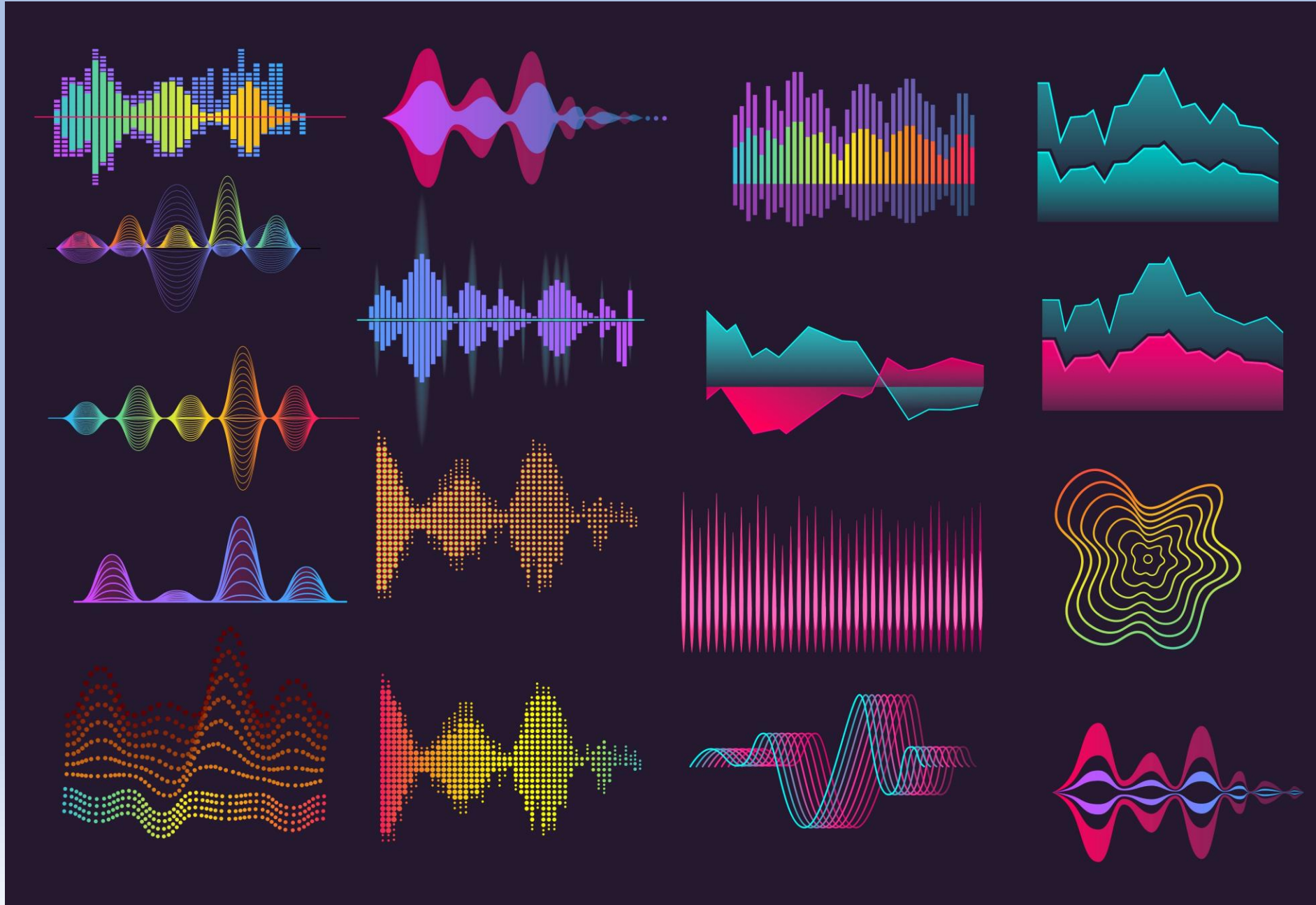
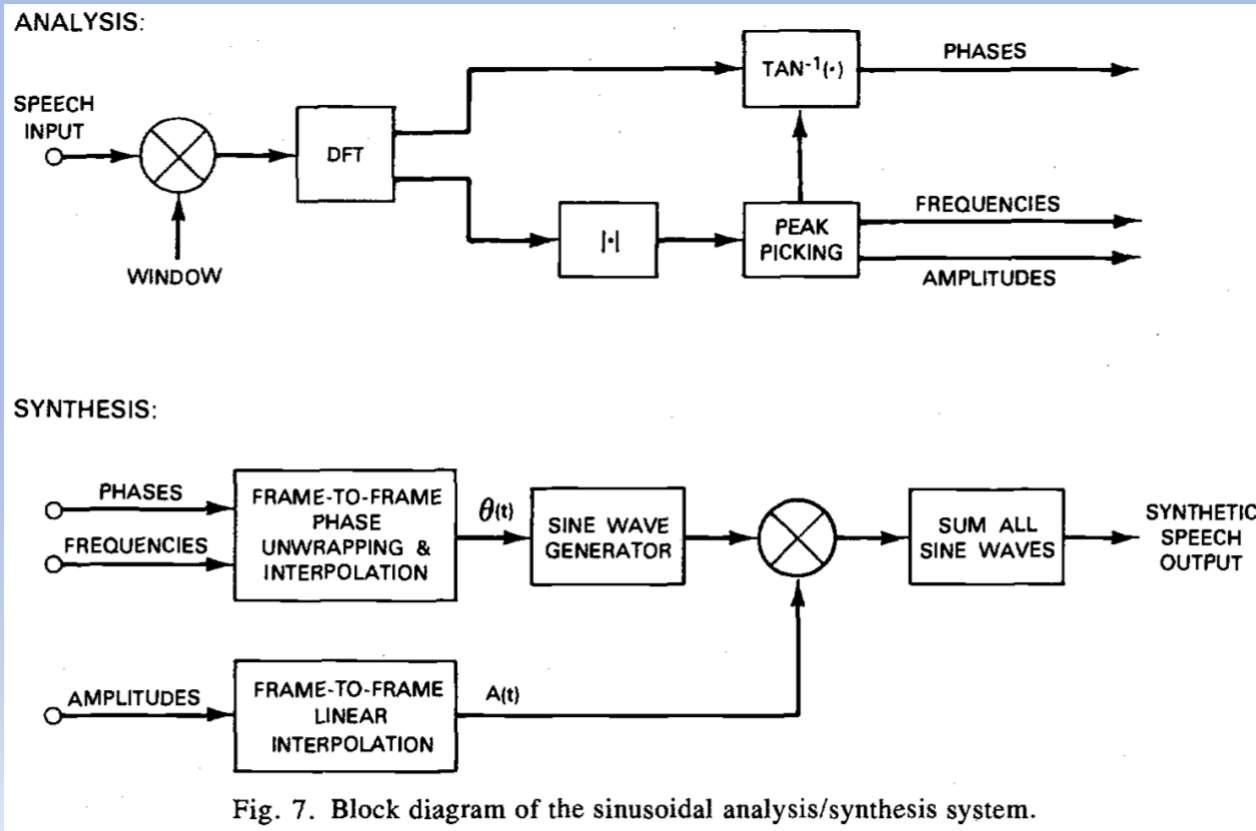


# Neural-Based Sinusoidal Vocoder



# Sinusoidal Model

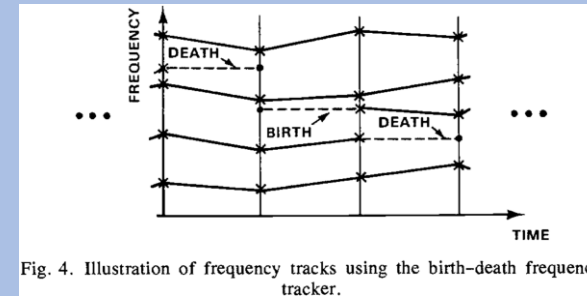
$$s[n] \approx \hat{s}[n] = \sum_{l=1}^{L(k)} \hat{A}_l^k \cos(n\hat{\omega}_l^k + \hat{\theta}_l^k).$$



## Amplitudes:

$$\tilde{A}(n) = \hat{A}^k + \frac{(\hat{A}^{k+1} - \hat{A}^k)}{S} n \quad (\text{linear interpolation})$$

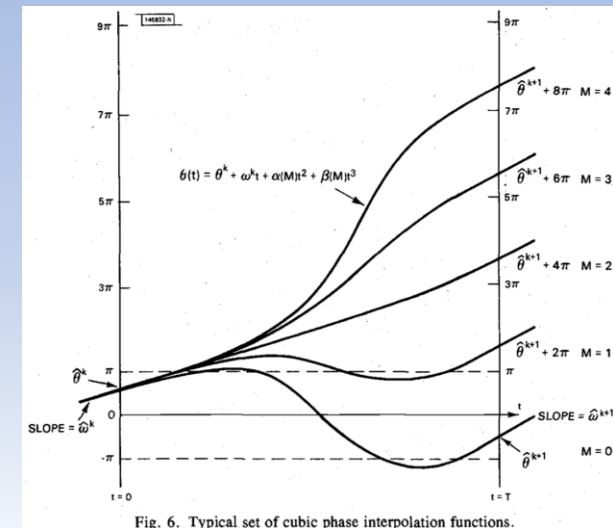
## Frequencies:



(birth-death method)

## Phases:

$$\tilde{\theta}(t) = \zeta + \gamma t + \alpha t^2 + \beta t^3.$$

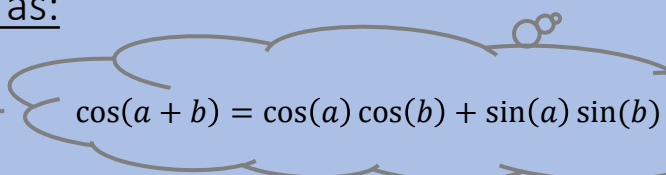


(cubic interpolation)

# Redefining the problem: Phase and Amplitude

Any generic **AM** sinusoidal discrete-time wave  $h[n]$  can be written as:

$$\begin{aligned}h[n] &= A[n] \cos\left(\frac{2\pi f}{f_s}n + \varphi\right) \\&= A[n] \left( \cos(\varphi) \cos\left(\frac{2\pi f}{f_s}n\right) - \sin(\varphi) \sin\left(\frac{2\pi f}{f_s}n\right) \right) \\&= A[n] \cos(\varphi) \cos\left(\frac{2\pi f}{f_s}n\right) - A[n] \sin(\varphi) \sin\left(\frac{2\pi f}{f_s}n\right) \\&= \alpha[n] \cos\left(\frac{2\pi f}{f_s}n\right) + \beta[n] \sin\left(\frac{2\pi f}{f_s}n\right).\end{aligned}$$


$$\cos(a + b) = \cos(a) \cos(b) + \sin(a) \sin(b)$$

Therefore, a speech wave  $s[n]$  can be approximated with AM sinusoidal waves as:

$$s[n] \approx \hat{s}[n] = \sum_{m=1}^M \alpha_m[n] \cos\left(\frac{2\pi f_m}{f_s}n\right) + \beta_m[n] \sin\left(\frac{2\pi f_m}{f_s}n\right).$$

$f_s$  = Sampling rate.

$s$  = Input speech wave.

$\hat{s}$  = Predicted output speech wave.

$M$  = Total number of sinusoid pairs.

$f_m$  = Frequency of the  $m^{th}$  sinusoid pair.

$\alpha_m[n]$  = Amplitudes of the  $m^{th}$  cosine wave.

$\beta_m[n]$  = Amplitudes of the  $m^{th}$  sine wave.

# Redefining the problem: Frequencies

$$s[n] \approx \hat{s}[n] = \sum_{m=1}^M \alpha_m[n] \cos\left(\frac{2\pi f_m}{f_s} n\right) + \beta_m[n] \sin\left(\frac{2\pi f_m}{f_s} n\right).$$

In fact, we can represent **any arbitrary signal** using sums of **AM sinusoids**:

Let  $h[n]$  be an arbitrary discrete-time signal. Choose frequencies  $f_1, f_2$  s.t.  $\text{LCM}(f_1, f_2) < f_s/2$ . Then, we can write  $h[n]$  as:

$$h[n] = h[n] \mathbb{1}\{f_2 n = k f_s/2\} + h[n] \mathbb{1}\{f_2 n \neq k f_s/2\}, \quad k \in \mathbb{N} \Leftrightarrow$$

$$h[n] = \frac{h[n] \mathbb{1}\{f_2 n = k f_s/2\}}{\cos\left(\frac{2\pi f_1}{f_s} n\right)} \cos\left(\frac{2\pi f_1}{f_s} n\right) + \frac{h[n] \mathbb{1}\{f_2 n \neq k f_s/2\}}{\sin\left(\frac{2\pi f_2}{f_s} n\right)} \sin\left(\frac{2\pi f_2}{f_s} n\right) \Leftrightarrow$$

$$h[n] = \alpha_1[n] \cos\left(\frac{2\pi f_1}{f_s} n\right) + \beta_2[n] \sin\left(\frac{2\pi f_2}{f_s} n\right).$$

So we can work with **constant frequencies** over a longer frame.

Of course, we **do not want** such scenario to actually happen, i.e., to basically directly synthesize a speech wave  $s[n]$  using two AM signals. Ideally, **many sinusoids should cooperate** to make this seemingly hard task, an easier one. This just goes to show the “limitless” representation capabilities of this approach from a mathematical/theoretical point of view.

# Redefining the problem: Input

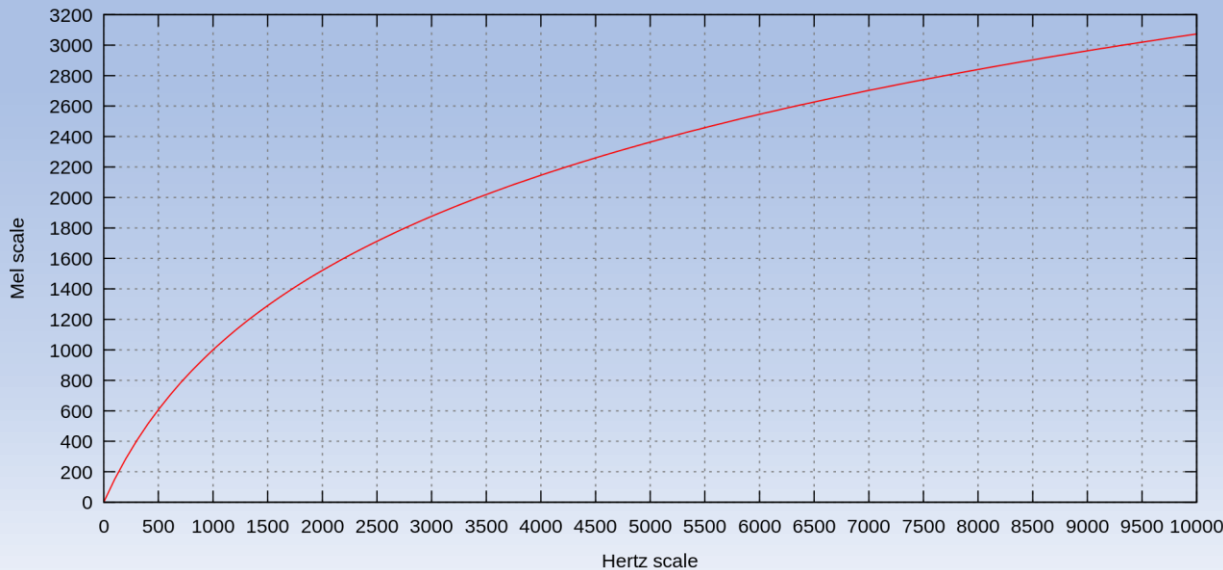
A classic/linear spectrogram gives “equal importance” to all frequencies, e.g., the space between 1-2kHz is half of 2-4kHz. However, we know that human hearing is more sensitive/precise at distinguishing **lower frequencies** than higher ones. For example, a difference between 400-800Hz will be obvious, but between 7600-8000Hz much harder noticeable.

We can take advantage of that, by having as input a **mel-spectrogram**, which **logarithmically** scales the frequencies:

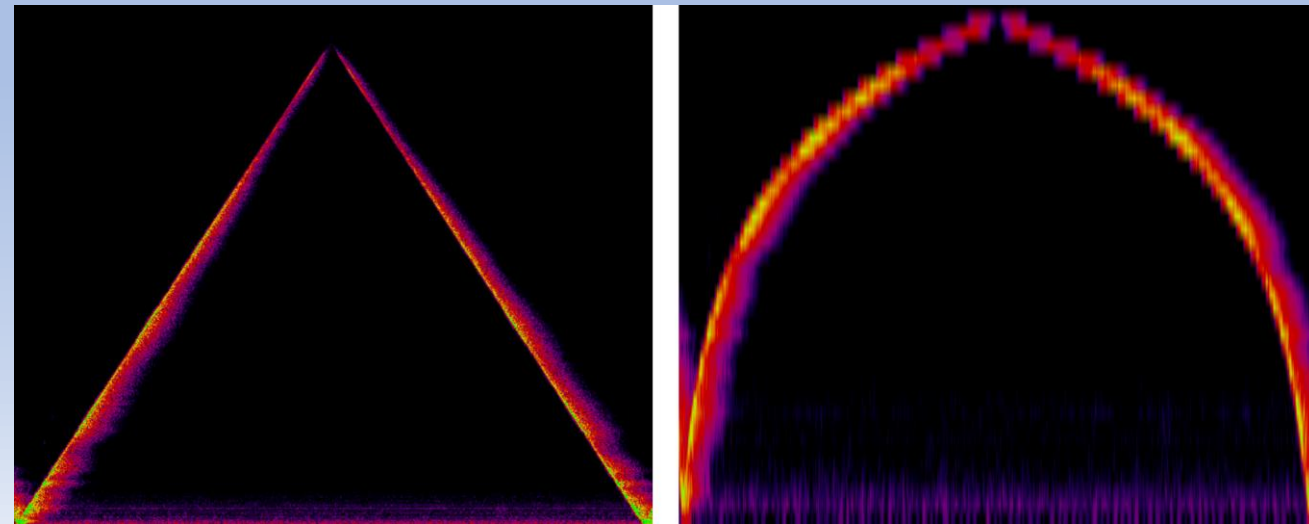
$$m(f) = 2595 \log_{10} \left( 1 + \frac{f}{700} \right) = 1127 \ln \left( 1 + \frac{f}{700} \right). \quad (\text{Common conversion function from hertz to mel scale.})$$

$$f(m) = 700(10^{m/2595} - 1) = 700(e^{m/1127} - 1). \quad (\text{Common conversion function from mel to hertz scale.})$$

Plot of frequencies in Mel versus Hertz scale:



An increasing and decreasing tone from 20Hz to 22kHz and back:

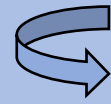


Spectrogram

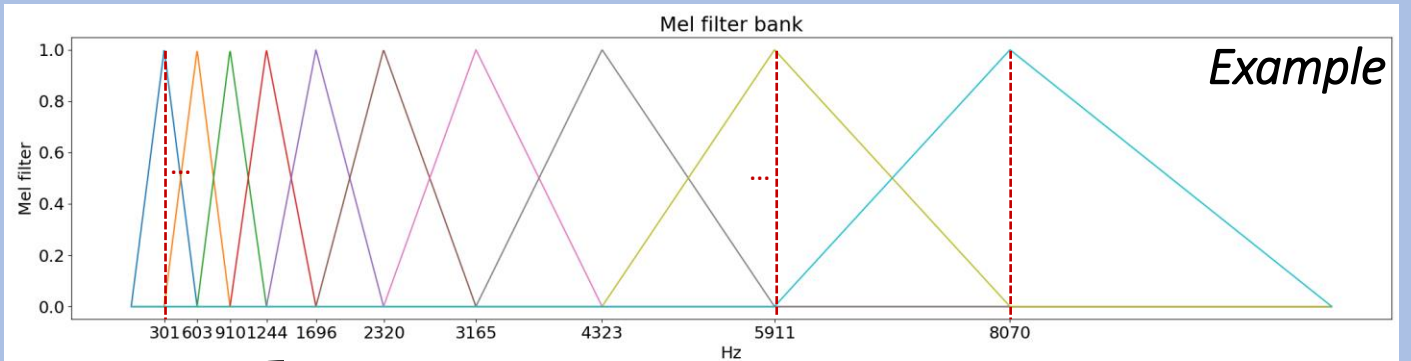
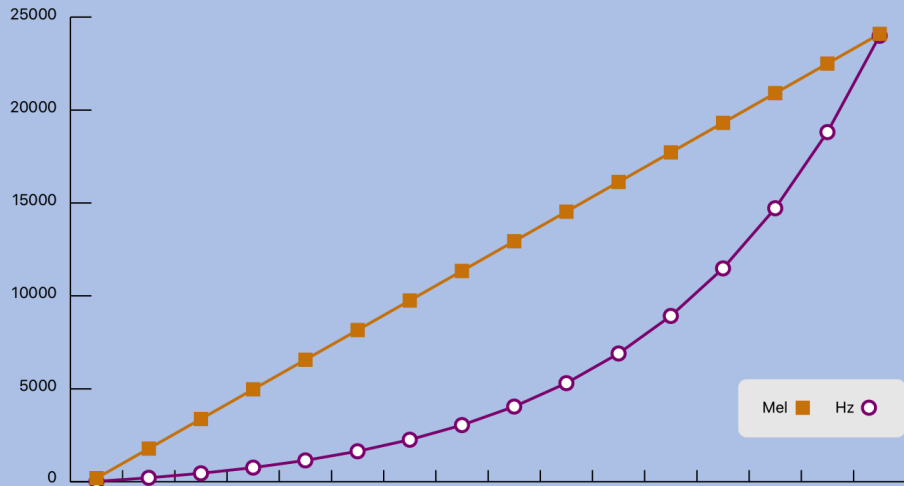
Mel-spectrogram

# Redefining the problem: Input and Frequencies

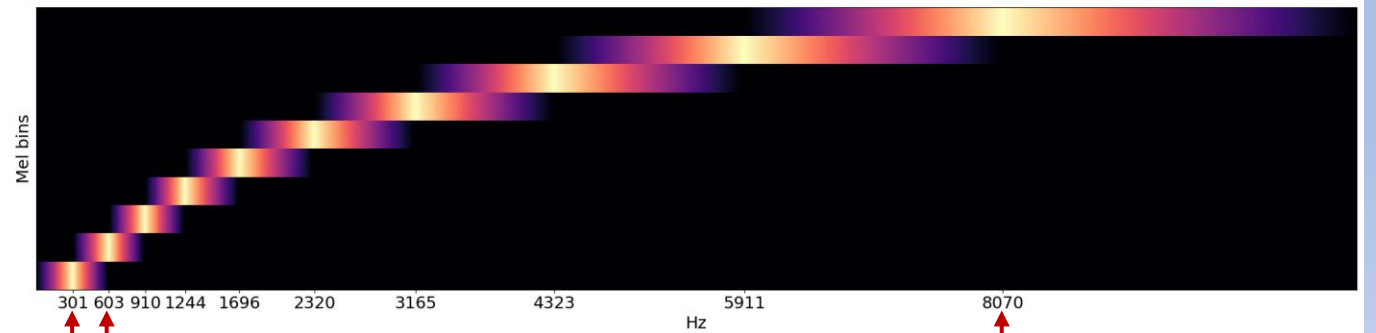
We create  $M$  equally spaced frequencies  $\mathbf{m}_f = (m(f_{min}), \dots, m(f_{max}))$  and then convert them back in Hz scale  $f(m_f)$ .



$M$  Overlapping triangular filters of (exponentially) increasing size:



We can now know the center frequency of each band



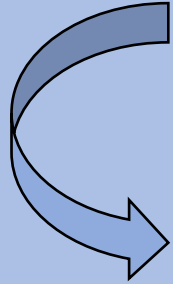
These central band frequencies are the choice for our constant frequencies  $f_m$  in the model !

$$\hat{s}[n] = \sum_{m=1}^M \alpha_m[n] \cos\left(\frac{2\pi f_m}{f_s} n\right) + \beta_m[n] \sin\left(\frac{2\pi f_m}{f_s} n\right).$$



So the number of AM sinusoid pairs in our model will be equal to the number of mel bands  $M$  that we choose for our input spectrogram !

# Redefining the problem: Summary



$$s[n] \approx \hat{s}[n] = \sum_{l=1}^{L(k)} \hat{A}_l^k \cos(n\hat{\omega}_l^k + \hat{\theta}_l^k).$$

(Original Sinusoidal Model)

$$s[n] \approx \hat{s}[n] = \sum_{m=1}^M \alpha_m[n] \cos\left(\frac{2\pi f_m}{f_s} n\right) + \beta_m[n] \sin\left(\frac{2\pi f_m}{f_s} n\right).$$

(Proposed Model)

## Main assumption differences:

- 1) Not constructing small frames with sums of sinusoids:  
→ **longer frames** with sums of AM sinusoids.
- 2) No alternating frequency estimation methods:  
→ **constant frequencies** instead.
- 3) No separate phase or amplitude interpolation:  
→  **$\alpha$ ,  $\beta$  coefficients** compensate for them.
- 4) No linear spectrogram as input:  
→ **mel-spectrogram** instead.
- 5) No analytical parameter estimation from the input:  
→ **???** approach instead.

# Redefining the problem: Solution

$$s[n] \approx \hat{s}[n] = \sum_{m=1}^M \alpha_m[n] \cos\left(\frac{2\pi f_m}{f_s} n\right) + \beta_m[n] \sin\left(\frac{2\pi f_m}{f_s} n\right).$$

Input = Mel-Spectrogram  $S$  generated from a speech signal  $s[n]$ .

Output =  $\alpha_m[n], \beta_m[n]$  coefficients  $\rightarrow$  prediction speech signal  $\hat{s}[n]$ .

Goal = approximate  $s[n]$  with  $\hat{s}[n]$  as closely as possible.

Since we want to emulate real human speech, we can think of it as an **optimization problem**, in which we want to **minimize** the “distance” of our synthetic speech wave  $\hat{s}[n]$  from the corresponding ground truth speech wave  $s[n]$ .

This “distance” is measured by a function of our choosing, i.e., the **loss function**  $\mathcal{L}$ , which gives a numeric estimation of how “close” or “far” its inputs are. E.g., it can be the Mean Squared Error between the two:

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

So we want to create a parametric model  $F$  that takes  $S$  as input, such that its parameters  $\theta$  that calculate the output  $\alpha, \beta$  minimize our defined loss function  $\mathcal{L}$ :

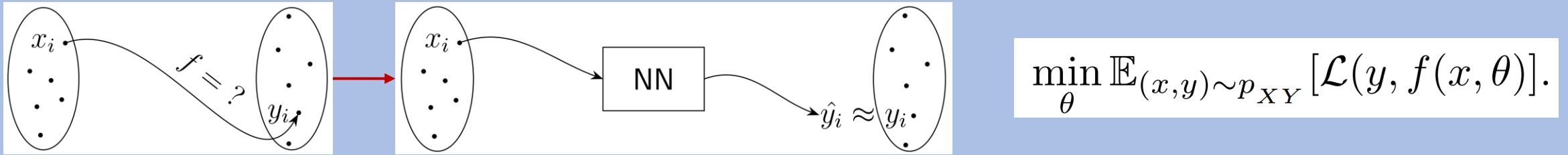
$$\begin{aligned} \min_{\theta} \mathcal{L}(s, \hat{s}) \\ \alpha, \beta = F(S, \theta). \end{aligned}$$



# Solution: Optimization Approach

$$\min_{\theta} \mathcal{L}(s, \hat{s})$$
$$\alpha, \beta = F(S, \theta).$$

There exist many optimization methods depending on the problem's complexity/hardness, e.g., Linear, Convex, Message Passing, Belief Propagation, etc., but the hardness of this problem (vocoder creation) has shown the need for **Neural Networks**.



Our function  $F$  in this case will be a Neural Network.

It will receive  $S$  as its input and has to tune its (large) set of parameters  $\theta$  s.t. its  $\alpha, \beta$  outputs minimize  $\mathcal{L}$ .

Neural Networks are comprised of **layers**, and each layer is made of **weights**, or **neurons** (the trainable parameters).

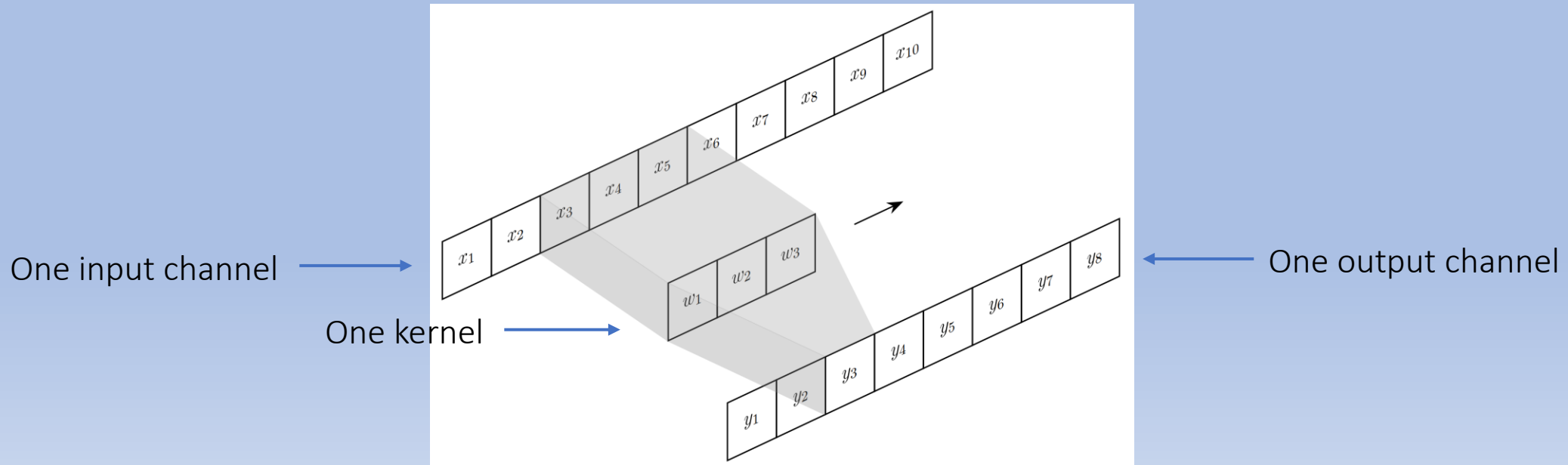
Each type of layer applies a different operation on its input to give its output.

The main layer of interest for us will be the **Convolutional Layer**.

# Neural Networks: 1D Convolutional Layer

The equation describing convolution in 1D discrete space between an input sequence  $x$  and an output sequence  $y$  using one filter, or kernel  $w$  is:

$$y_j = \sum_{m=1}^M \left[ w_m \cdot x_{j'} \right] + b, \quad j' = j - 1 + m.$$



One can think of convolving as sliding a window over the input.

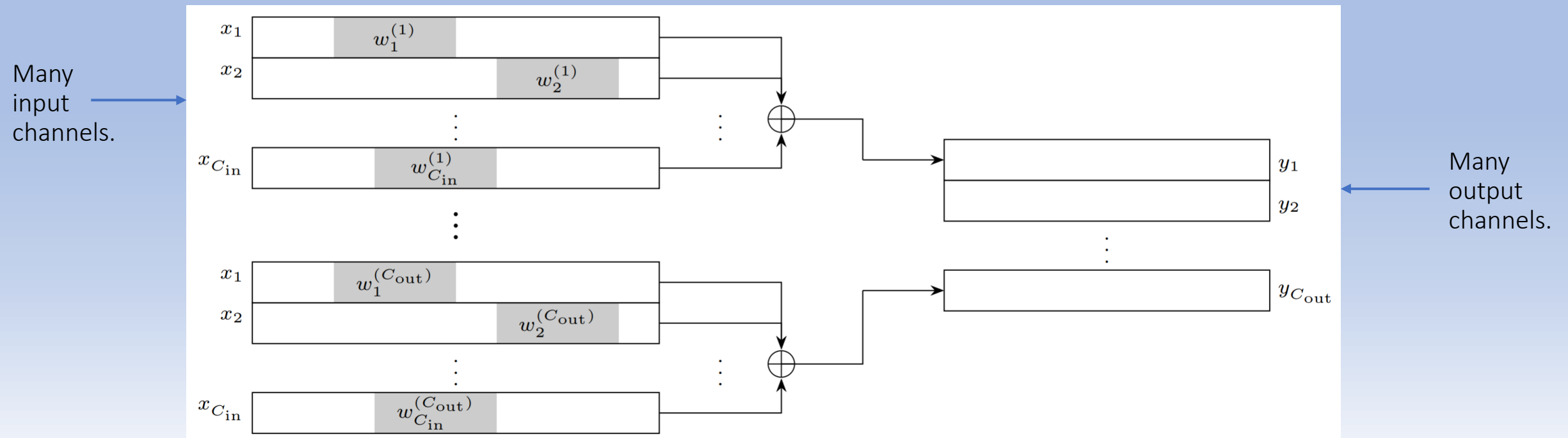
Here, the filter values  $w_i$  and the bias term  $b$  are the trainable parameters.

# Neural Networks: 1D Convolutional Layer

How to process a 2D array, e.g., a spectrogram, with a 1D convolutional layer?  
We can treat each input row (frequency band) as an input channel.

How do we also get a 2D output from a 1D convolutional layer?  
We can have multiple filters operate over one input, thus resulting in a **different output** sequence **per set of filters**, i.e., output channel:

$$y_{i,j} = \sum_{i'=1}^{C_{in}} \sum_{m=1}^M \left[ w_{i',m}^{(i)} \cdot x_{i',j'} \right] + b_i, \quad j' = S \cdot (j - 1) + m.$$



# Neural Networks: Transposed 1D Convolutional Layer

How can we upsample along the time axis within an neural network in a “smart way” ?

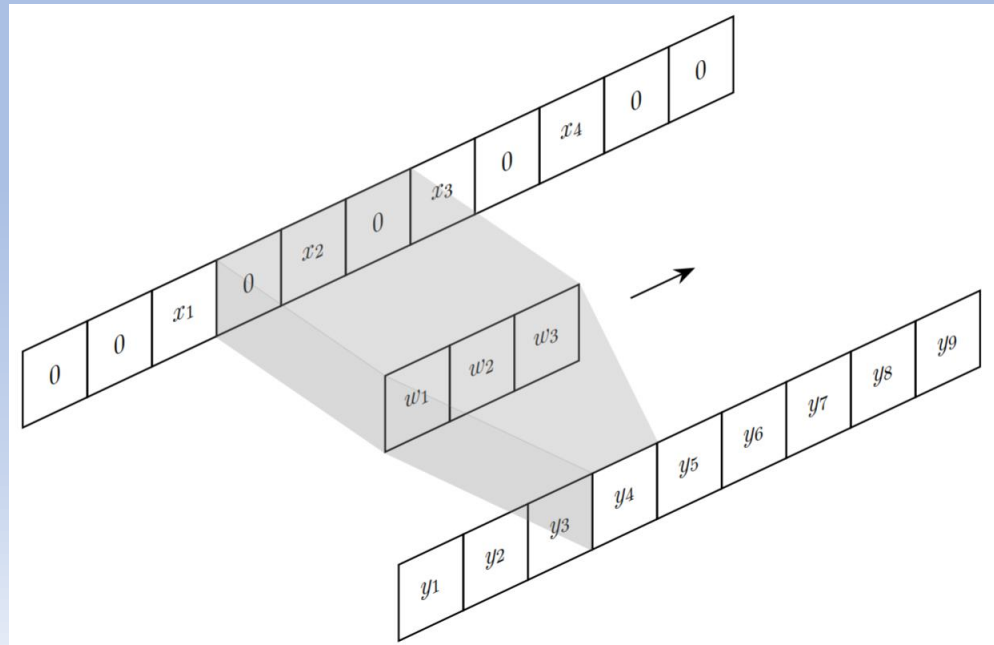


The transposed convolutional layer is essentially a trainable upsampling layer.

It firstly pads between the input's consecutive values:

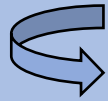
$$x'_{i,j} = \begin{cases} x_{i,j'}, & \text{if } j = S \cdot (j' - 1) + 1 \\ 0, & \text{otherwise.} \end{cases}$$

And then acts as a normal convolutional layer:



# Neural Networks: Optimizer

But how are the trainable parameters, e.g., convolution filters being updated ?



The **optimizer** of the Neural Network is the algorithm responsible for updating these parameters:

---

## Algorithm 1 Stochastic Gradient Descent

---

**Require:** Number of Iterations:  $T \in \mathbb{N}$ , Parametric Model:  $f$ , with Trainable Weights:  $w$  and Biases:  $b$ , Input Training Dataset:  $x$ , Corresponding Ground Truth:  $y$ , Loss Function:  $\mathcal{L}$ , Learning Rate:  $\eta \in \mathbb{R}^+$ , Weight Initialization:  $w_0$ , Bias Initialization:  $b_0$

```
1:  $w \leftarrow w_0$            # Initialize the weights  $w$  with the given weight initialization  $w_0$ .
2:  $b \leftarrow b_0$          # Initialize the biases  $b$  with the given bias initialization  $b_0$ .
3: for  $i \in [1, \dots, T]$  do # For all number of iterations  $T$  given:
4:    $x_i \leftarrow \text{Random}(x)$  # Fetch a sample  $x_i$  from the input training data  $x$  uniformly at random.
5:    $\hat{y}_i \leftarrow f(x_i)$    # Get the model's prediction  $\hat{y}_i$  for the input  $x_i$ .
6:    $L_i \leftarrow \mathcal{L}(\hat{y}_i, y_i)$  # Compute the loss between the prediction  $\hat{y}_i$ , and the ground truth  $y_i$ .
7:    $\Delta w \leftarrow -\nabla_w L_i$  # Compute the negative gradient  $-\nabla$  of the loss  $L_i$  w.r.t. the weights  $w$ .
8:    $\Delta b \leftarrow -\nabla_b L_i$  # Compute the negative gradient  $-\nabla$  of the loss  $L_i$  w.r.t. the biases  $b$ .
9:    $w \leftarrow w + \eta \cdot \Delta w$  # Scale  $\Delta w$  by the learning rate  $\eta$ , and update the weights  $w$ .
10:   $b \leftarrow b + \eta \cdot \Delta b$  # Scale  $\Delta b$  by the learning rate  $\eta$ , and update the biases  $b$ .
11: end for
```

---

This is not the actual optimizer used; there are many improved versions of SGD, e.g., Adam, but we do not have the time to dive deeper into optimizers.

The optimizer is based on the **gradient** for updating the weights, but how is this gradient computed?

# Neural Networks: Backpropagation

This gradient is computed efficiently by the **backpropagation** algorithm.

Firstly, a graph representing the entire corresponding output function of a neural network is formed, called **computational graph**.

With an automated procedure called **backward pass** the gradient of the loss function is essentially computed using the **chain rule**:

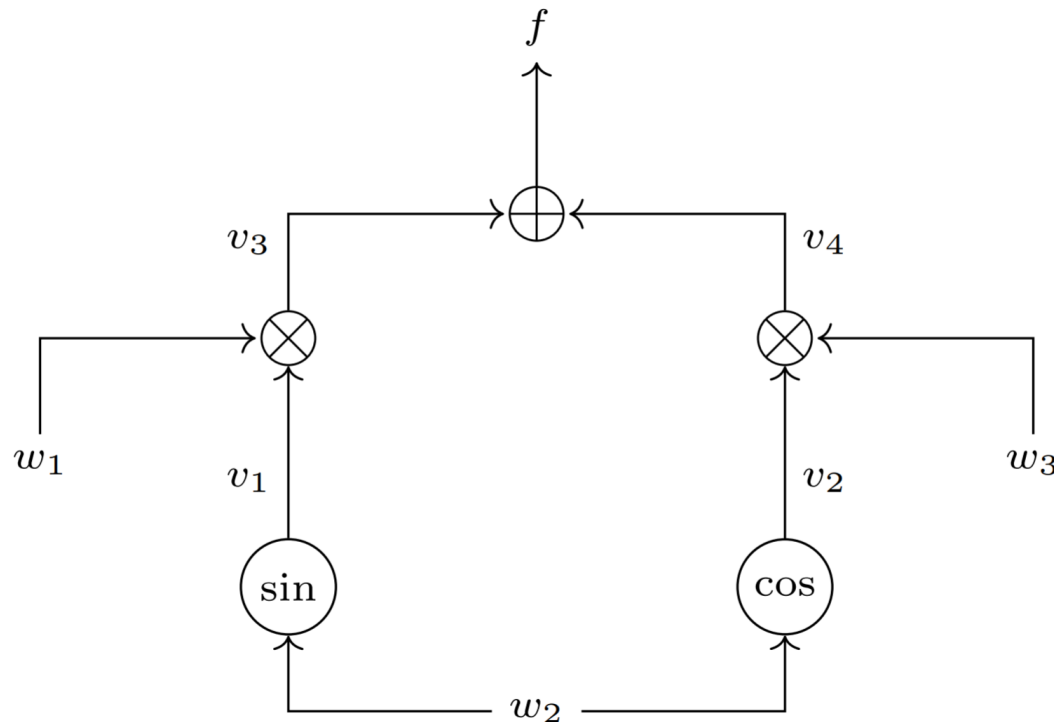
$$\frac{\partial}{\partial x}(f \circ g) = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x}$$

Toy Example:

$$f(w_1, w_2, w_3) = w_1 \sin(w_2) + w_3 \cos(w_2).$$

**Forward Pass**

$$\begin{aligned}v_1 &= \sin(w_2) \\v_2 &= \cos(w_2) \\v_3 &= v_1 \cdot w_1 \\v_4 &= v_2 \cdot w_3 \\f &= v_3 + v_4\end{aligned}$$



**Backward Pass**

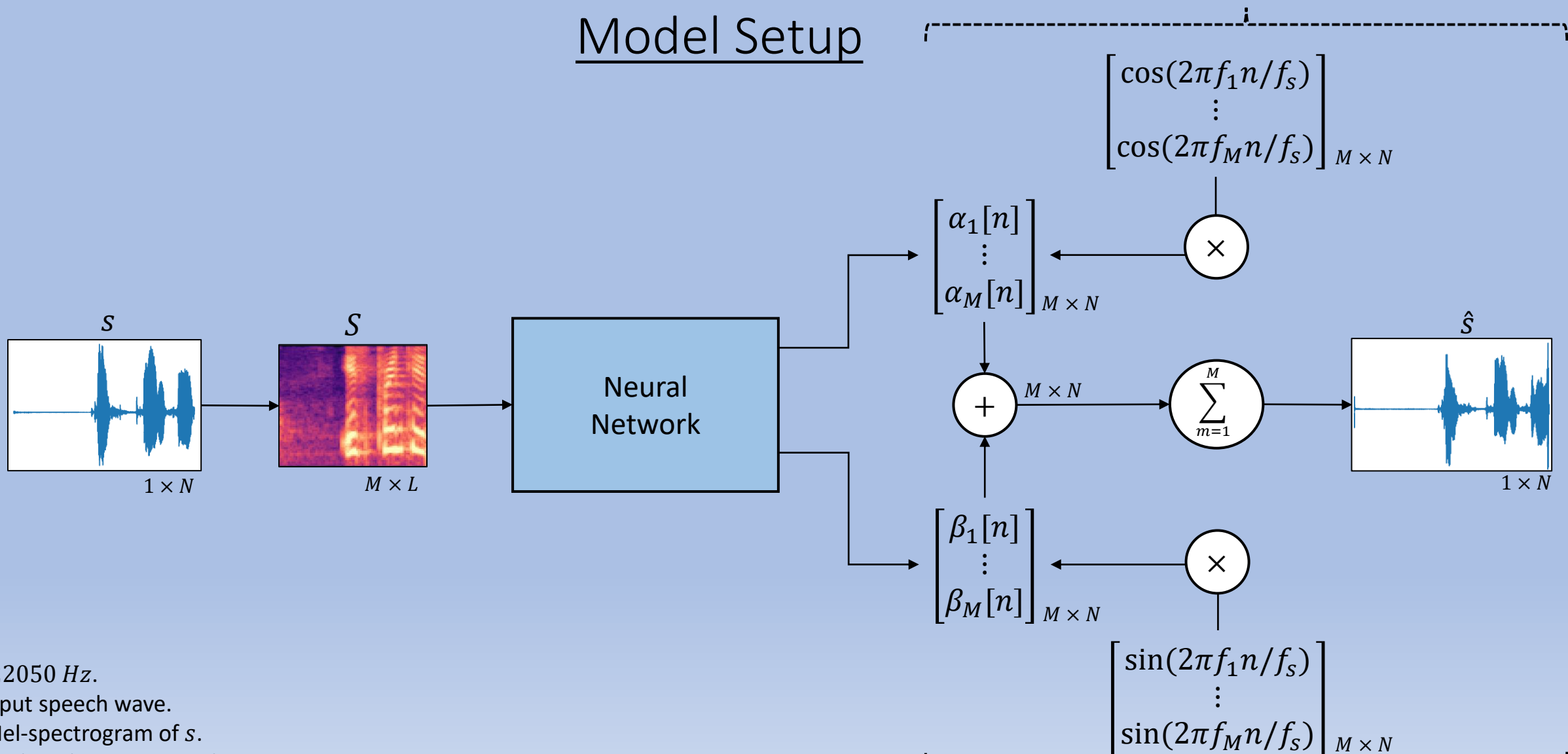
$$\bar{v}_3 = \frac{\partial f}{\partial v_3} = 1 \quad \bar{v}_1 = \frac{\partial f}{\partial v_3} \frac{\partial v_3}{\partial v_1} = \bar{v}_3 \cdot w_1$$

$$\bar{v}_4 = \frac{\partial f}{\partial v_4} = 1 \quad \bar{v}_2 = \frac{\partial f}{\partial v_4} \frac{\partial v_4}{\partial v_2} = \bar{v}_4 \cdot w_3$$

$$\bar{w}_1 = \frac{\partial f}{\partial v_3} \frac{\partial v_3}{\partial w_1} = \bar{v}_3 \cdot v_1 \quad \bar{w}_3 = \frac{\partial f}{\partial v_4} \frac{\partial v_4}{\partial w_3} = \bar{v}_4 \cdot v_2$$

$$\bar{w}_2 = \frac{\partial f}{\partial v_1} \frac{\partial v_1}{\partial w_2} + \frac{\partial f}{\partial v_2} \frac{\partial v_2}{\partial w_2} = \bar{v}_1 \cdot \cos(w_2) - \bar{v}_2 \cdot \sin(w_2)$$

# Model Setup

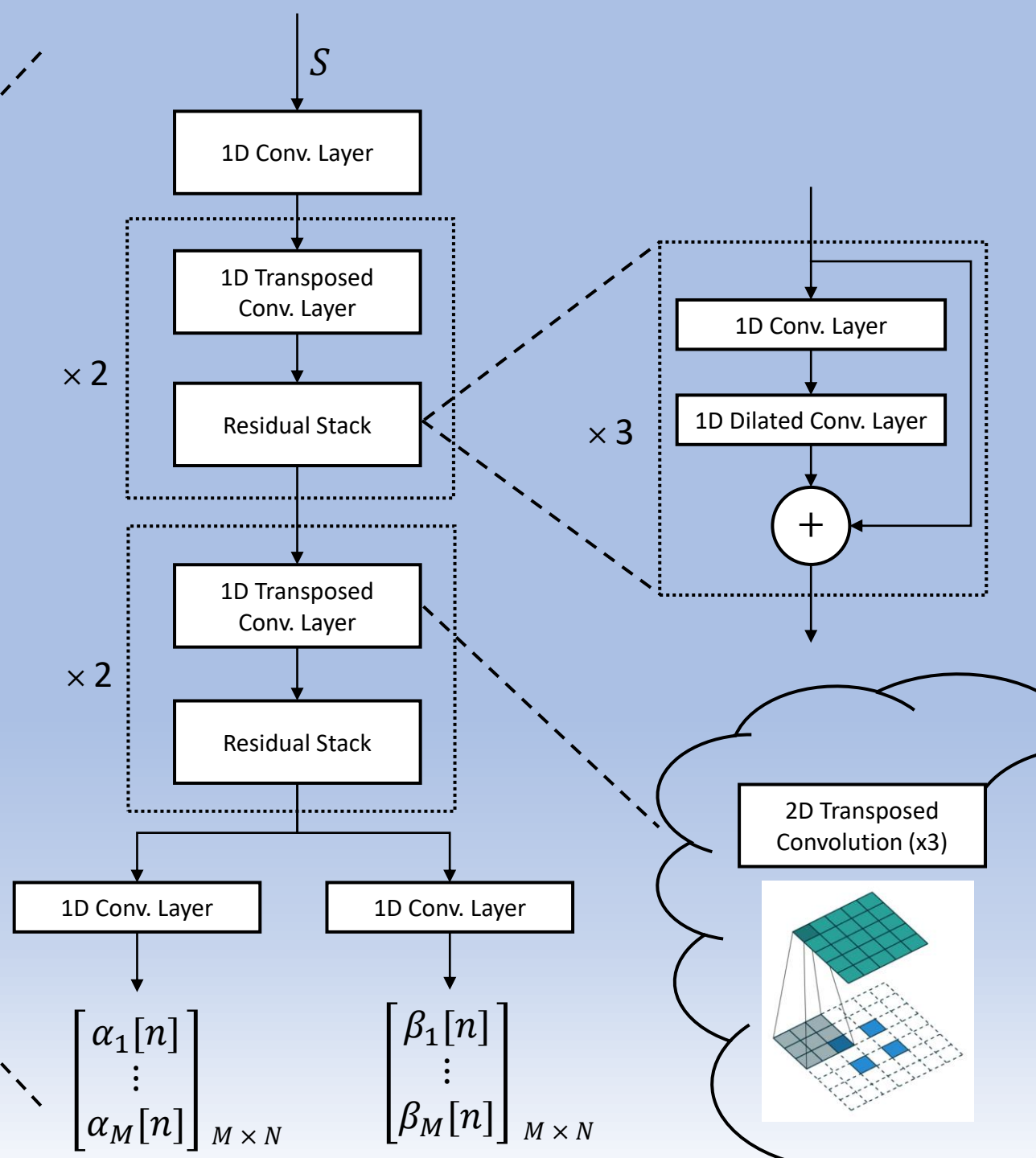
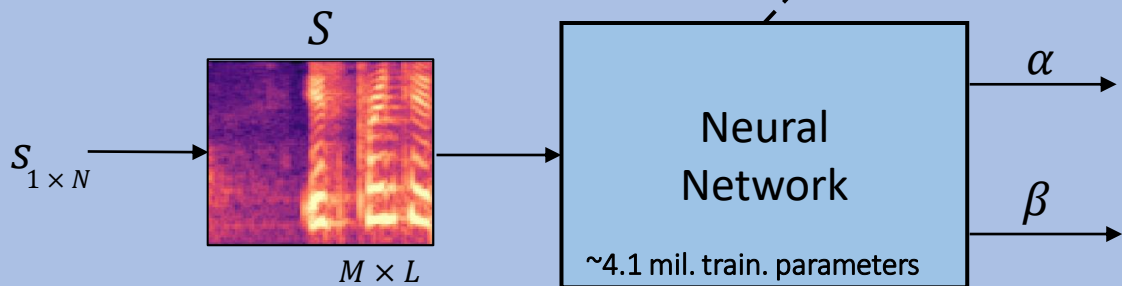


## Synthesis:

$$\hat{s}[n] = \sum_{m=1}^M \alpha_m[n] \cos\left(\frac{2\pi f_m}{f_s} n\right) + \beta_m[n] \sin\left(\frac{2\pi f_m}{f_s} n\right).$$

- $f_s = 22050 \text{ Hz}$ .
- $s$  = Input speech wave.
- $S$  = Mel-spectrogram of  $s$ .
- $\hat{s}$  = Predicted output speech wave.
- $N$  = Total number of samples.
- $M$  = Total number of mel bins or sinusoids pairs.
- $f_m$  = Central frequency of the  $m^{\text{th}}$  mel band.
- $\alpha_m[n]$  = Amplitudes of the  $m^{\text{th}}$  AM cosine wave.
- $\beta_m[n]$  = Amplitudes of the  $m^{\text{th}}$  AM sine wave.

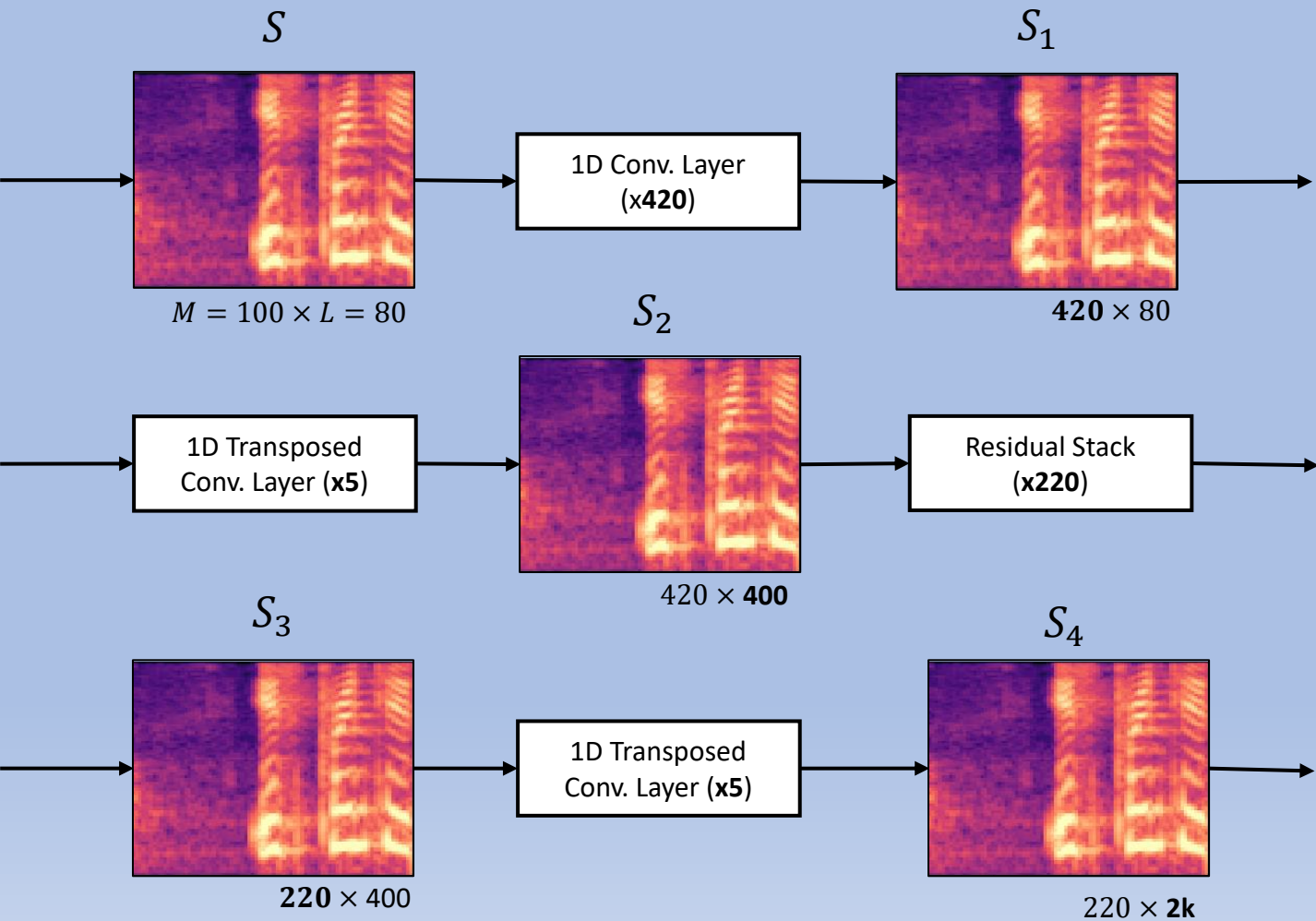
# Neural Network Architecture



*MeGAN: Generative Adversarial Networks for Conditional Waveform Synthesis, Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, Aaron Courville.*

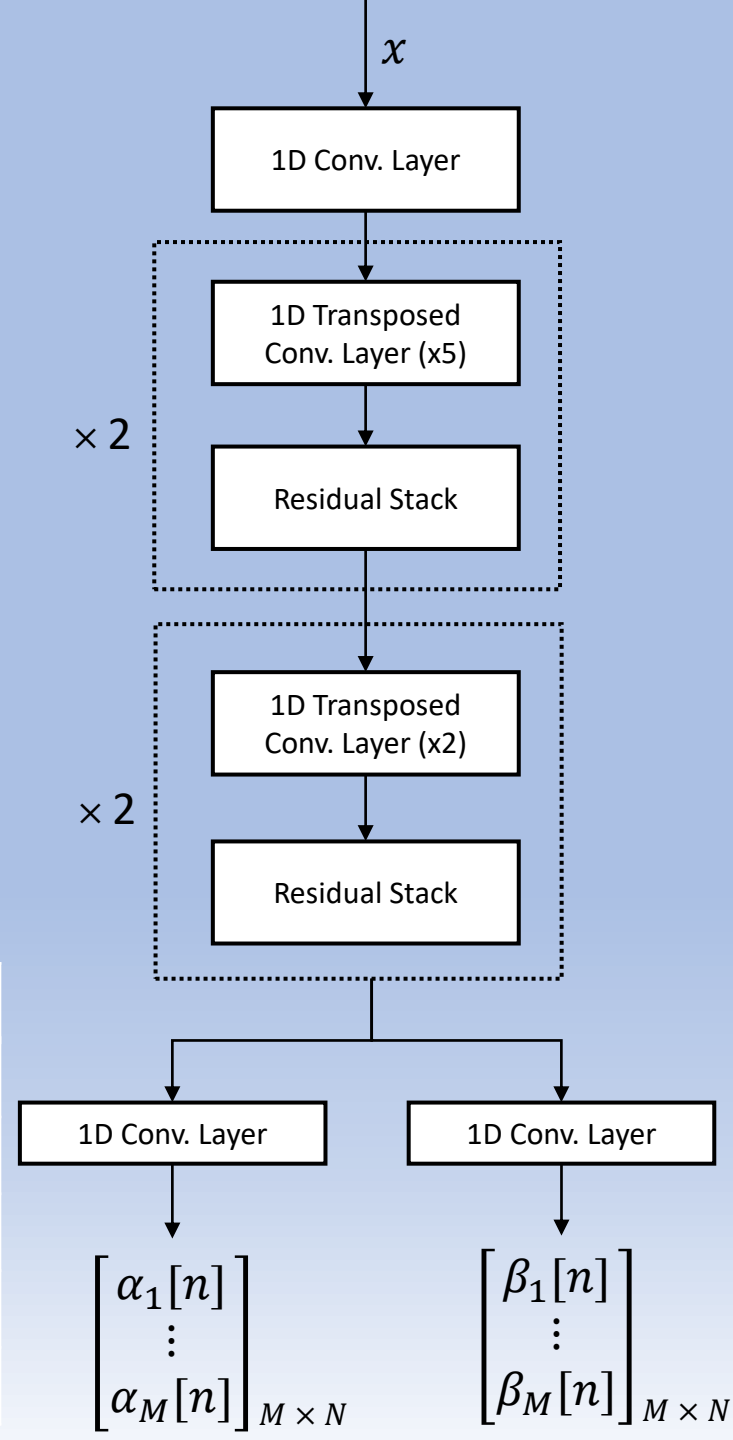


# Upsampling Example



$$\begin{aligned} & \rightarrow \begin{bmatrix} \alpha_1[n] \\ \vdots \\ \alpha_M[n] \end{bmatrix}_{M = 100 \times N = 8192} \\ & \rightarrow \dots \\ & \rightarrow \begin{bmatrix} \beta_1[n] \\ \vdots \\ \beta_M[n] \end{bmatrix}_{M = 100 \times N = 8192} \end{aligned}$$

filters
420
220
160
140
$M = 100$



# Loss Function

Our loss function focuses on the similarity, (or the differences)  $s$  and  $\hat{s}$  have on the spectral domain. Meaning that our assumption is that the better our prediction  $\hat{s}$  gets, the closer its spectrogram will also get to the ground truth  $s$  (so the loss will have a smaller value). Hence, the network will try to minimize this difference between the two signals.

Spectral Convergence:

$$L_{sc}(s, \hat{s}) = \frac{\| |STFT(s)| - |STFT(\hat{s})| \|_F}{\| |STFT(s)| \|_F}$$

Logarithmic Magnitude:

$$L_{mag}(s, \hat{s}) = \frac{1}{N} \| \log|STFT(s) + \epsilon| - \log|STFT(\hat{s}) + \epsilon| \|_{L_1}$$

Summing over different parameters:

$$L_S(s, \hat{s}) = \sum_{i=1}^3 L_{sc}^{(i)}(s, \hat{s}) + L_{mag}^{(i)}(s, \hat{s})$$

$N$	win_len	hop_len
1024	1024	256
512	240	50
2048	1200	240

Final Loss Function:

$$\mathcal{L}_S(s, \hat{s}) = L_S(s, \hat{s}) + L_S(s', \hat{s}')$$

Objective:

$$\min_{\hat{s}} (\mathcal{L}_S(s, \hat{s}))$$

# Model Architecture

$$f_s = 22050 \text{ Hz}$$

$s$  = Input speech wave.

$S$  = Mel-spectrogram of  $s$ .

$\hat{s}$  = Predicted output speech wave.

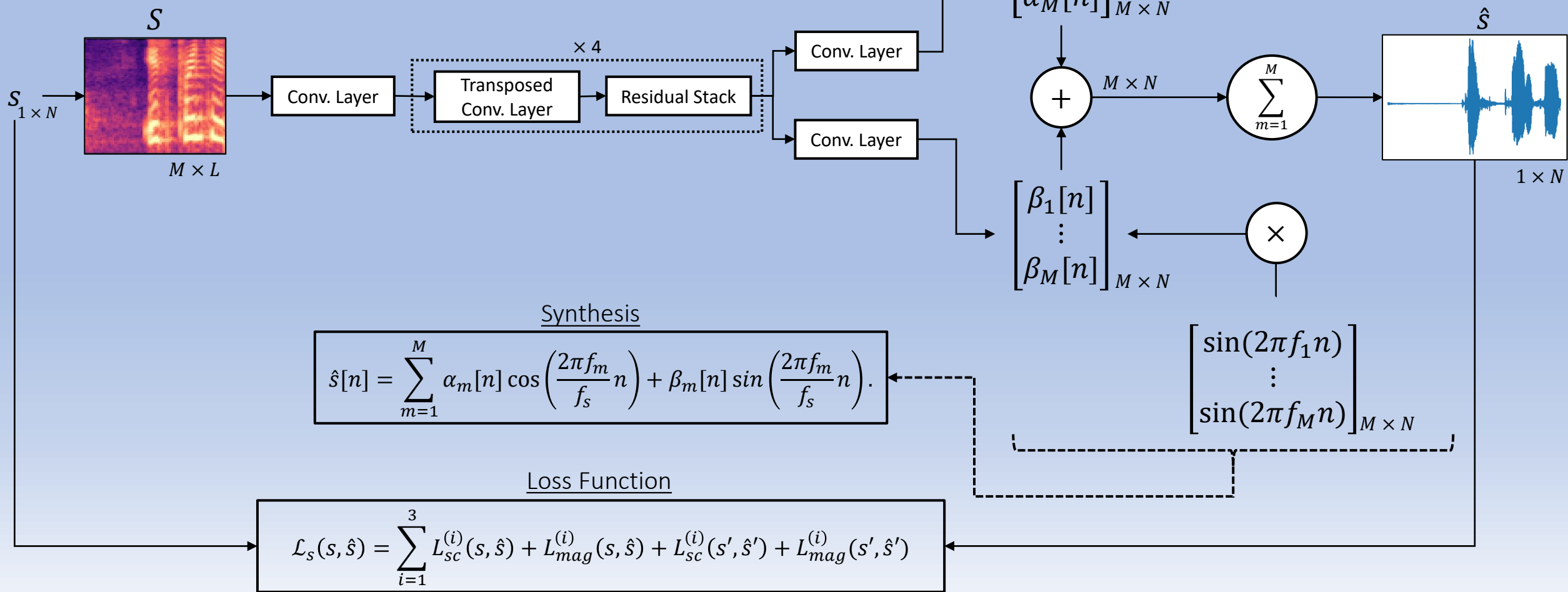
$N$  = Total number of samples = 8192.

$M$  = Total number of mel bins or sinusoid pairs = 100.











$f_m$  = Central frequency of the  $m^{\text{th}}$  mel band.

$\alpha_m[n]$  = Amplitudes of the  $m^{\text{th}}$  AM cosine wave.

$\beta_m[n]$  = Amplitudes of the  $m^{\text{th}}$  AM sine wave.



# Results

<i>Ground Truth</i>	<i>Inference</i>	<i>Text</i>
		"...thorough description of the responsibilities of the advance agent..."
		"...by mid June nineteen sixty four..."
		"...it will support any reasonable request for funds..."
		"...it has received assistance also..."
		"...from data processing experts at the CIA..."

Thank You !

