DEEP LEARNING PROJECT REPORT

****

**Tittle:-“Text-To-Speech Synthesis(Hindi)”**

**SUBMITTED BY**

B. Ieeshasree 19BCS021

Chaithanya B 19BCS028

M. Sai Chaitra 19BCS065

Nikhita B K 19BCS077

Sabiha H B 19BCS094

**UNDER THE GUIDANCE OF**

Dr. K.T. Deepak

Assistant Professor, Electronics & Communication Engineering

**Acknowledgment**

We would like to express our sincere gratitude to Dr. K T Deepak, Asst. Professor, Department of ECE-IIIT Dharwad for his guidance and constant support throughout the course of this minor project. We would also like to thank all the faculty and administration of the institute who ensured the needs were fulfilled for the completion of this project.

Date : December,2022

Place: Dharwad

B. Ieeshasree(19BCS021)

Chaithanya B(19BCS028)

M. Sai Chaitra(19BCS065)

Nikhita B K(19BCS077)

Sabiha H B(19BCS094)

**Table of Contents:**

**1.Introduction**

1.1 Traditional methods

1.2 Statistical parametric speech Synthesis:

**2.Encoder-Decoder**

2.1 Encoder

2.2 Attention Mechanism

2.3 Tacotron 2 Attention

2.4 Decoder

2.5 Post-Net

**3.Vocoder**

3.1 Introduction to Vocoder

3.2 Methods of building vocoder

3.3 Waveglow

3.4 Brief introduction of GAN architecture

3.5 Short-time Fourier transform

3.6 The training process of waveglow

**4.Results**

**5.Summary**

**6.References**

**1.Introduction**

**Speech synthesis** is the artificial production of human speech. A computer system used for this purpose is called a **speech computer** or **speech synthesiser.** A text-to-speech (TTS) system is a system which converts input text into artificial speech.

**In TTS, we have mainly two important modules:**

**Text analysis:** Whichis required because generally texts won’t be standard format (i.e texts may be in the format of abbreviation or symbols) we need to organise that and convert into appropriate standard format.

**Waveform generations**: Now from the text we can generate waveforms using waveform generator.

**The ideal requirements of speech synthesiser:**

* Intelligibility: (speaks about whether the message content conveyed/ understandable or not)
* Naturalness: (similarity of synthesised speech with the human speech)
* Expressive speech (speaks about how well we can modulate the synthesised speech)

**Applications**

● e-book and email reader.

● Alternate Communication for people with disabilities.

● Spoken dialogue system, Voice bot.

● Speech-to-speech Translation.

● Communication robots.

* 1. **Traditional Methods:**
* **Formant Speech Synthesis:**



**Fig:** Formant Speech Synthesis physical model.

Formant synthesis does not use human speech samples at runtime. Instead our vocal tract have many Format frequencies (a formant is usually defined as a broad peak, or local maximum, in the spectrum) (f1, f2, f3..) instead of human speech samples, here we analyse format frequencies for different sound units, after analysing can we try to get synthesised speech from physical model. Parameters such as fundamental frequency, voicing, and noise levels are varied over time to create a waveform of artificial speech. This method is sometimes called **rules-based synthesis.** Formant-synthesised speech can be reliably intelligible. However, natural speech is not achieved here (i.e quality of speech feels robotic).

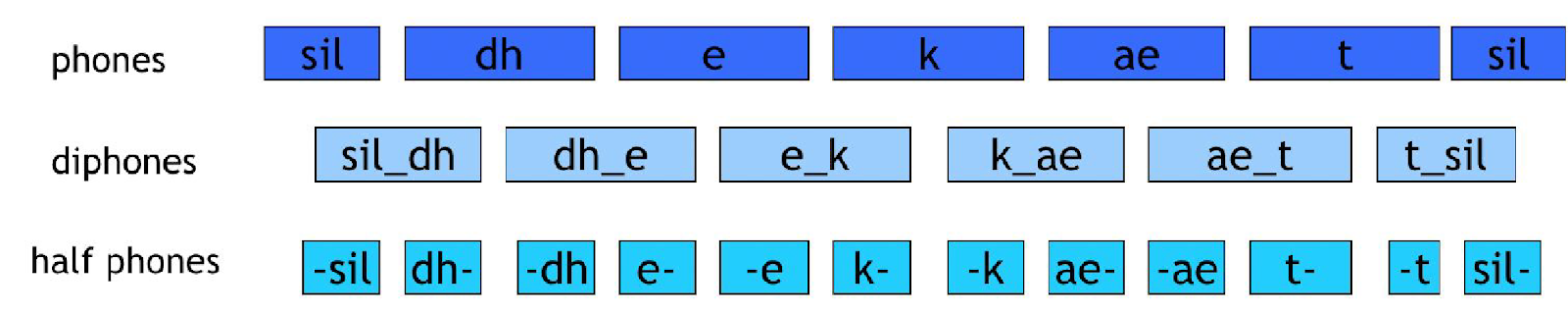
Here we build the model for each sound/ phoneme and try to combine them to form a single waveform for a sequence of words as an input.

* **Concatenative Based Speech Synthesis:**

They have a waveform for each unit (example: for ‘bus’ (we have different components ba, aa, sa )), we concatenate them, since we keep the original waveform, this naturalness of speech is also taken into account. Concatenative units are taken from the database of original speech recordings.

Two types of concatenative TTS depend on the type database used:

* **Diphone synthesiser:**
* Sound units are diphones (In phonetics, a **diphone** is an adjacent pair of phones in an utterance. For example, in [daɪfəʊn], the diphones are [da], [aɪ], [ɪf], [fə], [əʊ], [ʊn]. It is usually used to refer to a recording of the transition between two phones).
* Single example diphone inventory/database.
* Units having fixed length.
* **Unit selection speech synthesis system:**
* Here units can be diphone, half phone and even word.
* Large database of units with multiple examples.
* Optimal units are selected by searching through the database.



**Fig:** Types of units in the unit selection speech synthesis system.

Diphones provide less spectral distortion at the concatenation points. To get phonetic coverage a large database is required.

**Selection of units:**

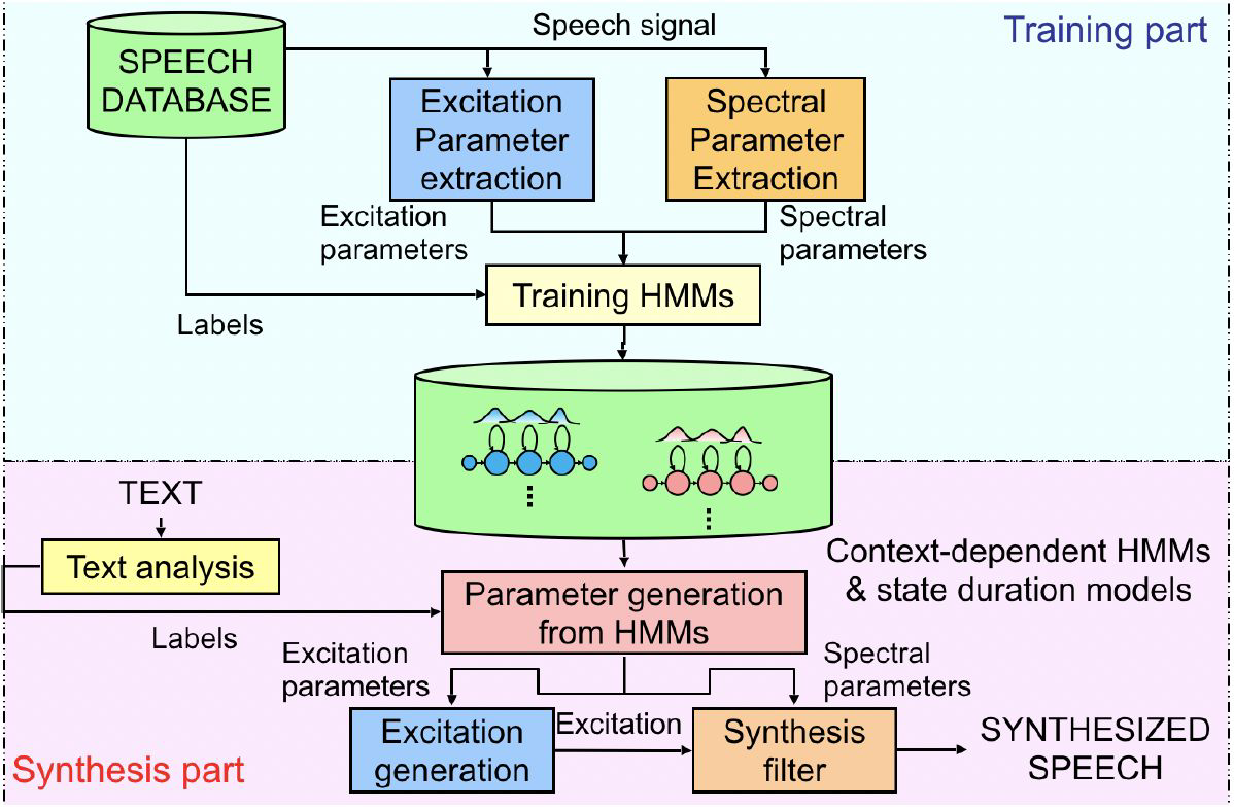
**Target cost:**

For the target cost how much is the candidates units are near to features (such as f0(fundamental frequency), duration and energy)

**Join cost:**

For join cost (joining 2 units): smooth spectral match at the concatenation points.

* 1. **Statistical Parametric Speech Synthesis (HMM based):**



**Fig:** HMM based Speech Synthesis

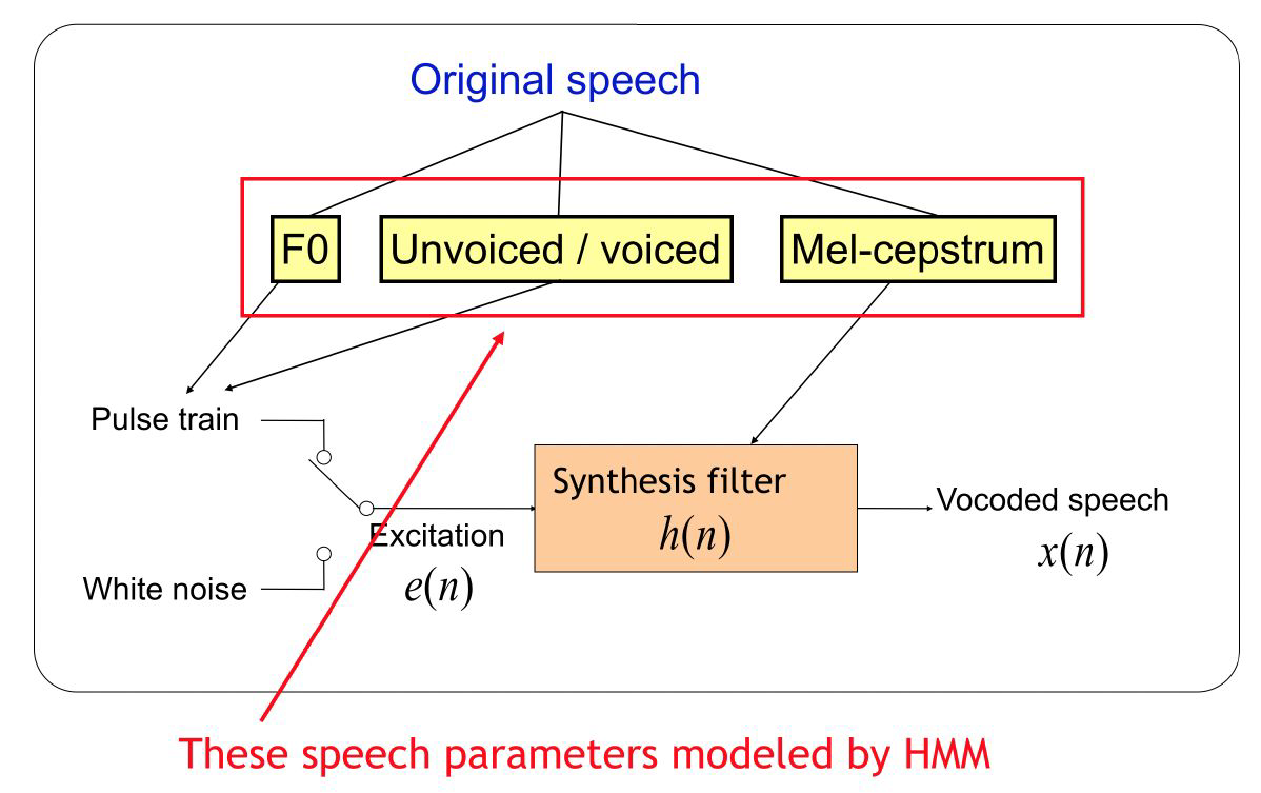
**Excitation parameter**: It is like f0(fundamental frequency gives intonation for the audio) or phonemes– this to helpful for generating voiced decision (periodic) or unvoiced decisions (non-periodic).

**Spectral parameter**: It represents vocal tract shape, and different format frequency can be represented from the mfcc coefficient.

First we extract both the parameters (**Excitation parameter and Spectral parameter**) from the speech database in the training process in the hmm model.

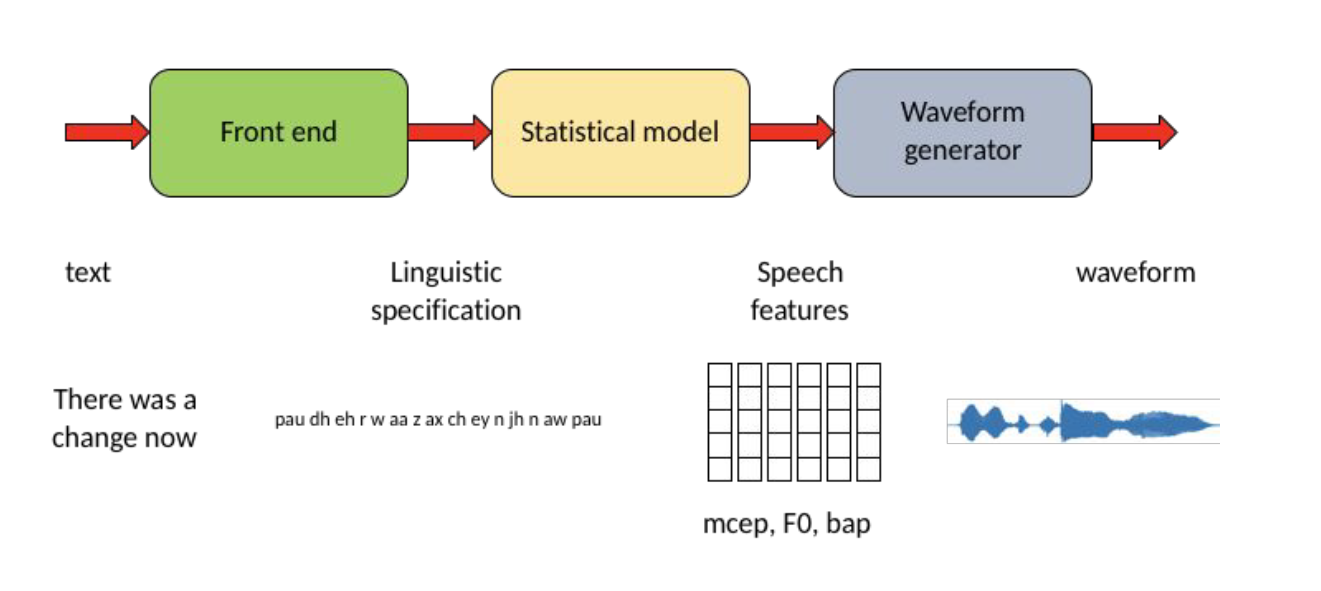
While synthesising we generate parameters(**Excitation parameter and Spectral parameter**).

* As a result, we get 2 signals in output which are excitation signal and signal from synthesis filter, convolving both we get synthesised speech.

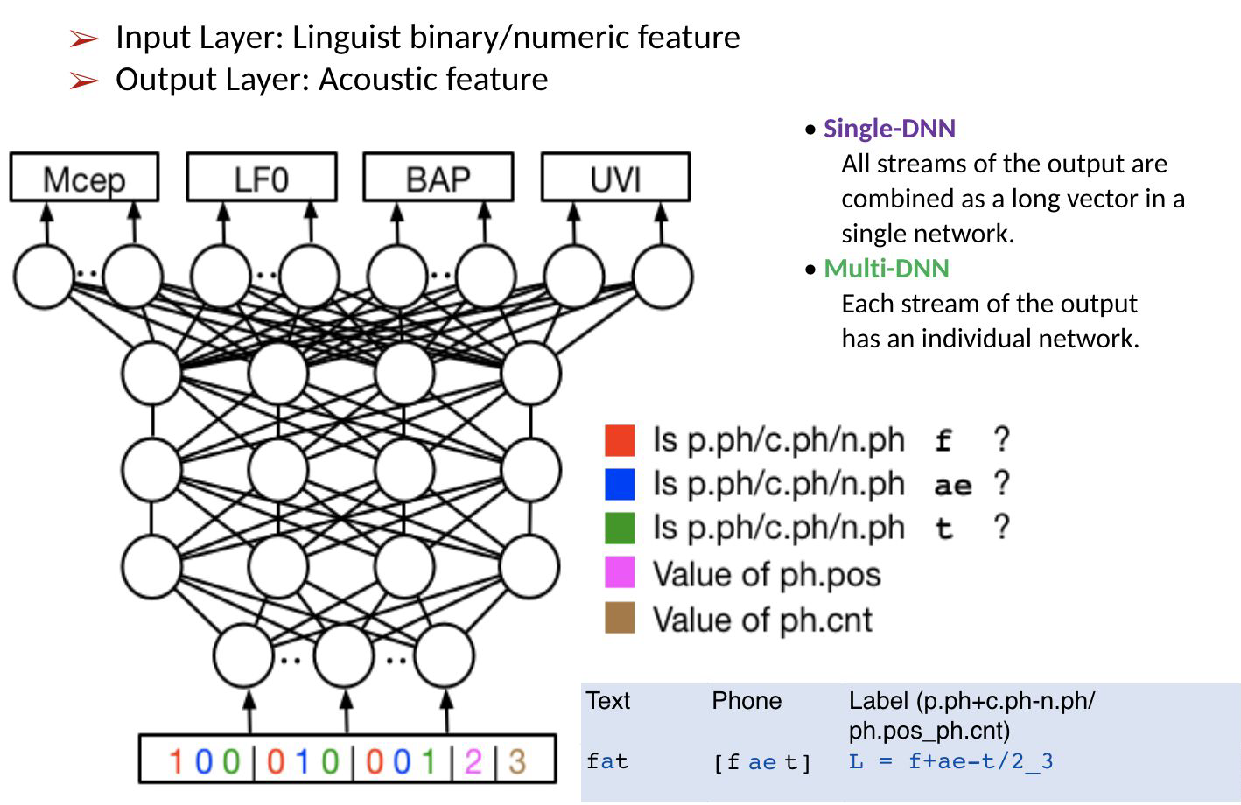


**Fig**: Synthesis filter (h(n)) block diagram.

**DNN based speech synthesis:**

 **Fig:** DNN based speech synthesis block diagram.

Here we convert the sequence of text into linguistic specification (such as phonemes), later we extract speech features (such as mel-ceps, f0(fundamental frequency), bap), and give these features to waveform generators to get waveform.



**Fig:** DNN based speech synthesis model.

Example: “fat” word is first converted to linguistic specifications where we convert them to labels where phonemes are represented in the form of:

**POS (part of speech**)- mapping speech and the word

**LTS(letter to sound rule)-**..how phoneme is there for each letter(mapping)

And then further converted into binary which maps to speech features(mel-ceps, f0(fundamental frequency)), then after generating into waveform.

**Database for Hindi TTS:**

**Encoder-Decoder**

Encoder-decoder architecture is usually the way for Neural Networks in a sequence-to-sequence scenario, where the inputs and outputs are both sequences.

An encoder reads an input sequence (or sentence) and compresses it to a fixed-length "thought vector" in the original encoder-decoder system. Using the hidden thought vector, the decoder is trained to predict the output sequence one at a time. RNNs are used as both the encoder and decoder in the most typical method.

Bahdanau et al. devised the Attention technique in response to the encoder's limited ability to hold compressed information in a fixed-length context vector. Attention is typically used to relieve the encoder of summarising the entire input sequence in a fixed-length context vector. Instead, the encoder converts the input sequence into equal-length annotations. The decoder learns to "Attend" to different parts of these annotations while creating the output sequence. We will go over this primary notion in more detail later, particularly in the section on Attention.

**Encoder**

The encoder in Tacotron-2 converts an input sentence X = X 1, X 2,..., X T x into a set of annotations known as encoder hidden states H = H 1, H 2,..., H T x. It is worth noting that both sequences are the same length. i.e., the encoder generates a series of annotations, each of which provides information on the equivalent X I word in input sequence X about its neighbors

Each input token X I in T2 corresponds to a single character. A bi-directional LSTM layer follows three Convolutional Layers in the T2 encoder. Convolutional layers, like image processing, are excellent for representing local correlations between inputs (in our instance, local correlations between characters) by producing feature maps useful for RNN modeling of sequences. T2 is given a long-term context via the usage of these convolutional layers, similar to N-grams. Consider the following words: Floor and Flour.

Floor and Flour are pronounced highly differently despite having the same first letters, and we humans take u in Flour into account when pronouncing them correctly. At this point, one would argue that utilizing an RNN to capture such features should be enough, as RNNs effectively collect time correlation in data. This is most likely true for small-range correlations, but capturing long-term dependency is difficult. Convolutions further strengthen the model's resistance to words with silent letters.

Instead of feeding the embedded characters straight to an RNN, we extract N-grams using convolutional layers and feed the output features to an RNN to capture a longer-term context. The T2 encoder can capture the long-term dependency between characters by feeding n-grams to the RNN. As a result, Tacotron-2 can capture sentence context to distinguish between past and present versions of words.

Consider the following scenario: we have an input sequence X from which we extract encoder outputs H:





where {K\_{x}} is the input tokens' vocabulary size (space of accessible tokens: characters + symbols) and T\_{x} is the input sequence length.

We first compute the convolutional encoder features f e by convolving embedded character inputs with the sequential convolution filters F\_{1}, F\_{2}, and F\_{3}, and then applying all nonlinearity to each convolutional layer. (The letter E stands for embeddings.)



The hidden encoder outputs H: are generated by feeding these features into a bi-directional LSTM Cell.



Bi-directional RNN ensures that the model "reads" the input sequence from left to right and from right to left, providing knowledge about each input character's "past and future." As a result, the annotation H j comprises summaries of the preceding and following words. Because RNNs prefer to reflect recent inputs, the annotation H j will be focused on the words around X\_{j}.

A bidirectional RNN is made up of two independent RNNs: an overrightarrow{f} forward RNN that reads the input sequence in order (from f\_{1} to f\_{T\_{x}}) and an overleftarrow{f} backward RNN that reads the input sequence in reverse order (from f\_{T\_{x}} to f\_{1}). (overrightarrow 1,...,overrightarrow T x) and (overleftarrow H\_{1},...,overleftarrowH\_{T\_{x}}) are their outputs, respectively. The forward and backward hidden states are concatenated in the final encoder outputs

H = (H\_{1},...,H\_{T\_{x}}):



The sigmoid function is represented by green squares, followed by the analogous activation function.

The following is a mathematical expression for LSTM Cells:











In T2, we pass the encoder outputs (hidden states) to an attention network, consuming these hidden states and generating the context vector.

Unlike the original Tacotron-1 encoder, which consisted of a prenet+CBHG block, the T2 encoder is a large-scale \*\*simplification \*\* of its predecessor. T2 encoders are often over-simplified CBHG blocks in which the convolution bank, projections, and highway net are replaced with a stack of three basic convolutions with relu activations. Experiments that swapped T2 encoders for T1 encoders and vice versa yielded remarkably similar results, implying that T2 encoders provide equal performance with reduced processing cost.

**Attention Mechanism:**

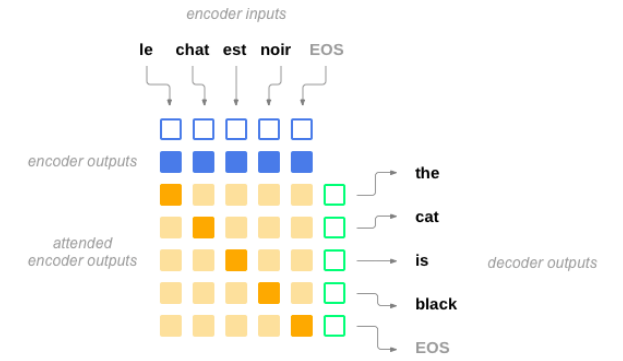
We have covered the Encoder part in general, and concerning Tacotron 2, we will cover the Attention part. Firstly we will know the Attention mechanism in general and later concerning Tacotron 2.

The cognitive process of selectively concentrating on one or a few things while disregarding others is referred to as Attention.

A neural network is a computer program that attempts to simplify human brain actions. In deep neural networks, the Attention Mechanism is called the same process of selectively concentrating on a few essential things while disregarding others.

Attention was first introduced by Bahdanau et al. (2015) as a direct relationship between encoder-decoder inputs and outputs. To put it another way, Attention is a mechanism that connects decoder and encoder outputs. Because each output is generated one at a time, Attention allows the decoder to "attend" or "pay attention" to the bits of the inputs that are most significant for the current output.

Now, we know how to define attention mechanisms; now, we will understand how attention mechanisms work with the help of a general example.



*Fig- Attention mechanisms in Machine translation*

Attention (Alignments), as seen in this diagram of a neural machine translation example, is a direct relationship between created decoder outputs and encoder outputs, resulting in a correlation between the encoder-decoder network's outputs and inputs. Orange The highest values from the alignment matrix are represented by colored cells. One might see that "Cat" is related to "Chat," which is the French term for Cat.

The decoder learns not only to anticipate outputs one at a time but also to align outputs with related inputs, thanks to a tiny Attention Network, which is often a parallel multilayer feed-forward network. The model is no longer obliged to decode sequences using the Attention it has learned.

By learning Attention, the model is no longer compelled to decode sequences using a short compressed fixed-length thought vector, which is unsuccessful for long sequences. When we consider the case with Tacotron 2, the model cannot read long phrases and sentences without the help of the Attention mechanism. So, we are using Attention in our model as it will help in reading and decoding long phrases.

If we look deep into attentions, they are broadly classified into types:

1. Bahdanau style Attention (Additive)
2. Luong style Attention (Multiplicative)

Now, let us look deeper into these sub-topics of Attention.

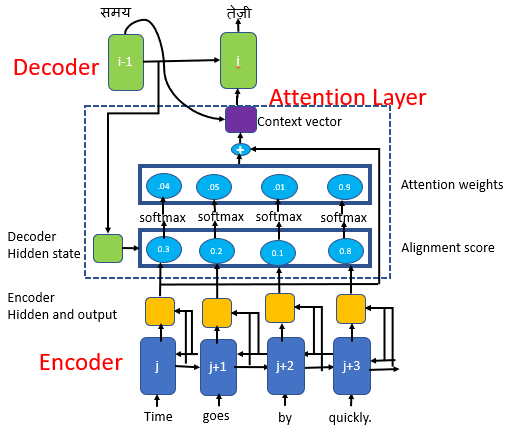
**Bahdanau Style Attention**

The attention mechanism was named after the original author of the paper in which it was published, Bahdanau.

It builds on the work of Cho et al. (2014) and Sutskever et al. (2014), who used an RNN encoder-decoder system to encode a variable-length source sentence into a fixed-length vector for neural machine translation. The latter would then be decoded into a target sentence with a changeable length.

According to Bahdanau et al. (2014), it is encoding a variable-length input into a fixed-length vector squashes the information of the source sentence, regardless of its length, leading to the performance of a basic encoder-decoder model to degrade fast as the input sentence length increases.

On the other hand, the solution they offer replaces the fixed-length vector with a variable-length one to improve the fundamental encoder-decoder model's translation performance.



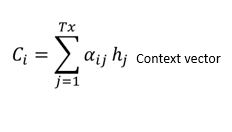
*Fig - Bahdanau Attention*

Let us now understand a bit more about Bahdanau's Attention.

1. From the input sentence, the encoder generates a set of annotations (\mathbf{h}\_i)
2. These annotations and the initial hidden decoder state are given into an alignment model. The attention scores (e\_{t, i}) are generated by the alignment model using this information.
3. The attention scores are normalized using the softmax algorithm, which converts them into weight values (\alpha\_{t, i}) in the range of 0 to 1.
4. Through a weighted sum of the annotations, these weights, together with the previously computed annotations, are used to build a context vector (\mathbf{c}\_t).

**Context Vector:**

The context vector is utilized to calculate the decoder's final output. The context vector ci, which maps to the input sentence, is the weighted sum of attention weights and encoder hidden states (h1, h2,...,htx).

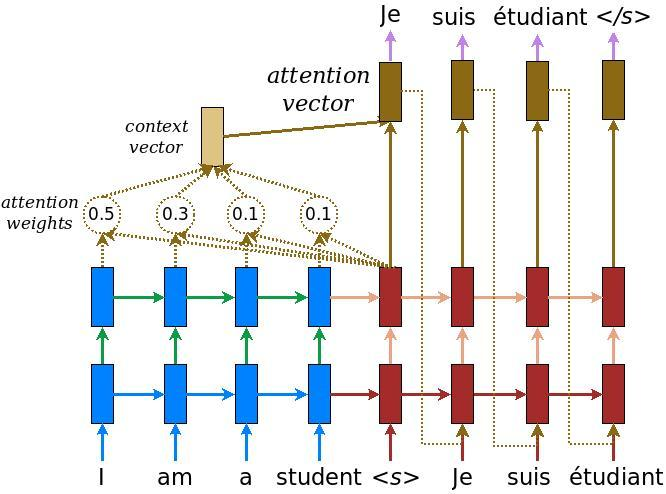


1. To compute the final output (y\_t), the context vector is supplied to the decoder, the prior concealed decoder state, and the previous output.
2. Steps 2–6 are continued until the sequence is completed.

Bahdanau et al. evaluated their design on an English-to-French translation job. They found that it outperformed the traditional encoder-decoder model by a wide margin, regardless of sentence length.

**Luong Style Attention**

Multiplicative Attention is another name for Luong's Attention. It uses basic matrix multiplications to convert encoder and decoder states into attention scores. It is faster and more space-efficient when you use simple matrix multiplication.



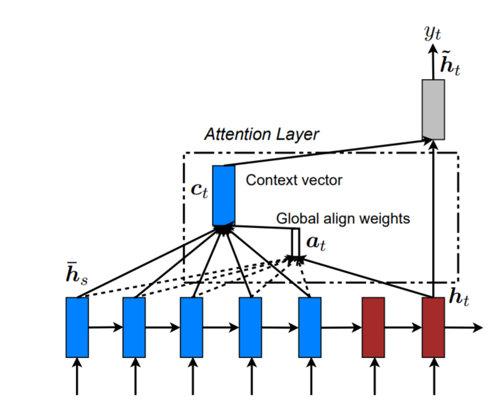
*Fig - Luong Style Attention*

Luong proposed two types of attention mechanisms based on where the focus is placed in the source sequence.

1. Global Attention
2. Local Attention

Now, let us study a bit about **Global Attention**.

When constructing the context vector ct, the global attentional model considers all of the encoder's hidden states. By comparing the current target hidden state ht with each of the source's hidden state hs, a variable-length alignment vector is generated equal to the number of time steps in the source sequence. The alignment score is a content-based function that considers three different options.



*Fig - Global Attention*

Because the global attention model considers all of the words in the source sequence to forecast the target words, it is computationally expensive and challenging to translate lengthy phrases.

**Local Attention**

Unlike global Attention, which focuses on the entire source sequence, local Attention only focuses on a limited subset of source places per target word. Global Attention is computationally less expensive. At time t, the local attention model creates an aligned position Pt for each target word. The context vector ct is calculated as a weighted average of the source hidden states within the window specified. The aligned position might be chosen in a predictable or monotonous manner.

**Computation of Attention in Bahdanau and Luong attention mechanisms.**

In their non-stacking unidirectional decoder, Bahdanau et al. combine the forward and backward hidden states in the bi-directional encoder and the primary target's hidden states.

In both the encoder and decoder, Loung et al. use hidden states at the top LSTM layers.

The alignment vector is computed by the Luong attention mechanism using the current decoder's hidden state, whereas Bahdanau uses the output of the previous time step.

In Tacotron 2, we will mainly focus on Bahdanau's Attention, i.e., Additive style attention.

**Content-based Attention**

At each decoding step, we must compute a context vector, also known as an attention vector, in order for the decoder to map each decoding step to each input character. We already know about the context vector during Bahdanau's Attention. Here we will see the mathematical equations regarding the context vector.

Let the computed Context vector be {c i }

The context vector {c i} is produced as a weighted sum of the encoder annotations {h j} or as previously termed {H j} in the original Attention presented by Bahdanau.

Let us now represent this in a mathematical equation,



{c i} = \sum\_{j=1}^{T\_{x}} \alpha\_{ij} h{j}

{alpha\_{ij}} in this equation is called **Attention weights** or **alignments**

The context vector is computed concerning all encoder outputs to discover the most important ones, as shown in the preceding equation. The question remains as to how these alignments are computed.

Different attention mechanisms diverge at this point. Soft, hard, or monotonous Attention are terms we hear from time to time. I will discuss soft Attention in this document.

As the name implies, we compute soft alignments between the decoder and encoder outputs. Softmax, which we employ in the computation of attention weights, comes with soft alignments:



Here in the above equation, e{ij} is a score usually called Energy

The methods for calculating this Energy vary across Bahdanau and Luong's works and between different forms of Attention (content-based, location-based, and hybrid Attention).

For content-based attentions



The ability to associate different outputs with the corresponding input tokens, regardless of their position, was demonstrated by content-based Attention. When producing Mel spectrogram frames matching to an "s" character, utilizing content-based Attention would teach the model to look at most, if not all, "s" characters. Because this is not how we humans read sentences, it is probably not the most excellent option.

**Location-Based Attention**

The only distinction between location-based Attention and content-based Attention is how the network scores e\_{ij}:



F I,j are location characteristics derived by convolution prior alignments alpha i-1 with convolution filters F. The weights v a, W, U, and F must be learned.

The content of the input tokens is unimportant to location-based Attention, which is simply interested in their locations and the distances between them. Without the decoder hidden states as a source of information, it should not be easy to correctly forecast distances between consecutive phonemes and their corresponding input tokens. An example is a quiet in the middle of the output sequence due to a "," token in the input sequence.

**Hybrid Attention**

As its name implies, hybrid Attention combines the two previously stated attention systems. Like both of them, the only difference is in the computation of Energy.



Prior decoder RNN hidden states are s i-1, previous alignments are alpha\_{i-1}, and the jth encoder hidden state is h\_{j}. The trainable parameters are W, V, U, and v\_{a}. The location features are also f\_{i, j}:



We usually retain all linear transformations in score computations bias-free. However, we can include biases to make the attention computation:



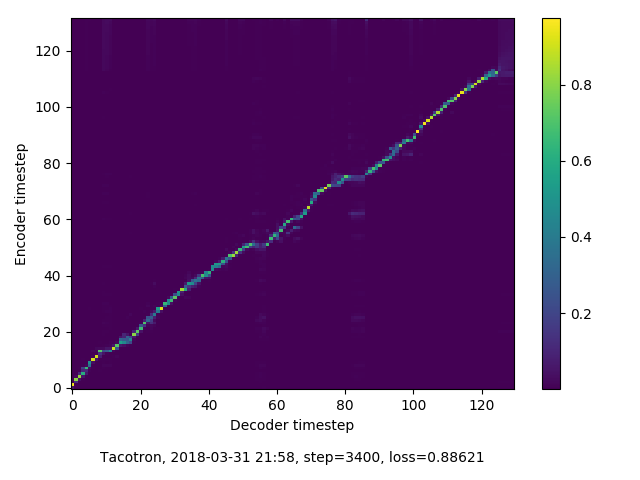
Mathematically, it is straightforward to merge the three bias terms into a single b vector, decreasing the number of model parameters and teaching the model to attribute values to the summed biases. The final energy calculation is as follows:



The hybrid Attention considers both the content and the location of input tokens, aiming to maximize the benefits of both previous attentions while overcoming their limitations. We built the Tacotron-2 "Location Sensitive Attention" based on Hybrid Attention since it produces superior results.

**Tacotron 2 Attention**

Finally, let us talk about the "Location Sensitive Attention" used in T2. We can assume that the context vector computation is similar to all previously seen attentions. We prefer to use the location-sensitive attention from, which extends the additive attention mechanism to use cumulative attention weights from previous decoder time step. ere is our interpretation, which has produced reasonable outcomes since the very beginning of training.



*Fig - Interpretation of Tacatron 2*

We compute the T2 Attention as follows.

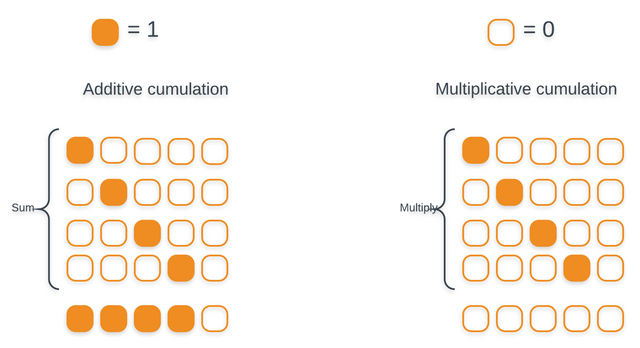


Instead of using the initial RNN hidden state, we utilize s\_{i}, which is the current decoding step RNN hidden state (reason explained down below in the decoder section). We set the bias b to a vector of zeros and anticipate the model to update them only when necessary; otherwise, they should converge to values near 0. The location features are calculated as follows using the cumulated alignments c\alpha\_{i-1}:





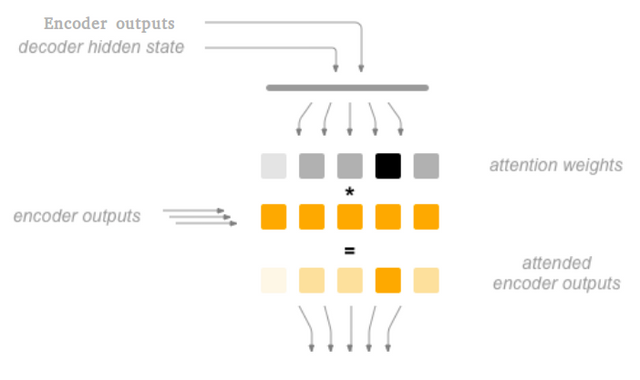
We assumed that the cumulation of alignments is done additively, as shown analytically. Why did we employ an additive cumulation rather than a multiplicative one, one could wonder? Initially, it was a decision based on pure intuition. However, a graphic demonstrates that in the event of perfect alignment, multiplicative cumulation is prone to losing the position information we are looking for:



*Fig- Types of Cumulations*

We feed the attention network information about input characters it has already attended to with such cumulated past alignments. It can use this to keep moving forward in the sequence and prevent repeating unpleasant voices.

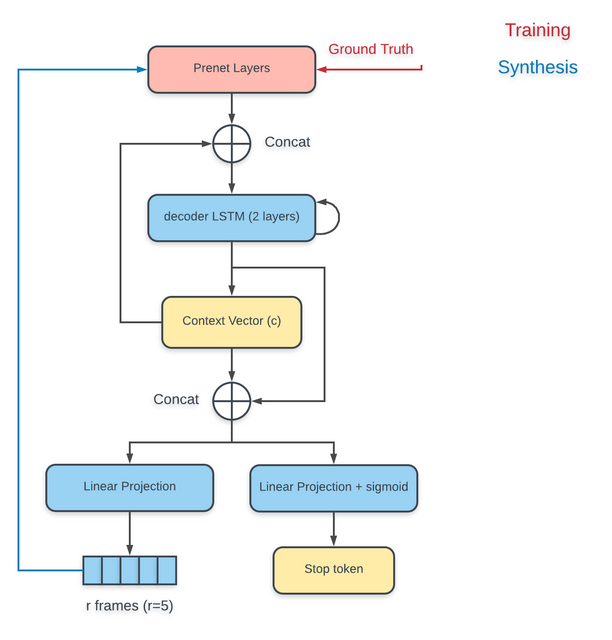
To summarise, we add a few convolutional connections that the model uses to learn alignments to match the decoder outputs directly to the input tokens. The context vector is then determined by comparing alignments to encoder inputs, which provides information about the most relevant token in the input sequence. To summarise the procedure here is a basic diagram:



*Fig- Attention Diagram Procedure*

**Decoder :**

The details of the Tacotron-2 decoder, i.e., Attention, hidden states, and outputs are computed at each decoding step, can be summarized in the following chart.



*Fig - Decoder*

The previous output spectrogram frame is fed to the Pre-Net layers as the first step in decoding. Synthesis particular actions are seen in blue, while training specific actions are shown in red in the previous chart.

The preset's outputs are then concatenated with the context vector produced in the previous decoding stage and supplied to the RNN decoder as input. This is commonly referred to as "input feeding."

The RNN decoder's output is utilized as a query vector to construct the new context vector.

Finally, the newly computed context vector is concatenated with the decoder outputs (hidden states) and given to projection layers to predict the output (spectrogram frame) and the stop token> probability.

The form of the scoring function, the attention function, whether the previous state [h i-1] is used instead of [h i] in the scoring function determine these versions. We determined empirically that only a few options matter:

1. Direct links must be present between the destination (output) and the source (input).
2. Feed the attention (context) vector to the next timestep (input feeding) to update the network about previous attention decisions.
3. Different scoring functions might typically result in different results.

Based on a thorough study, we decided to use an "ala Luong" context vector computation, in which the context vector is computed after the prediction of a new hidden state [s i] but before creating the output [y i]. When computing the new hidden states, we also feed the earlier context vector [c i-1] to the decoder to ensure the model understands the Attention it attributed during the previous decoding step. The procedure is known as "input feeding."

In decoders, projection layers project RNN hidden states to output space. The prenet, as an information bottleneck, was essential for learning Attention, according to the authors of the T2 article. The phrase "information bottleneck" relates to the concept of shifting the preceding mel-spectrogram frame to a hidden representation level that the decoder-RNN can understand.

Consider it as though you have two representation spaces, one of which is the frequency space utilized in Mel-spectrograms. The other is an abstract space in which these men frames' network view is stored. The present aims to translate frames from frequency space to hidden abstract space, allowing the RNN to forecast the future frames.

We support a reduction factor in our approach, which allows the decoder to anticipate r (reduction factor) Mel spectrogram frames in a single decoding step. This resulted in faster processing and less memory use. We also tried with more excellent reduction factors, as stated in the T2 study, and came to the following conclusions:

* Aside from reducing memory use and speeding up computations (train/test times), r=2 or r=3 allow for better alignments by allowing each decoding step to approximation model more significant sections of a phoneme (75-100ms), allowing for a more certain relationship between frames and text.
* When using Griffin-Lim, however, acoustic quality (voice clarity) was superior with r=1. This is because r=1 generated spectrograms have more high-frequency information than those with higher reduction factors.
* For WaveNet, r=1, r=2, or r=3 make no difference in acoustic quality as long as training is done in GTA mode.
* Although the reduction factor has little effect on prosody quality, we did find that r=1 has a somewhat less flat speech than the others.
* We employ r=1 for our final pre-trained models to ensure that our GL and WaveNet T2 models have good acoustic and prosody quality.

As a result, the i-th decoder step can be represented mathematically as the following set of equations:









**Post-Net**

Finally, after the decoder has finished decoding, the anticipated Mel spectrogram is sent through a stack of convolutional layers with tanh nonlinearities to increase the output quality. This is most likely owing to convolutional layers' capacity to catch features in images or sequences, mainly because they see both past and future context in spectrograms that have already been decoded. A final linear projection layer to the output space follows these convolutions.

The Post-Net output, known as "residual," can be calculated mathematically using:



where f\_{ps} is a stack of 5 convolutions with tanh activations that computes the post-net convolutional features:



for the first Post-Net layer, with x, the output of the previous convolutional layer or decoder output

Finally, we aggregate the decoder spectrogram outputs with the residual to obtain the final features of the Spectrogram prediction network:



**Vocoder**:-

It will convert acoustic features of a speech signal into audio waveforms. It constructs the audio waveforms by extracting features from the log scale Mel spectrogram.

Acoustic features refer to pitch, quality of the speech, and mel spectrum cepstrum coefficients.

There are two methods for building a vocoder:-

1. Auto-regressive models
2. Non-auto-regressive models

Auto-regressive models:-

It is a seq-to-seq model. It predicts the future inputs by taking the past outputs.

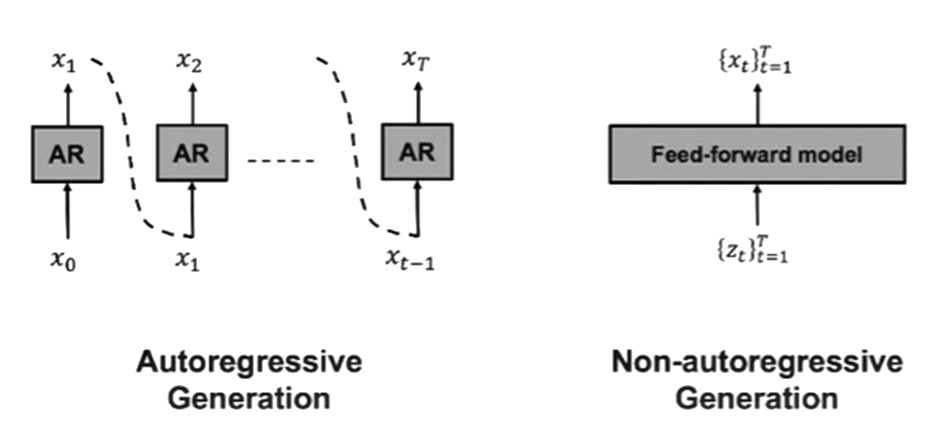


Fig :*Auto-regressive model*

In the above figure, we can see that in the middle block, to predict the output of x2, it needs the output of the previous block. Each block's output will be fed into the next block as inputs.

The sound quality will be clear without any noise, and the concatenation of the phonemes will be done, giving good results. The major disadvantage of using auto-regressive models is the speed of the outputs.

To predict the output of each block requires the output of the previous blocks. It will be time-consuming.

Non -regressive auto models:-

This model is based on parallelization. It enables us to generate all model outputs independently without depending on the previous outputs. It is independent of states.

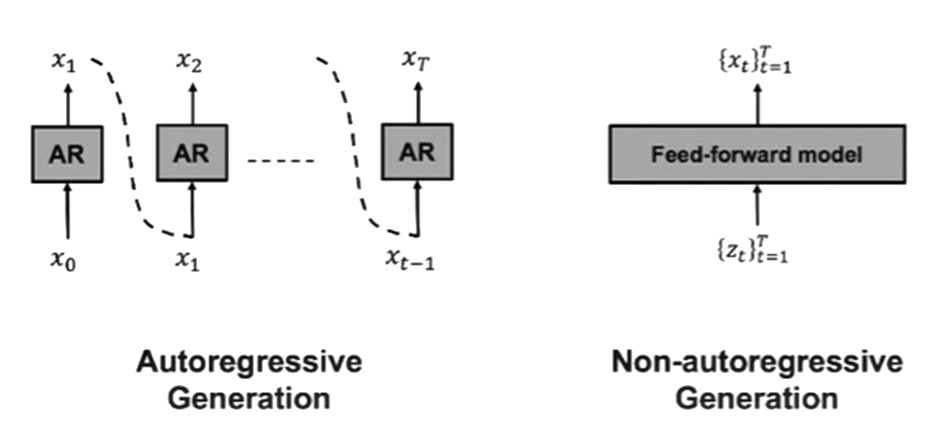


Fig : *Non-Autoregressive Model*

Here, the sound waveform speed is improved drastically compared with autoregressive models. The increase of drastic change in speech production is because of parallelization in the model. The trade-off for the waveform's speed of production is that the waveform's quality will be reduced, and training the model will be tough.



Fig : *Auto-Regressive Model vs Non Auto-Regressive Model*

**Brief introduction of Griffin Lim's vocoder**:-.

The Griffin and Lim's algorithm recovers an audio signal given only the magnitude of its Short-Time

Fourier Transform (STFT), also known as the spectrogram. It is an iterative algorithm that attempts to find the signal having an STFT such that the magnitude part

is as close as possible to the modified spectrogram.

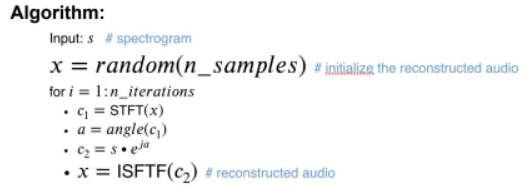


Fig *Griffin-Lim Algorithm*

The Griffin-Lim algorithm converges after 30 to 50 iterations. The Griffin-Lim produces characteristic artefacts and lowers audio quality than approaches like WaveNet.Nevertheless, the Griffin-Lim spectrogram inversion is efficient and allows backpropagation of

derivatives since it is differentiable. Therefore, it could be the initial choice when debugging a new end-to-end system.

**Waveglow**:-

WaveGlow is a generative model that generates audio by sampling from a distribution. To use a neural network as a generative model, we take samples from a simple distribution, in our case, a zero mean spherical Gaussian with the same number of dimensions as our desired output, and put those samples through a series of layers that transforms the simple distribution to one which has the desired distribution. In this case, we model the distribution of audio samples conditioned on a mel-spectrogram. z ∼ N (z; 0, I) (1) x = f 0 ◦ f 1 ◦ . . . f k (z) (2) We would like to train this model by directly minimizing the negative log-likelihood of the data. If we use an arbitrary neural network this is intractable. Flow-based networks [11, 12, 1] solve this problem by ensuring the neural network mapping is invertible. By restricting each layer to be bijective, the likelihood can be calculated directly using a change of variables: log pθ(x) = log pθ(z) +X k i=1 log | det(J(f −1 i (x)))| (3) z = f −1 k ◦ f −1 k−1 ◦ . . . f −1 0 (x) (4) In our case, the first term is the log-likelihood of the spherical Gaussian. This term penalizes the l2 norm of the transformed sample. The second term arises from the change of variables, and the J is the Jacobian. The log-determinant of the Jacobian rewards any layer for increasing the volume of the space during the forward pass. This term also keeps a layer from just multiplying the x terms by zero to optimize the l2 norm. The sequence of transformations is also referred to as a normalizing flow [13]. Our model is most similar to the recent Glow work [1], and is depicted in figure 1. For the forward pass through the network, we take groups of 8 audio samples as vectors, which we call the ”squeeze” operation, as in [1]. We then process these vectors through several ”steps of flow”. A step of flow here consists of an invertible 1 × 1 convolution followed by an affine coupling layer.

**Brief introduction of GAN architecture**:-

Ian Goodfellow developed generative adversarial neural networks in June 2014. It is an unsupervised learning algorithm where the input is unlabelled. It is majorly used in image processing and image reconstruction.

The basic architecture of GANs:-

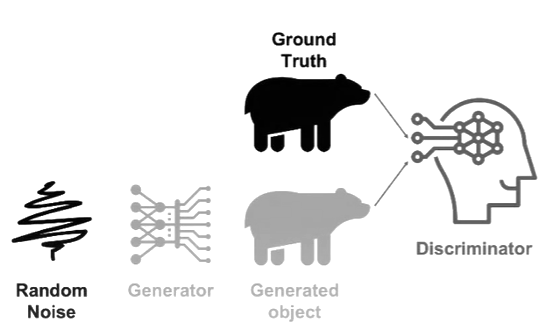


Fig 3.5 *Generative Adversarial Networks methodology*

There are two neural networks in generative adversarial neural networks

1. Generator

1. Discriminator

The generator takes the features of the given application, in our case, the acoustic features like pitch and quality of the speech. It adds random noise to the features we had taken, combines those, and feeds them into the neural network. The generator gets trained and gives the output as a generated audio waveform.

The discriminator takes the generator's output and compares it with the original audio waveform given as training data. The discriminator checks whether the generated output is real or fake speech. We take the loss function of both the generator and discriminator and gradient them to their optimal parameters. The generator will train until it fools the discriminator. We are fine-tuning data and refining our model parameters through this process.

**Short-time Fourier Transform**:-

This method converts the one-dimensional speech signal into a 2-dimensional speech signal to tell us how the frequency of the speech signals varies with time.

There are two auxiliary losses in short-time Fourier transform:-

1)Local convergence short-time Fourier transform loss

The graph shows that the low frequency and mid-frequency components with high power are emphasised

Using this spectral converange loss as an auxiliary loss, the generator will train to generate high-powered components centred on the low frequencies of the speech.

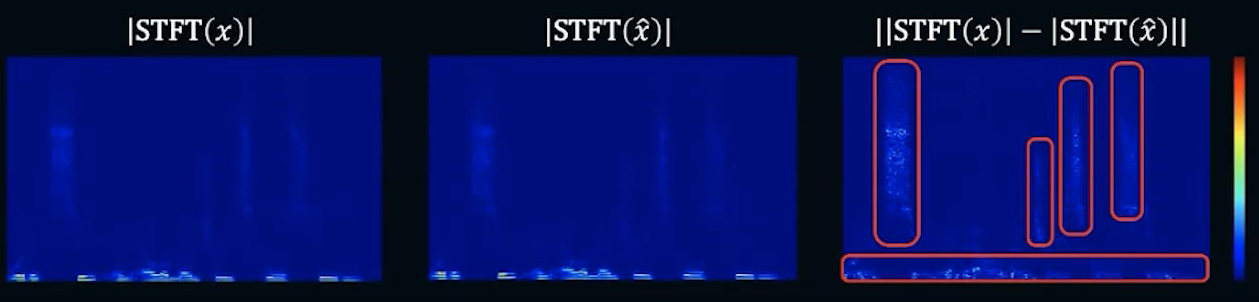


Fig: *Spectral convergence STFT loss*

The above figure shows that the STFT of the natural speech |STFT(X)| and STFT|X' | are differentiated to get the auxiliary loss.

The equation to calculate Spectral Convergence:-



2)Log-scale short-time Fourier transform magnitude loss:-

The difference between the first and second losses is that we take both STFT|X| & STFT|X' | and differentiate it to find out the magnitude loss of the speech signal.

The above figure shows that the high-powered components are spread throughout the mid-wide speech band. By taking the log, the difference between two STFTs is even more emphasised in the frequency band, even with a small amplitude.

By taking a log of STFT loss, the generator will train to reproduce the detailed structure of the speech.

The spectral convergence and log scale magnitude loss are complementary to each other.

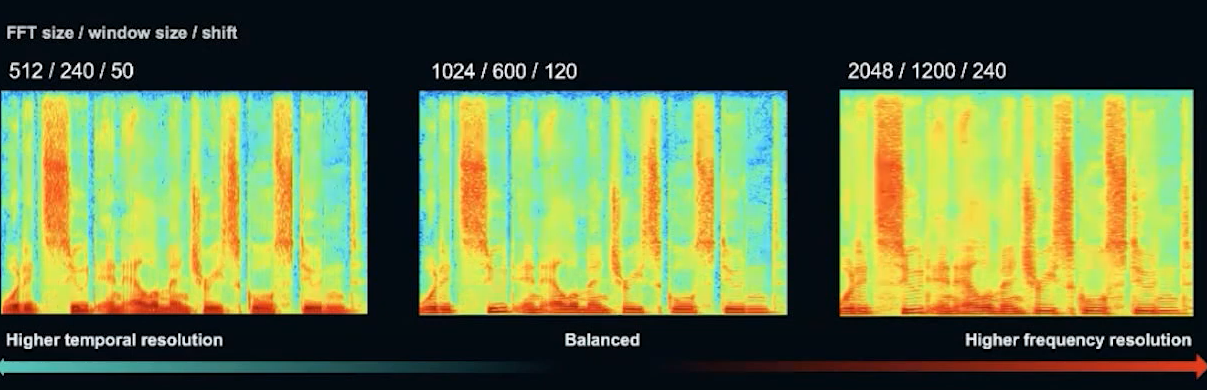


Fig :log-scale STFT magnitude loss

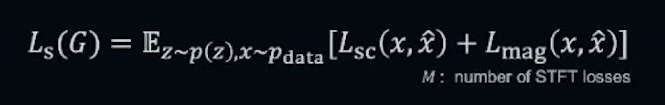
The two losses mentioned above are combined to form a multi resolution. However, there is a trade-off between the time resolution and frequency resolution in the time-frequency representation of STFT.

For example, if we try to increase the time resolution of the speech signal, the frequency resolution will become rough. The time-domain will become rough if we try to increase frequency resolution.

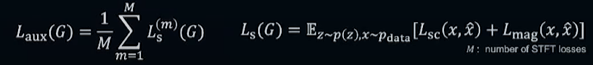
A simple solution is to combine different time-frequency representations to solve this issue.

In the case of parallelowavegan, three types of STFT loss combine to prevent the generator from overfitting into one time-frequency representation.

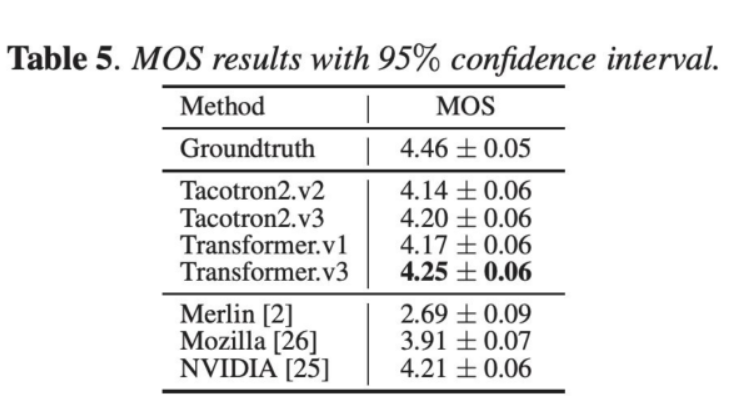
The equation to calculate the log-scale STFT magnitude loss:-



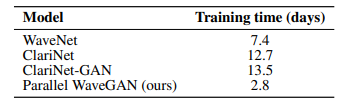
The below equation can represent the multi-resolution of the auxiliary loss:-



**Results**:-



.



NVIDIA Tesla V100 GPUs

**Summary:-**

In this project, we successfully built a functional and working model of Text-to-Speech (TTS) for the Hindi Language.

The TTS model we have built using Tacotron 2 and Parallel WaveGAN architecture generated sound waveforms similar to that of a human voice. It was seen that the output was robust, and it did not take much time to compute in the decoding step. However, it would be better to include the concept of voice cloning, and it would be nice if the model supports multilingual speech with comprehensive vocabulary sets.