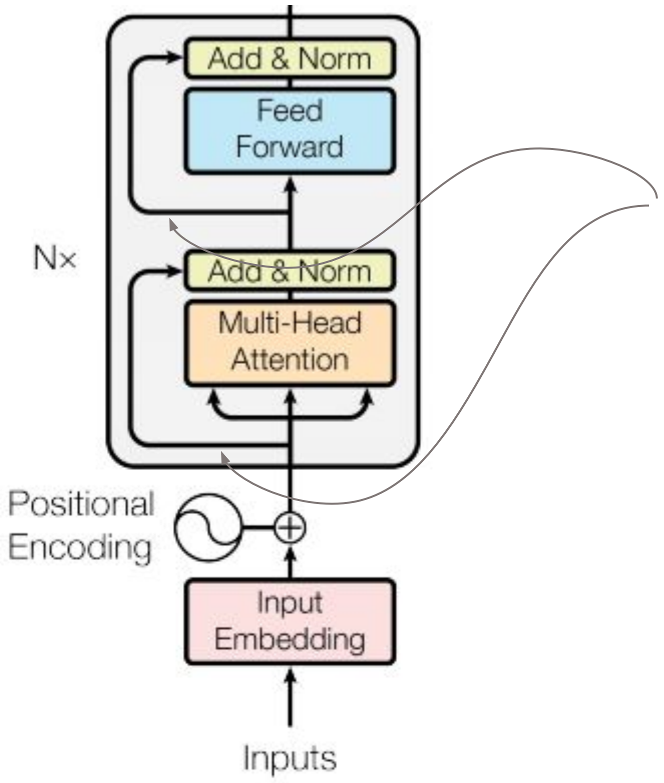
## Additional transformer explanations

# Transformer Encoder



- Residual sum
- Inputs $(e_1, e_2, \ldots, e_T)$, each $e_t$ is now a vector!
- Outputs $(c_1, c_2, \ldots, c_T)$ each $c_t \in R^d$

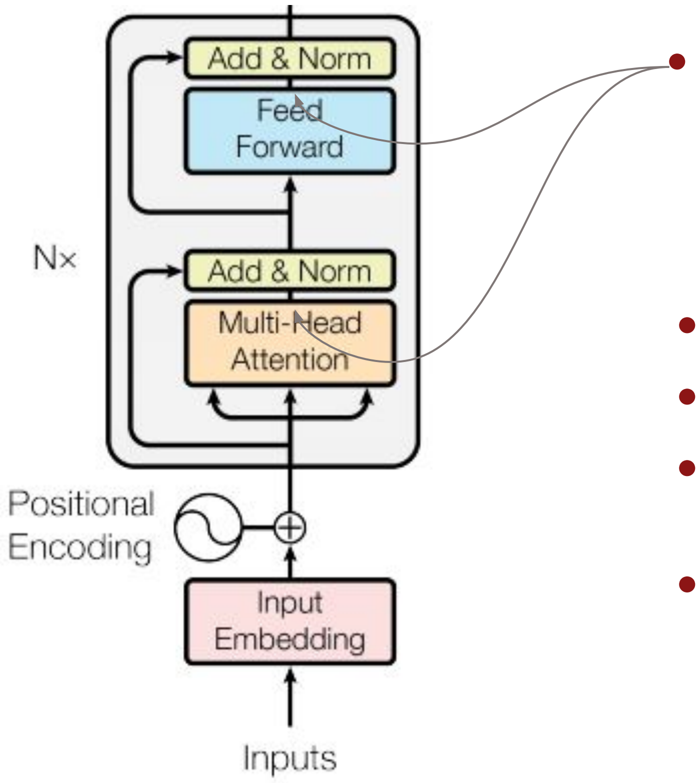
## Transformer Encoder



- We've seen this! We do self-attention with H heads on the lookup embeddings.

- $(e_1, e_2, \ldots, e_T)$ inputs (spectrogram features)
- $(c_1, c_2, \ldots, c_T)$ attention outputs
- $h_t^{res} = c_t + e_t$ residual output

# Transformer Encoder

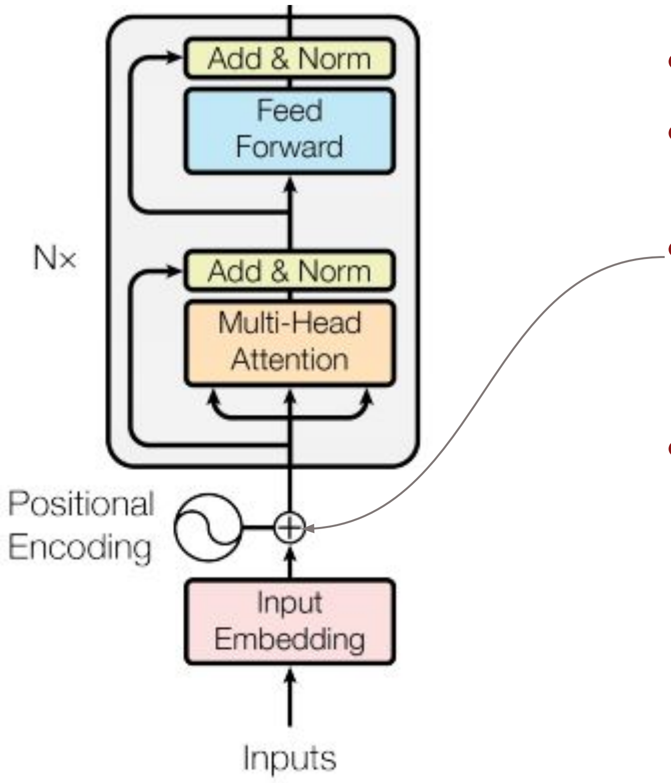


- Layer normalization

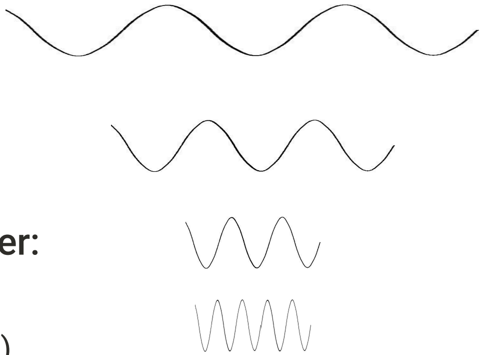
$$h^{norm} = \frac{h^{res} - E[h^{res}]}{\mathrm{sqrt}(\mathrm{Var}[h^{res}] + \varepsilon)} * \gamma + \square$$

- Note $\{\gamma, \square\} \subseteq \theta$ e.g. learnable parameters
- hres = (h1res, h2res, …, hTres)
- The mean and variance are over the sequence of size T.
- Not like batch norm (which is over a batch of examples). This is only on 1 example.

# Transformer Encoder

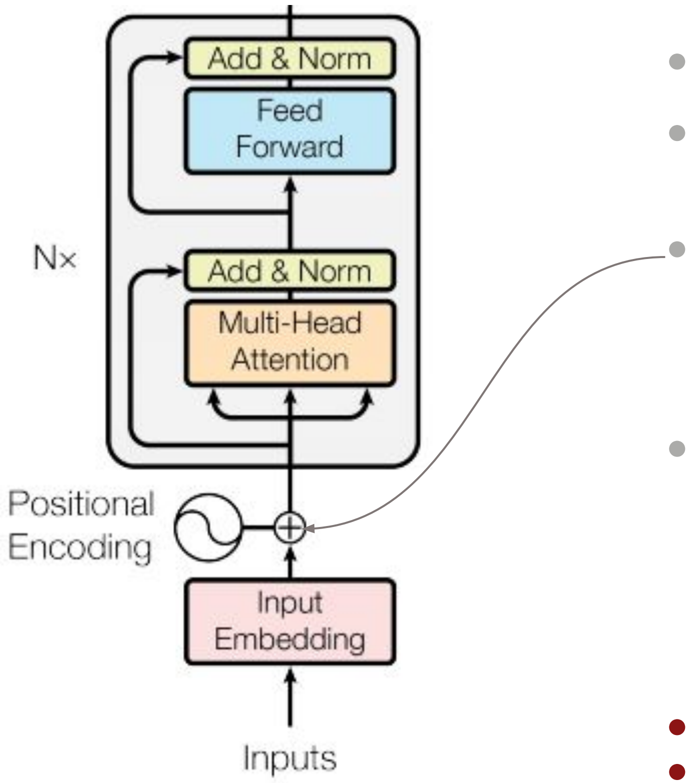


- Unlike RNNs, transformers have no order!
- But speech is left-to-right so it might be useful to tell the model that
- Positional Encodings
  - Input: $(x_1, x_2, x_3, \ldots, x_T)$
  - Position: (1, 2, 3, …, T)



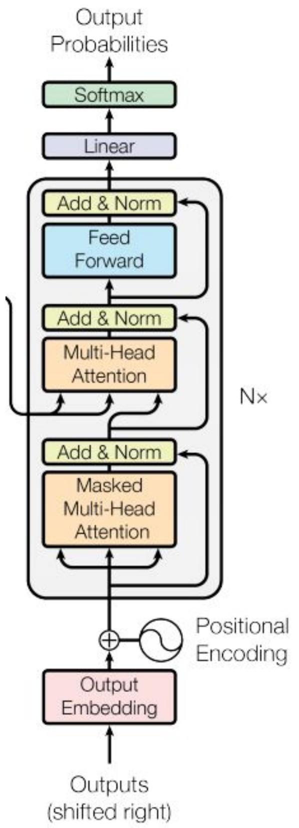
- But we can be a bit more clever:
  - $PE(t, 2i) = \sin(t/10000^{2i/d})$
  - $PE(t, 2i+1) = \cos(t/10000^{2i/d})$
  - $PE(t) = [PE(t, 0), PE(t, 1), \ldots, PE(t, d)]$
  - Add embedding of t-th token $e_t = e_t + PE(t)$

## Transformer Encoder



- Unlike RNNs, transformers have no order!
- But speech is left-to-right so it might be useful to tell the model that
- Positional Encodings
  - Input: $(x_1, x_2, x_3, \ldots, x_T)$
  - Position: $(1, 2, 3, \ldots, T)$
- But we can be a bit more clever
  - $PE(t, 2i) = \sin(t/10000^{2i/d})$
  - $PE(t, 2i+1) = \cos(t/10000^{2i/d})$
  - $PE(t) = [PE(t, 0), PE(t, 1), \ldots, PE(t, d)]$
  - Add embedding of t-th token $e_t = e_t + PE(t)$
- **Assigns every timestep a unique waveform**
- **No need to specify maximum length**

## Transformer Decoder



- **Masked Multi-head Attention:**
  - We can't do exactly what we do in the encoder b/c we don't want to bleed future info.

$$(x_1, x_2, \ldots, x_{t-1}, x_t, \underbrace{x_{t+1}, \ldots, x_{T-1}, x_T})$$

Current

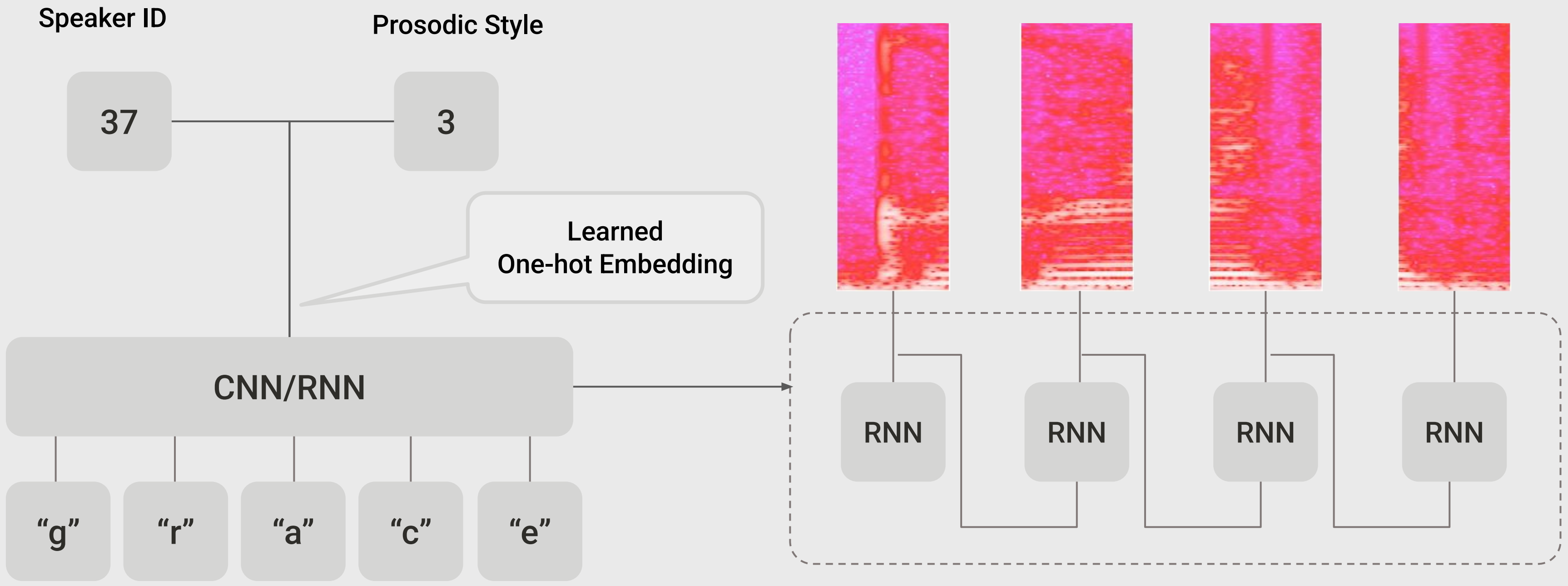
- **Cheating if we see this b/c in test time, we don't have access to > t+1**

## More on style and speaker embeddings

# Speaker/Style with One Hot Labels

- The same text has infinitely many voicings
- Controllable speaker and prosody is very useful in dialog systems and elsewhere
- 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

# Sequence to Sequence Problem

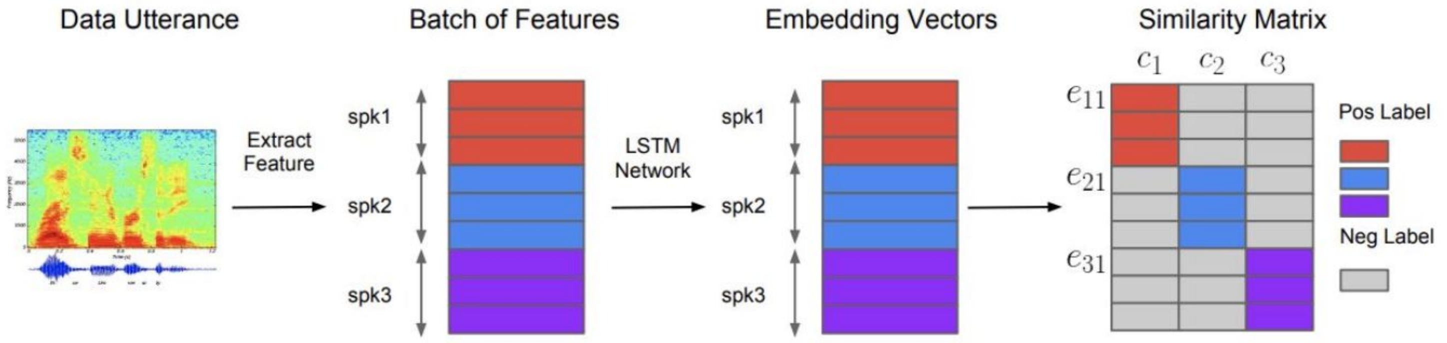


## Learned Speaker Embeddings

- 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

# Learned Speaker Embeddings



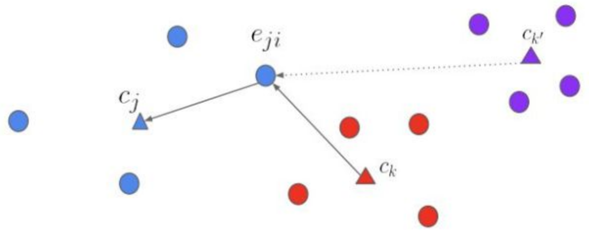
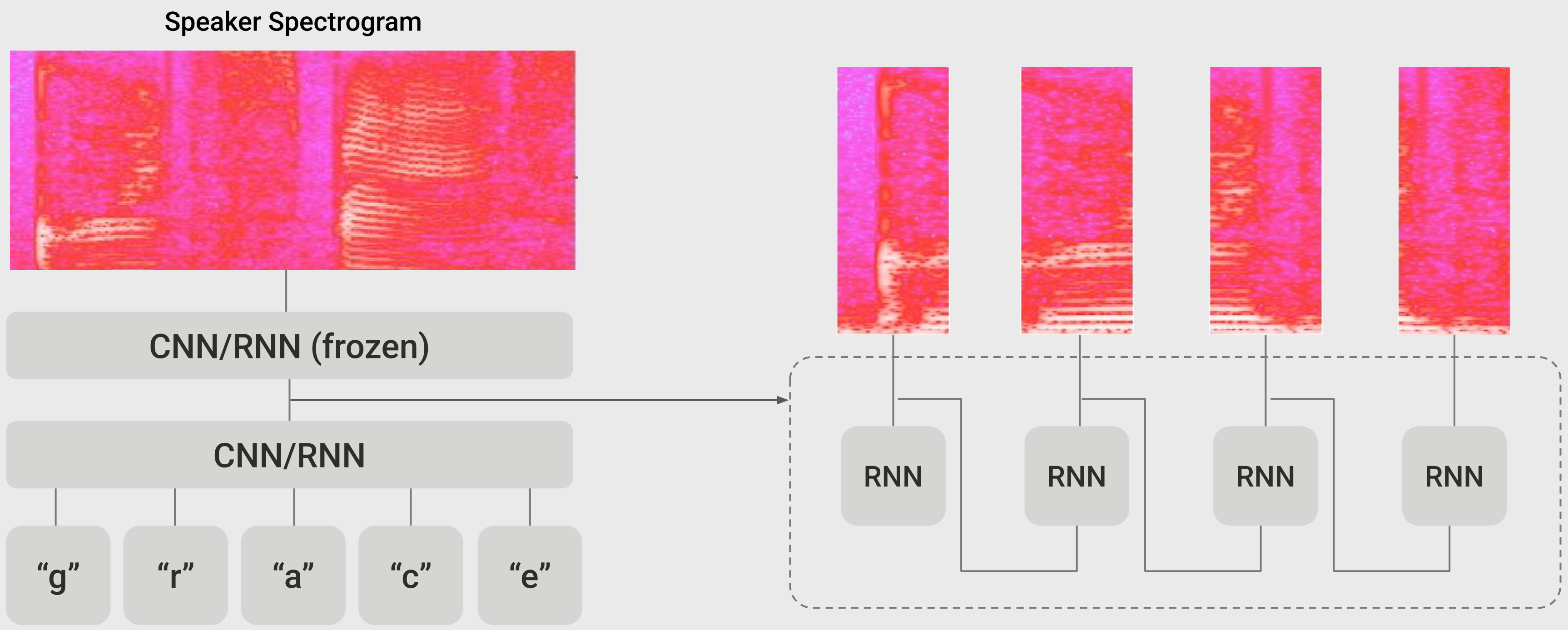


**Figure 1:** System overview. Different colors indicate utterances / embeddings from different speakers.

**Figure 2:** GEE loss pushes the embedding towards the centroid of the true speaker, and away from the centroid of the most similar different speaker. Wang et al 2017

## Learned Speaker Embeddings



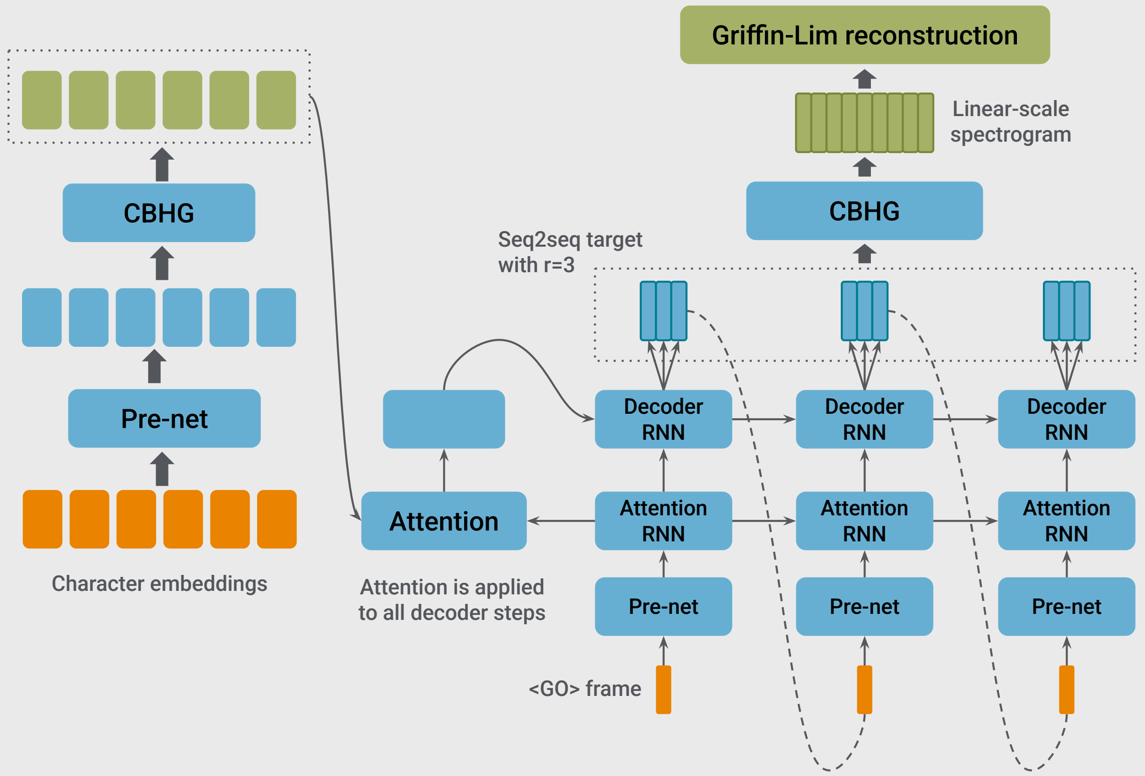
## Learned Style Embeddings

- 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

## Tacotron steps

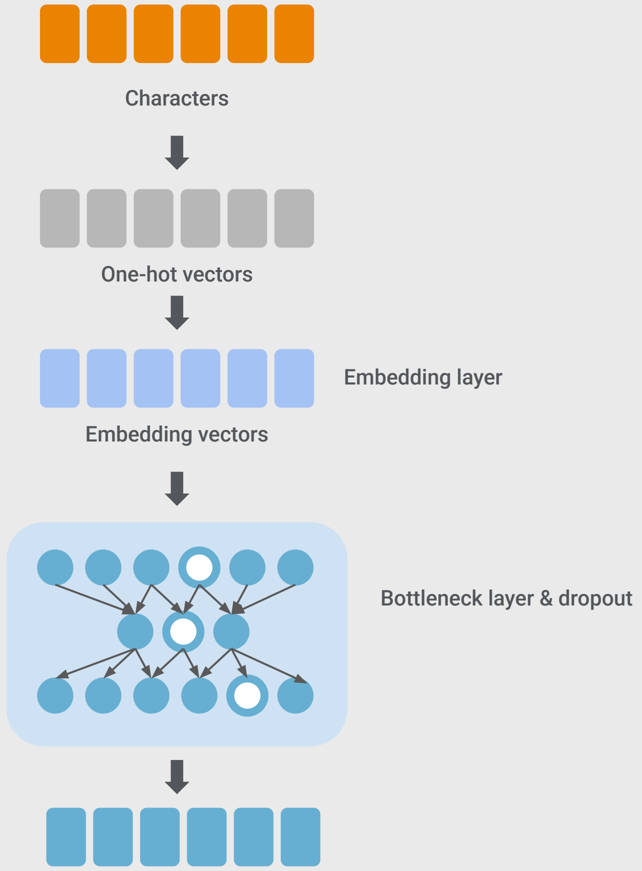
# Tacotron

- Encoder decoder model with attention
- Predicts mel spectrograms from character inputs
- "Information bottleneck" in pre-net crucial for regularization
- Older architecture. Details not relevant for modern TTS.



## Pre-Net

Maps characters into a continuous vector

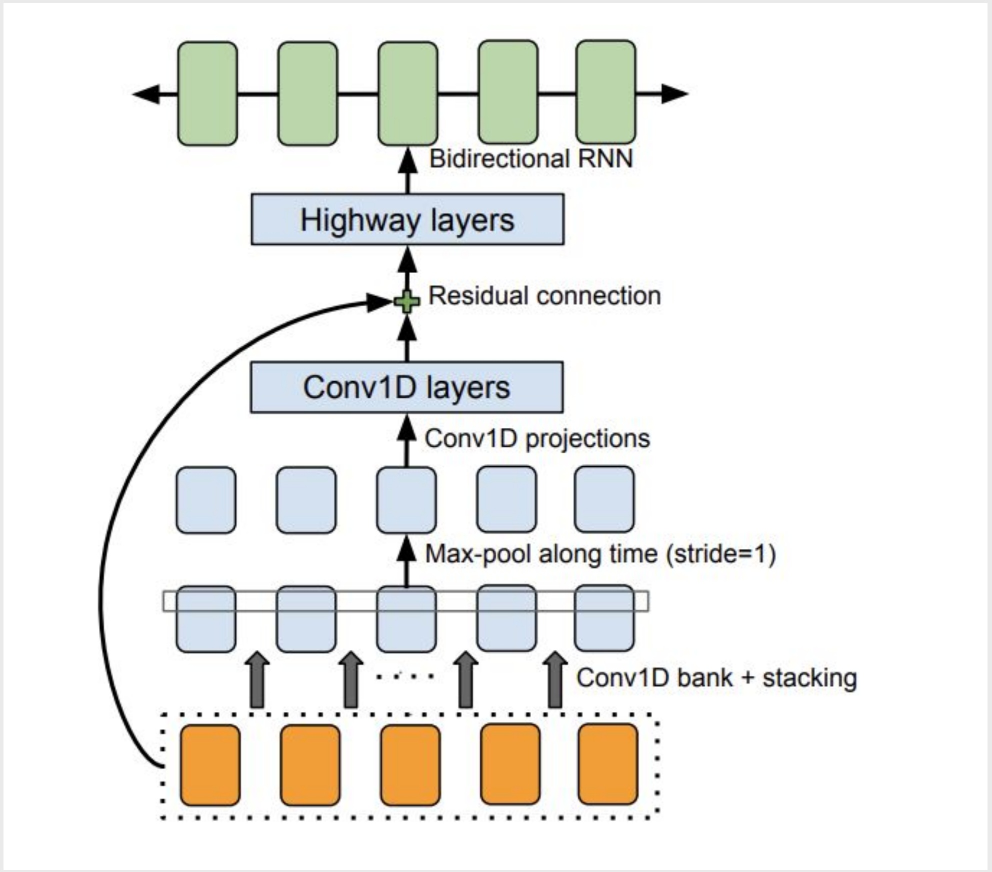


## CBHG Encoder

This is a special multi-layer RNN that includes a convolutional layer.

Mixes information 2 ways:

1. With neighboring characters via conv1d
2. Throughout entire sequence with GRU



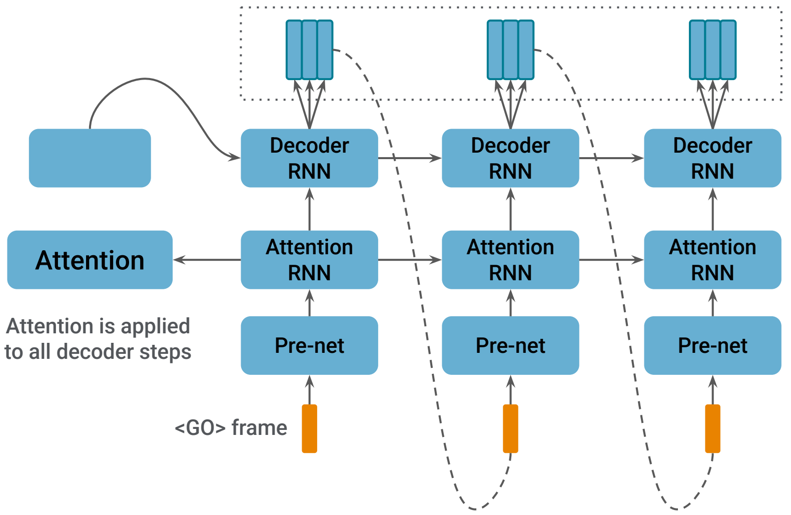
## Tacotron Decoder

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

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

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

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



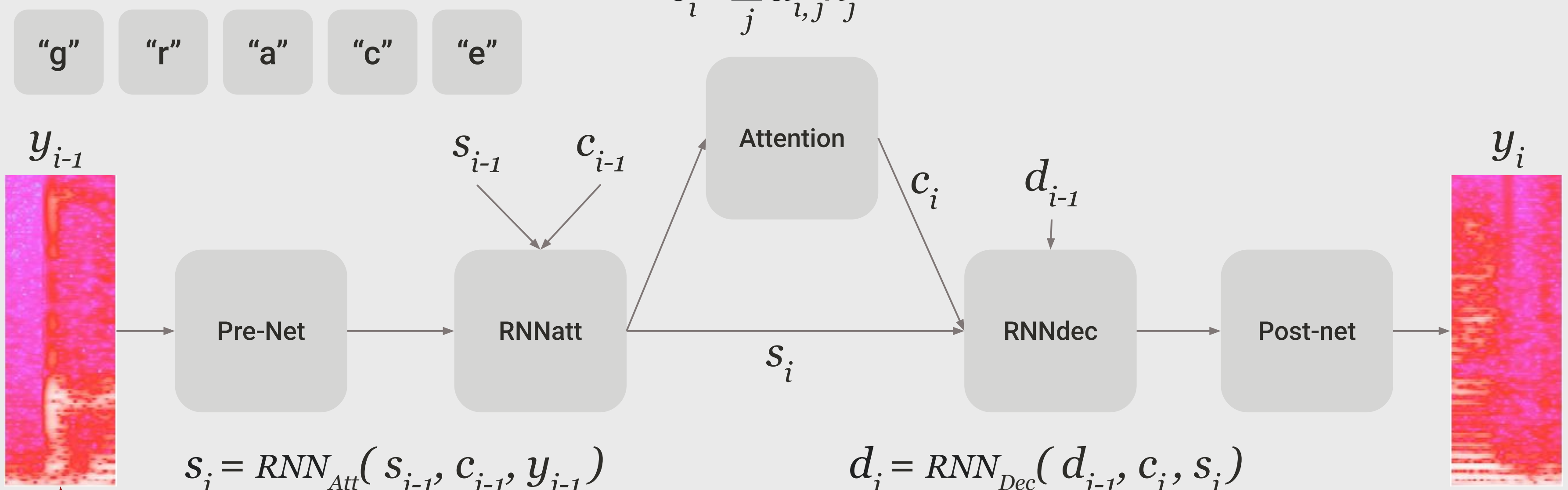
Wang et al 2017

Optionally takes previous alignment, encoder states

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

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

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



$$s_i = RNN_{Att}(s_{i-1}, c_{i-1}, y_{i-1})$$

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

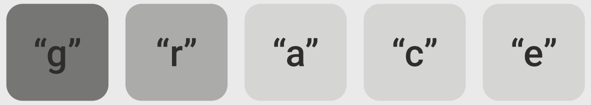
Can be 1-5 mel frames (reduction factor)

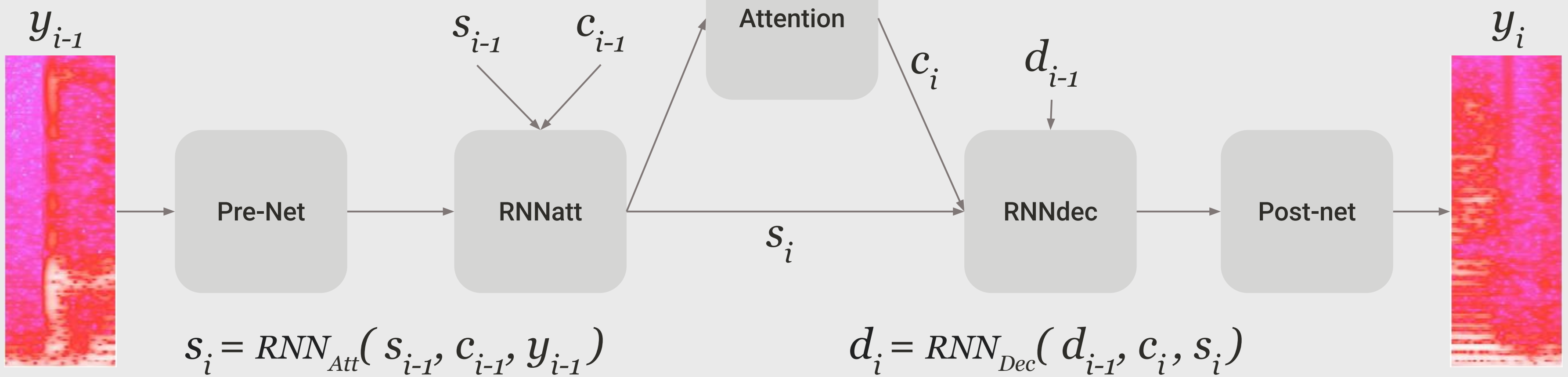
# j = 0

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

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

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





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

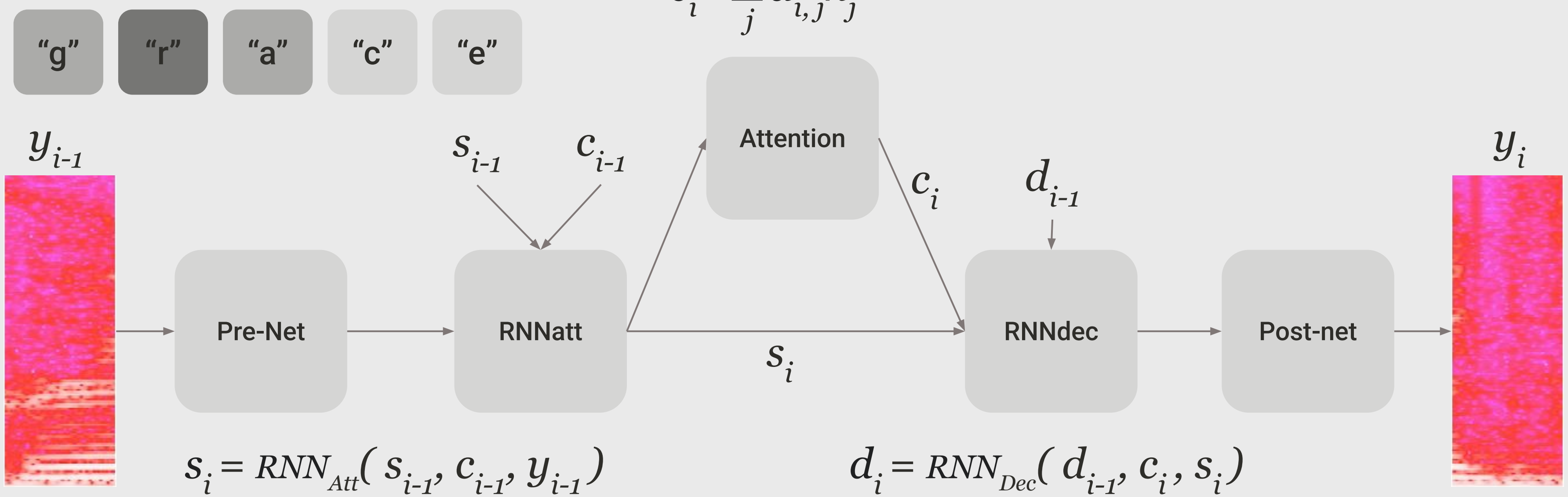
$$d_i = RNN_{Dec}(\, d_{i-1}, c_i, s_i)$$

## j = 1

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

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

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



$s_i = RNN_{Att}(s_{i-1}, c_{i-1}, y_{i-1})$

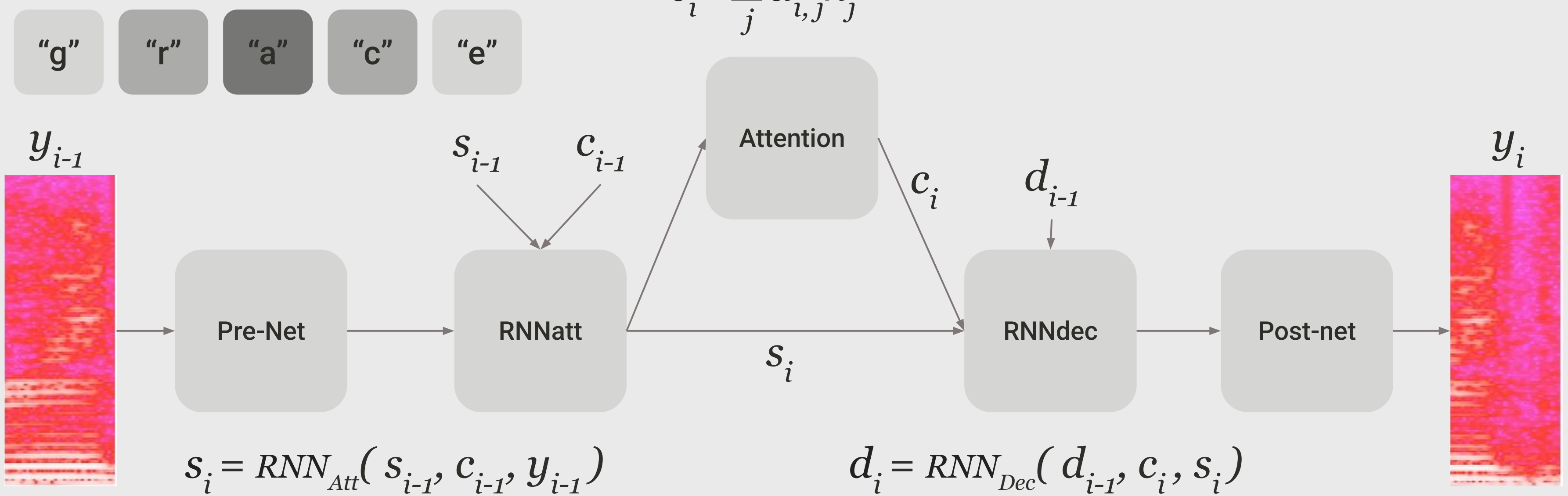
$d_i = RNN_{Dec}(d_{i-1}, c_i, s_i)$

## j = 2

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

$$\alpha_i = \text{Attention}\left(s_i, \ldots\right)$$

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



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

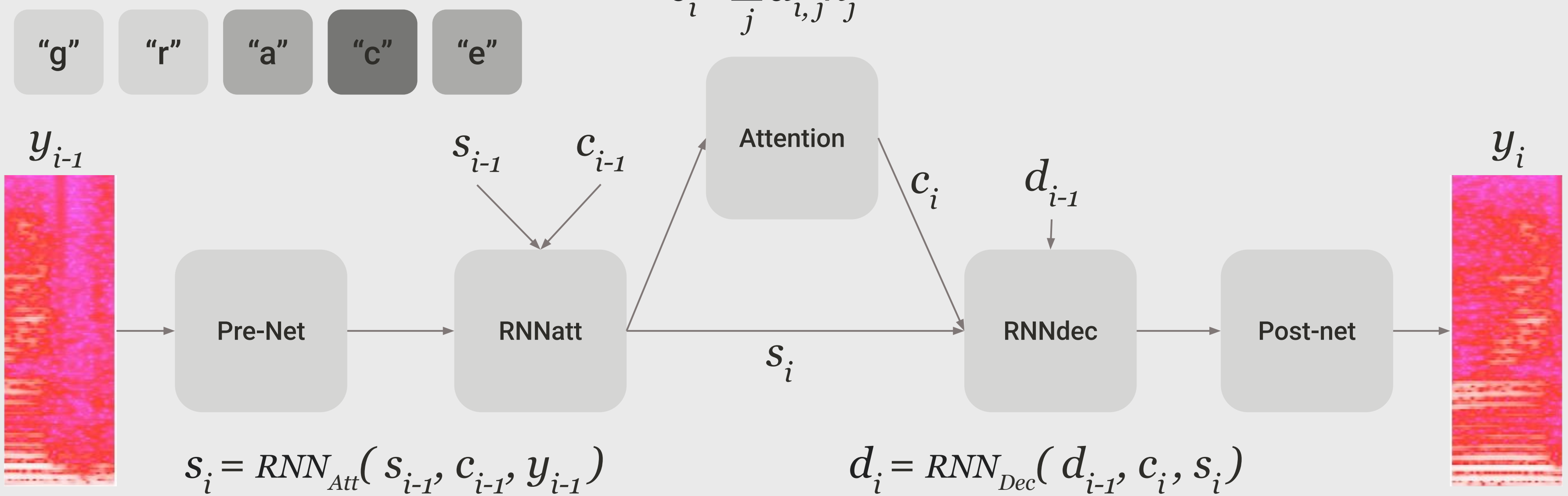
$$d_i = RNN_{Dec}\left(d_{i-1}, c_i, s_i\right)$$

# j = 3

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

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

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



$s_i = RNN_{Att}(s_{i-1}, c_{i-1}, y_{i-1})$

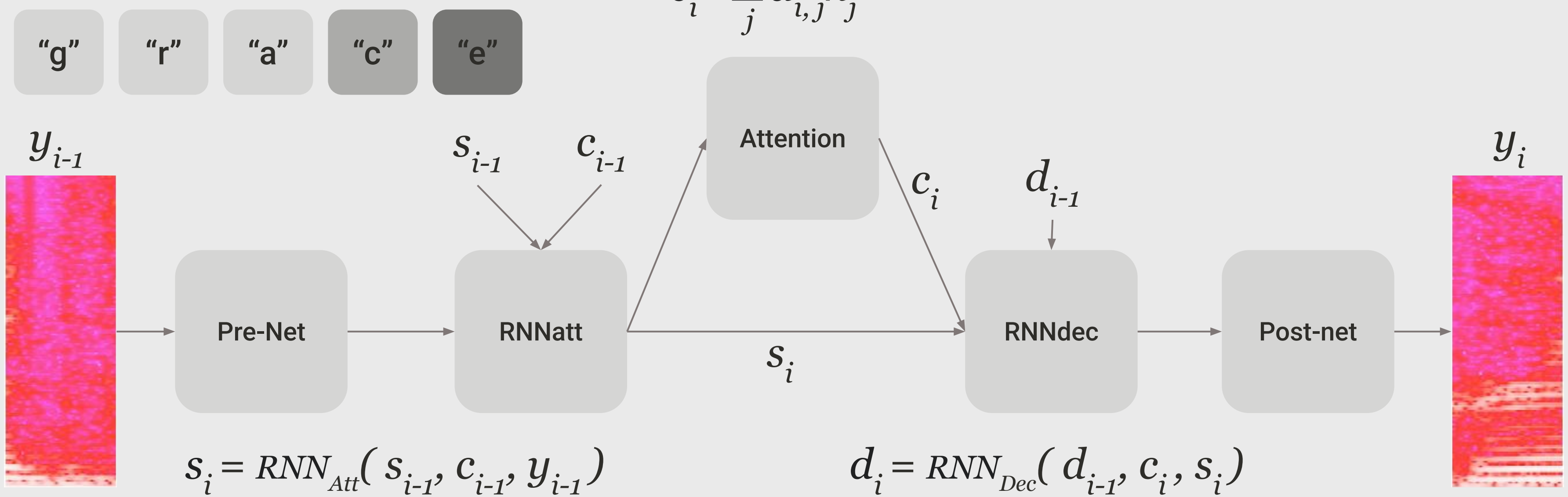
$d_i = RNN_{Dec}(d_{i-1}, c_i, s_i)$

## j = 4

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

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

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



$s_i = RNN_{Att}(s_{i-1}, c_{i-1}, y_{i-1})$

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

# GST Tacotron

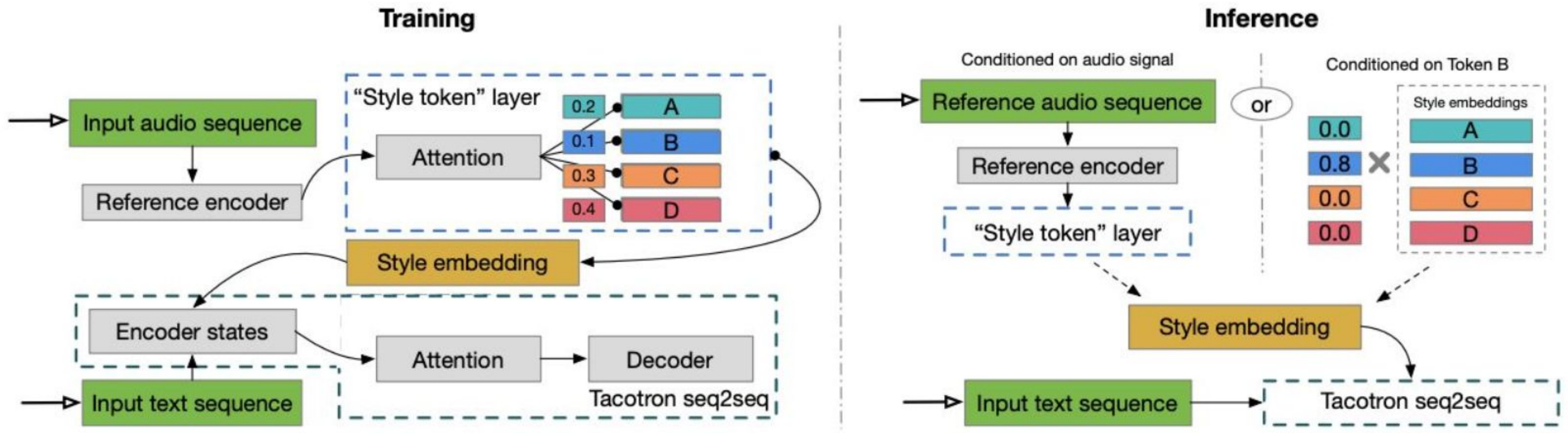


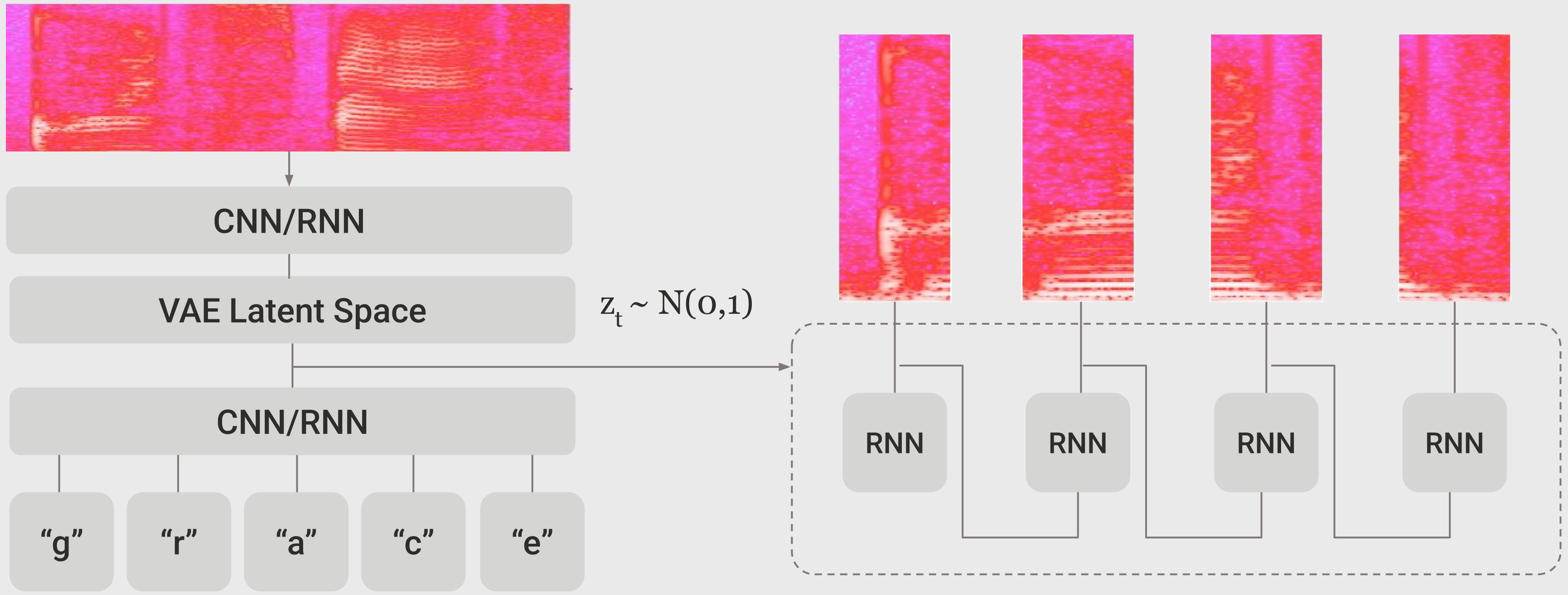
**Figure:** Model diagram. During training, the log-mel spectrogram of the training target is fed to the reference encoder followed by a style token layer. The resulting style embedding is used to condition the Tacotron text encoder states. During inference, we can feed an arbitrary reference signal 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 2017 | Samples

## VAE Tacotron

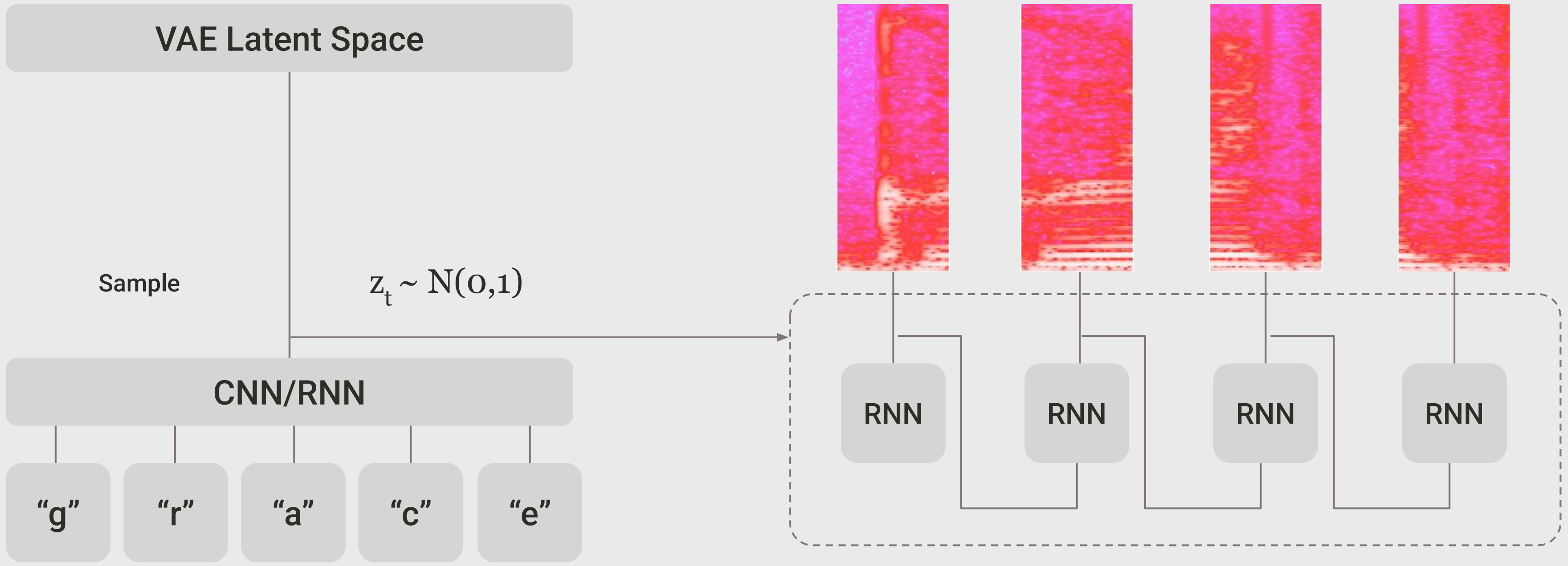
- Use variational auto encoder for style latent space
- Latent space then encouraged to follow a gaussian distribution
- Sample prosodies from latent space at inference time
- GMVAE Tacotron uses a hierarchical mixture of gaussians so each component learns a different prosodic component of the data
- 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

## Training


CNN/RNN
VAE Latent Space
$z_t \sim N(0,1)$
CNN/RNN
"g"
"r"
"a"
"c"
"e"
RNN
RNN
RNN
RNN

## Inference

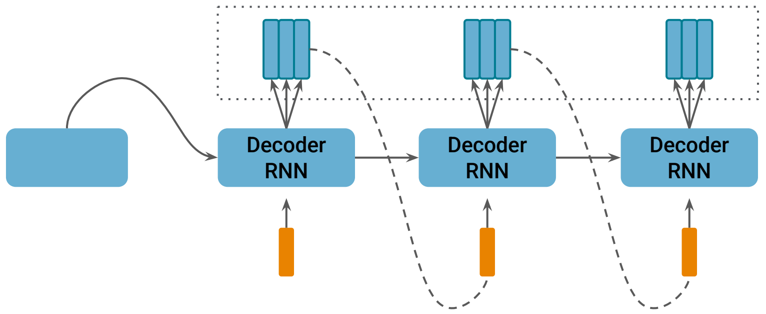

VAE Latent Space
Sample
$z_t \sim N(0,1)$
CNN/RNN
"g"
"r"
"a"
"c"
"e"
RNN
RNN
RNN
RNN

# Deep dive: FastSpeech 1 & 2

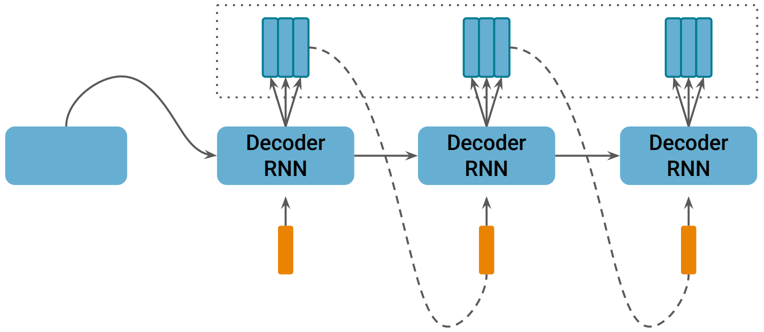
# Motivation

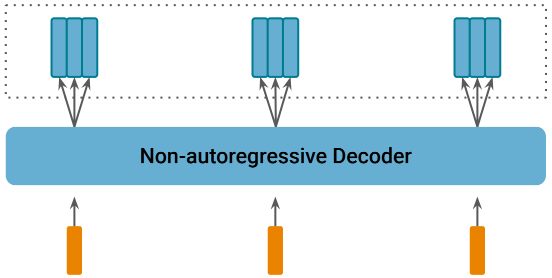
Disadvantages of auto-regressive generation:

- *Slow:* It is by definition step by step. Spectrogram sequences can be 1000s of steps long.
- Error prone: If an error happens on step 5, then it affects steps 6 to 1000.



## Motivation





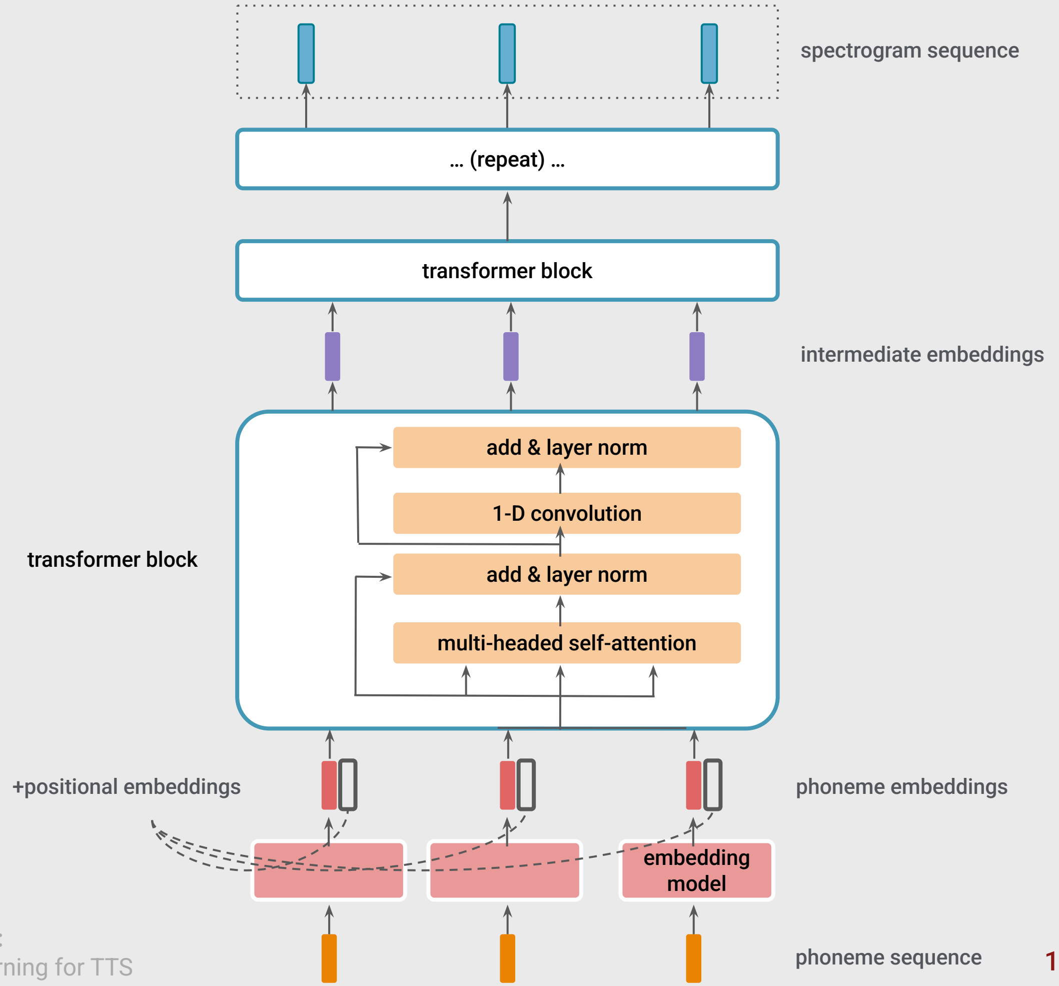
What if we could do this?

- ***Fast***: everything happens at once.
- **Less error prone**: no forward propagation.

# Feed forward Transformers

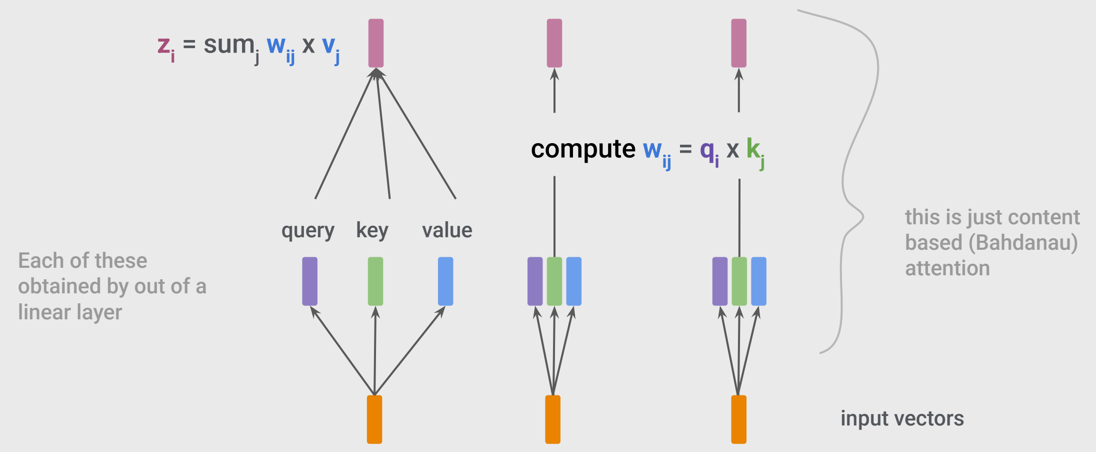
Borrow recent NLP breakthroughs for non-autoregressive modeling: **transformers**.

- Same architecture as LLMs e.g. GPT-4, Llama, etc.
- Stacked attention layers to mix information in the input phonemes
- Each phoneme input is mapped to a predicted mel-spectrogram



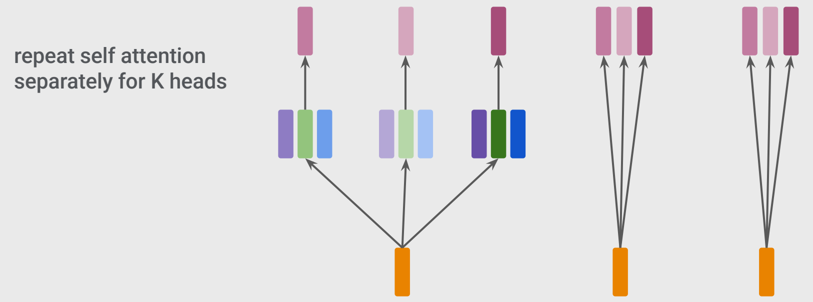
## Self attention

A self attention layer mixes information between a sequence of embeddings.


$z_i$ = $\text{sum}_j\ w_{ij} \times v_j$
compute $w_{ij} = q_i \times k_j$
query
key
value
Each of these obtained by out of a linear layer
this is just content based (Bahdanau) attention
input vectors

## Multi-headed attention

Goal is to provide diversity in what is being focused on

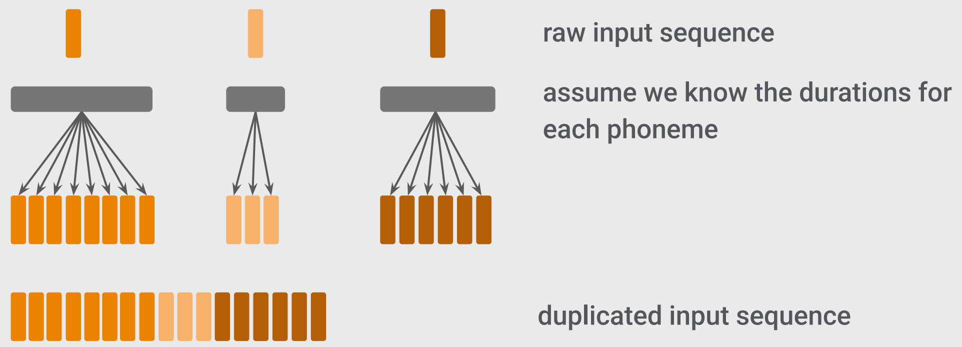


# Length Regulator

One of the properties of a transformer is that a sequence of N input tokens produces N output tokens.

**But this doesn't work for us!** Spectrogram sequences are often much longer than phoneme sequences.

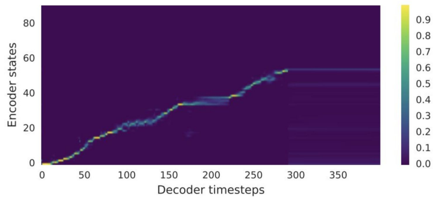
The trick is that we will **duplicate** the phonemes based on their duration. This way, a longer lasting phoneme will produce a longer sequence of spectrograms.


raw input sequence
assume we know the durations for each phoneme
duplicated input sequence

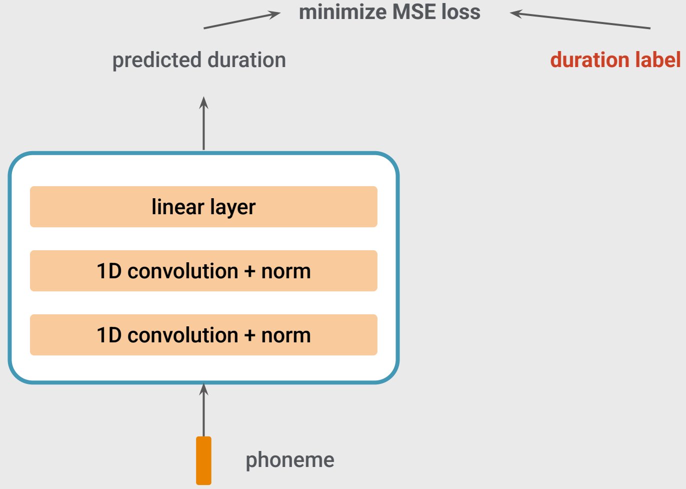
## Duration Predictor

To do the length correction, we need to know the duration from phonemes alone.

How do we do this?



Train a separate model jointly to predict the length of the mel-spectrograms for each phoneme.

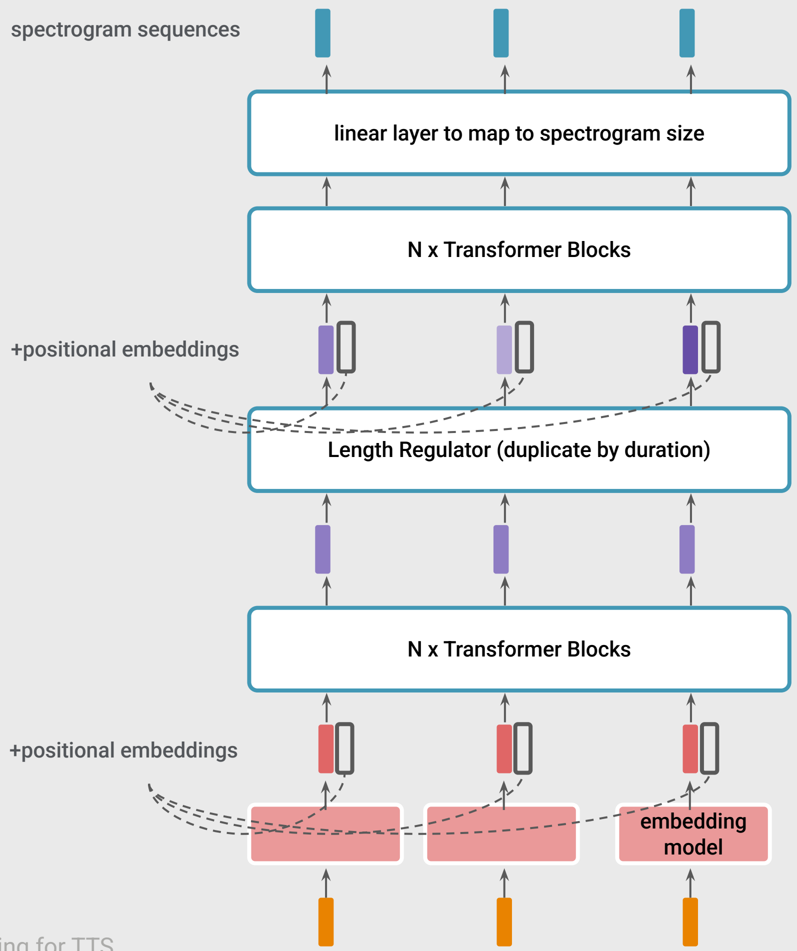


How do we get the duration label?

HACK: take a pretrained autoregressive TTS model, and estimate duration using attention from phonemes to spectrograms. Use that as label for this model.

# FastSpeech

Put together all the components and stack transformer layers.



## 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

Fastspeech2 samples

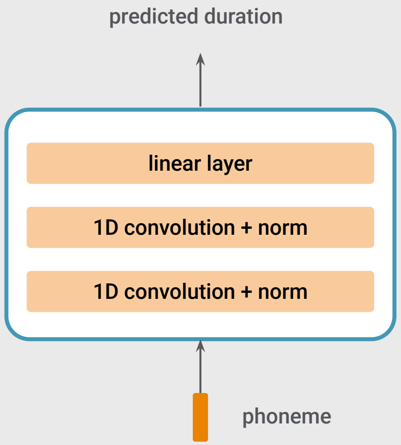
# Variance Adaptors

Decide “variance information”

- Duration
- Pitch
- Energy

using the phoneme so that the one-to-many problem becomes one-to-one.

## Duration / Energy / Pitch Predictor



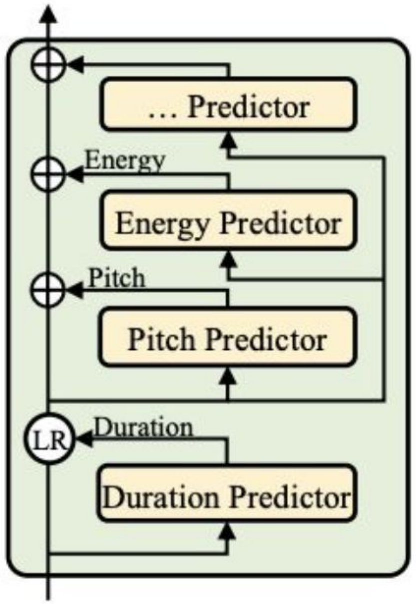
For each of three, we train a separate small model to predict them from the phoneme sequence. The labels are extracted using deterministic tools.

- **Duration** Use the Montreal forced alignment tool rather than a pretrained autoregressive model.
- **Energy** Treat $L_2$ norm of amplitude of the STFT of the frame as label.
- **Pitch** Use continuous wavelet transforms (CWT) to produce pitch spectrograms.

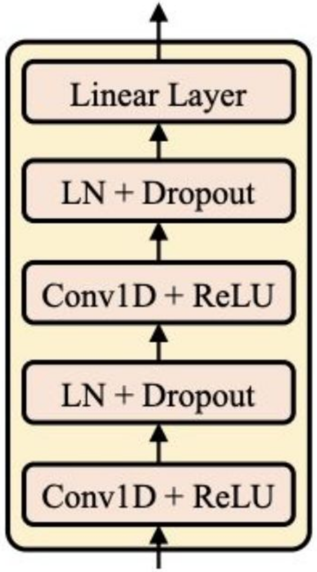
## FastSpeech 2 Variance Predictors

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

(FFT = Feed Forward Transformer not Fast Fourier Transform)



Variance Predictor Structure:

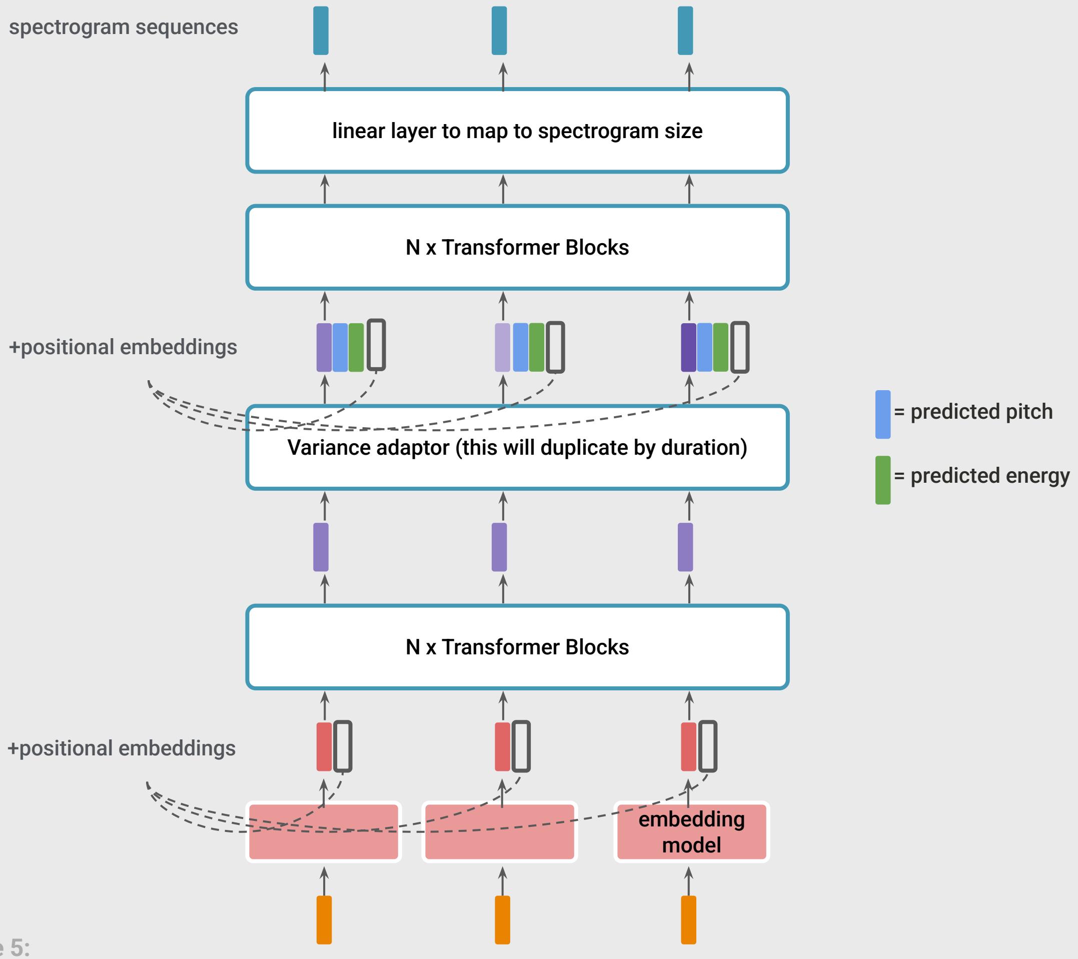


Ren et al. 2020

# FastSpeech2

Add extra variance information to the phoneme embeddings by concatenating them.

- No more reliance on a pretrained model.
- Extracts auxiliary labels from phoneme itself.



# 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
- More robust to silence in training data

## 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