CS230

Listen, Speak in Hindi!

Text-To-Speech Synthesis in Hindi

Dinesh Chaudhary*
chdinesh@stanford.edu

Abstract

We observe lack of Deep Learning models for Indic Languages and implement a parametric Text-To-Speech Synthesis model for Hindi. The model is fully-convolutional with attention mechanism and positional encodings and without any recurrent nodes, leading to faster training time. The model takes Hindi text sentences as input and generates Spectrogram and audio files as an output. We observe that it is possible to build a TTS model with reasonable accuracy and naturalness, on a small training dataset without any feature engineering and within a training time of less than 72 hours. We experiment with architecture given in Tachibana et. al [20] and DeepVoice3 [13]. Code for this project and synthesized examples are available at: https://github.com/chdin/hi-tts

1. Introduction

India is home to over 121 languages and 270 mother tongues [**7**]. With population consisting of over 272 million illiterates [7] and 62 million visually impaired, TTS systems for Indic languages should get more attention. For instance, even with new tech invading life of every Indian, it is surprising that the default language for most web applications is English, a language with just 129 million speakers in India [23]. As an example, taxi apps such as Ola and Uber give directions only in English, while most of the taxi-drivers speak Indic languages.

Most Indic languages scripts are phonetic in nature (one to one correspondence between what is written and what is spoken) and share common phonetic base. Hindi is the most spoken Indic language and fourth most spoken language in the world. 528 million people in India speak Hindi and its dialects as their mother tongue [7]. Our objective is to build a deep learning TTS model for Hindi.

2. Related Work

TTS models have shifted from concatenative to parametric with less reliance of hand-engineered modules. In 2016, Google Wavenet [11] used a fully-convolutional auto-regressive model with convolutional layers having various dilation factors. Tacotron [22] and Tacotron2 [18] use recurrent sequence-to-sequence models for more end-to-end model. DeepVoice [1] retained the modules of traditional TTS pipeline but trained these modules using neural network. DeepVoice3 [13] and DC-TTS (Deep Convolutional TTS) Model of Tachibana et. al. [20] though are fully-convolutional with attention mechanism.

For Hindi TTS, Indic TTS consortium has released TTS system for 13 Indian languages based on Hidden Markov Model [3]. Convolutional attention based model for multi-lingual multi-speaker synthesis covering four Indic languages and English is discussed in Baljekar, P. et. al. [5].

3. Data Summary

We have used Indic TTS data collected by TTS Consortium funded by Department of Electronics and Information Technology, Government of India [8]. We used Hindi dataset consisting of 5.2 hours of audio from a single female speaker. Dataset was split randomly into 80:15:5 training, validation and test datasets respectively. Summary statistics and a sample data point alongwith its audio's Spectrogram and Mel-Spectrogram are provided below:

*SUID 06349844, chdinesh@stanford.edu, CS-230, Deep Learning, Winter 2019, Stanford University, C.A.

Table 1: Data Summary Statistics

Data	Min Len	Mean Len	Max Len	Total
Text (in characters)	16	97	377	2318 files
Text (in words)	3	20	77	2318 files
Audio (in seconds)	1.5	8.1	29.5	5.2 Hours

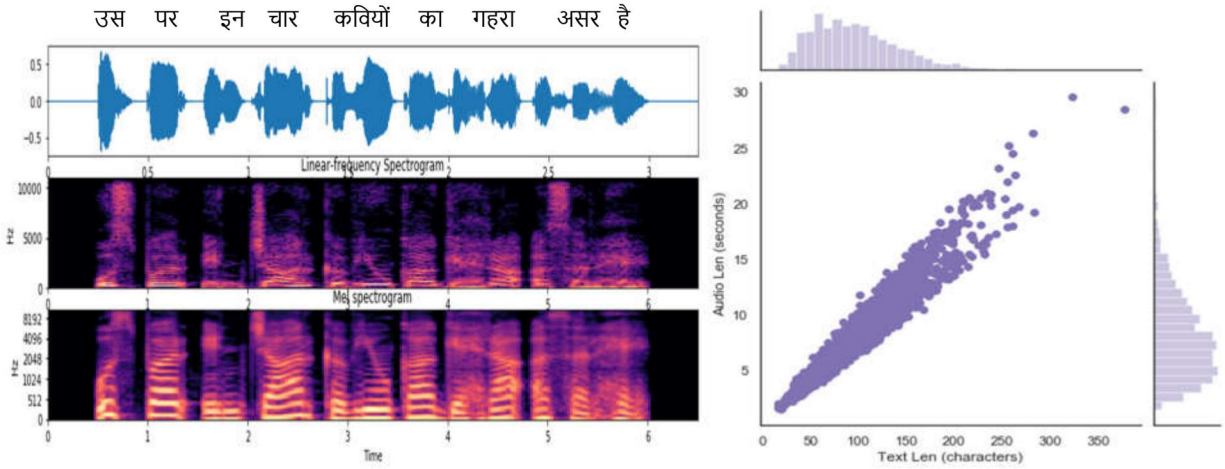

Figure 1: A sample from dataset alongwith Spectrogram and Mel-Spectrogram; Frequency distribution of length of input files

4. Model Architecture

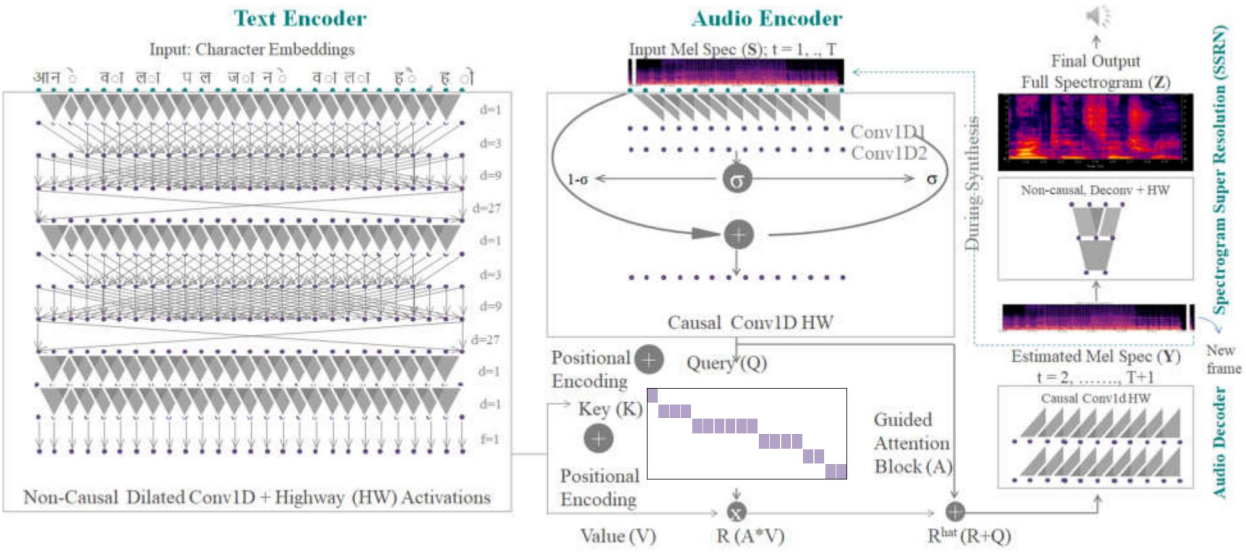

Figure 2: Model Architecture [20]. To simplify, we have shown convolutional details only for Text Encoder and highway activations only for Audio Encoder, even though these are part of all modules. Model consists of two main modules viz. Text2Mel that generates mel spectrogram Y from text and SSRN that generates full spectrogram Z from mel-spectrogram Y.

$$(K, V) = \text{TextEnc}(L). \qquad A = \text{softmax}_{n\text{-axis}}(K^{\mathsf{T}}Q/\sqrt{d}) \qquad \mathcal{D}_{\text{bin}}(Y|S) := \mathbb{E}_{ft}[-S_{ft}\log Y_{ft} - (1-S_{ft})\log(1-Y_{ft})]$$
$$Q = \text{AudioEnc}(S_{1:F,1:T}). \qquad R = \text{Att}(Q, K, V) := VA \qquad = \mathbb{E}_{ft}[-S_{ft}\hat{Y}_{ft} + \log(1+\exp\hat{Y}_{ft})],$$

4.1. Character Embeddings and Audio Files

We use a list of 71 characters consisting of 33 consonants, 14 vowels appearing in their independent and dependent form (Hindi vowels have different characters when attached with a consonant), punctuation, padding and end-of-sentence characters. In our experiments, we have used pre-trained as well as learned character embeddings. For pretrained embedding, we have used 300D Hindi character embeddings from FastText [16]. Hindi Script viz. Devanagari has no concept of letter case and the data did not consist of numeric figures. Therefore no other pre-processing was done on text files. Audio files were converted into spectrogram for training the SSRN module and mel-spectrogram for training the Text2Mel module. Audio files did not have any background noise or long silences.

4.2. Synthesis

As compared to RNNs, we use CNN based architecture for efficient training [13, 20], with following modules:

- Text2Mel consists of four sub-modules viz.:
    - Text Encoder: a non-causal module that converts character embeddings into encoded learned representation
    - Audio Encoder: a causal module that encodes mel-spectrogram of already spoken speech
    - Attention module: a guided attention module to align encoded audio with encoded text and
    - Audio Decoder: a causal module to generate mel spectrogram
- Spectrogram Super Resolution Network (SSRN): module to generate spectrogram from mel-spectrogram

Model uses stacked dilated convolutional layers to understand long-term context information in text sentences. Highway activations are used to avoid the problem of vanishing gradients. Use of non-causal convolutions in Text Encoder means that text synthesis can happen only after receiving the full text sentence as an input from the user.

To learn the network parameters, Text2Mel loss is computed as the sum of L1 loss and binary cross-entropy loss between predicted and ground-truth mel-spectrogram and error is backpropgated. SSRN loss is similarly computed as the sum of L1 and cross-entropy loss between ground truth and synthesized spectrogram. Attention matrix is forced to be nearly diagonal as we may expect text and audio to progress in proper sequence over time (unlike machine translation tasks). Attention loss is simultaneously optimized with Text2Mel loss with equal weight.

Table 2: Model Hyperparameters

HP for inputs	Value	HP for Architecture	Value	HP for training, optimization	Value
Character Embedding (e)	128 or 300	Kernel Size	3	Batch size	16
Sampling rate of audio	22050 Hz	Encoder, Decoder channels, Converter channels	256	Adam (β1, β2)	0.5, 0.9
STFT window length, shift	1024, 256	Query position rate	1,	Adam (α, ε)	0.0005, 10^{-6}
Spectrogram, Mel size	513x4T, 80xT	Key position rate	1.385	LR schedule	Noam
Pre-emphasis factor	0.97	Position encoding	Sinusoidal	Dropout	0.1/0.4

5. Results and Model Variations

We experimented with multiple architecture and hyperparameter choices and compared each variation after 80k iterations. To select the final model, we could not evaluate using a formal test such as Mean Opinion Score (MOS). Based on subjective analysis of performance on random samples from validation dataset, we selected a final model

(M2) and trained it further to 160k iterations. To formally evaluate the performance of the final model on validation and test datasets, we used the following custom metric:

$$\textbf{Normalized Word Recognition Rate (WRR)} = \frac{(1 - \text{Synthesized Audio WER})}{(1 - \text{Groundtruth Audio WER})}$$

We used Google Cloud Speech-To-Text API to convert synthesized audio to text ('synthesized text'). Synthesized text was then compared with original text to compute Model's Word Error Rate (WER). WRR for synthesized text was normalized by WRR for ground-truth audio files to account for maximum expected recognition rate of Speech Recognition System on that specific dataset. To summarise, we used a Speech Recognition System instead of Mechanical Turks and adjusted for the accuracy of Speech Recognition System. The metric maybe considered as conservative as an STT system would be trained on natural voices and therefore would penalise synthesized dataset more. Final model's performance on validation and test dataset is as follows:

Table 3: Final Model Performance on validation and test dataset

Metric	Validation	Test
Normalized WRR	73.1%	77.5%

We observe that final model's Normalized WRR is the lowest for shorter sentences and improves with sentence length. For shorter sentences, we observed relatively poor pronunciation as well as errors in attention alignment.

Table 4: Final Model Performance by length of input sentences (in Words)

Sentence length (in words)	% of cases in Val Set	Normalized WRR
<=12	21%	60%
13-17	25%	72%
18-21	17%	70%
22-27	18%	74%
28-33	9%	79%
>34	9%	77%

5.1. Model Variations

For our baseline model, we implemented DC-TTS with guided attention (with learned embeddings) as given in Tachibana et. al. and DeepVoice3 (with pretrained and learned embeddings) with monotonic attention. The three baseline models produced good quality sound on training dataset but failed to synthesize satisfactorily on validation data. For subsequent experiments, we focussed on DC-TTS model only. As the baseline models were overfitting on training data and were not converging on the val data, we experimented with three other models. Iterations were limited to 80k for faster experimentation.

- M1: Increase dropout from 5% in baseline DC-TTS model to 40%: Character-wise pronunciation on validation dataset improved as compared to baseline models, but attention alignments failed to converge.
- M2: M1 + sinusoidal position encoding as in DeepVoice3 + learned character embeddings
- M3: M1 + sinusoidal position encoding + pre-trained character embeddings

Based on subjective analysis as mentioned earlier, we selected M2 as the final model and trained it for 160k iterations to arrive at the final model. The final model learns alignment after 30-50k iterations for most of validation samples. As training loss is still reducing with each epoch, model performance may improve further by training the model for more iteration.

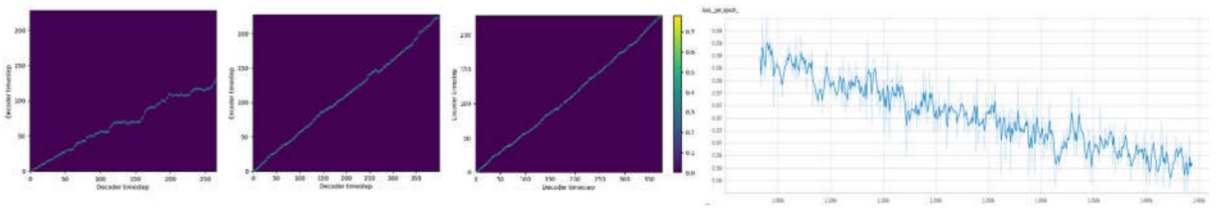

Figure 4: Learned alignments for a validation sample at 10k, 30k and 160 iterations. Training loss per epoch for final model (M2)

6. Observations

In our experiments, we have used pretrained (M3) as well as learned character embeddings (M1 and M2). As Hindi is a phonetic language, we may expect to see convergence between character embeddings pretrained on text-only corpus and character embeddings learned from a TTS system. Pretrained embeddings for most of the vowels (blue circle and oval), guttural 'back-of-the-mouth' consonants (orange triangle) and retroflex 'tongue-curled' consonants (gray oval) are in close neighbourhood. Pretrained embedding also provide some other interesting insights. For instance, embedding vector for compound character (characters that combine consonant with a vowel) with sound 'ka' minus embedding for vowel with sound 'aa' plus embedding of vowel with sound 'ee' has embedding vector for compound character 'ki' as its closest neighbour, which is how these characters are used in practice.

Learned embedding also fall in close neighbourhood for characters that have similar sounds or that originate from a certain region. However, except for retroflex consonants degree of convergence with pretrained embeddings is low. This may be due to smaller dataset for TTS. For three vowels, we observe that embeddings for their dependant and independent form converge in close neighbourhood.

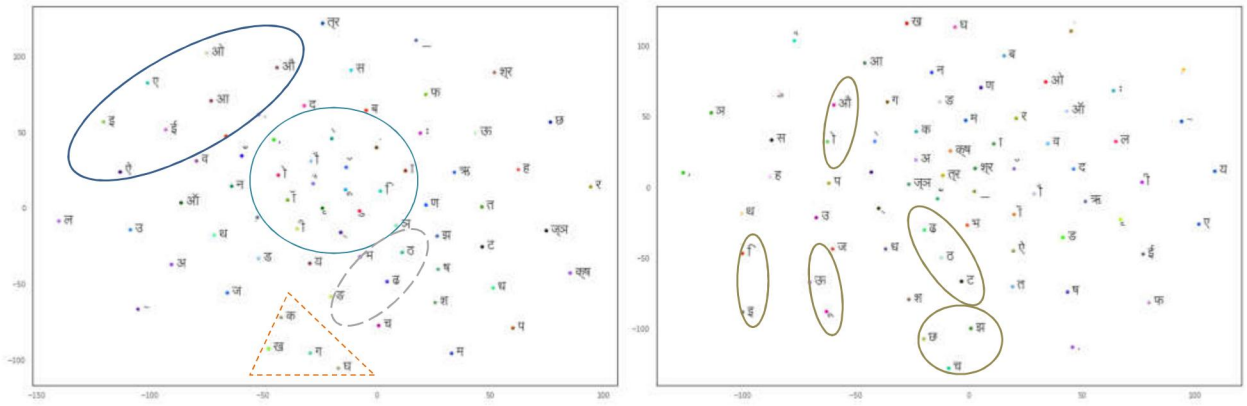

Figure 5: Pretrained Embeddings from FastText vs. Learned Embeddings from our final TTS Model (T-SNE)

7. Summary and Further Work

We build a Hindi TTS system using fully convolutional architecture with learned character embeddings, positional encoding and a guided attention mechanism. The model can be trained in less than 72 hours on a single 8GB GPU machine and works reasonably well on medium and long sentences. The audio quality can be improved further by tuning other hyperparameters that we did not tune. A mixed character and phoeneme model as in DeepVoice3 may also improve model performance. Training two separate models for short and longer sentences may also help attention errors to reduce. We would further like to extend the model to multi-speaker and multi-lingual synthesis. From real-life implementation perspective, we would like to build a new TTS model on code-mixed data consisting of Hindi transliterated into English and normal English sentences.

Contributions

The project team consisted of one member.

References

[1] Arik, S. et. al. (2017) DeepVoice: Real-time neural Text-to-Speech

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin (2017) Attention is all you need

[3] Atish, S. et. al. (2017) TBT (Toolkit to Build TTS): A High Performance Framework to build Multiple Language HTS Voice

[4] Baby, A., Thomas, A. L., L, N. N., and Consortium, T. (2016) Resources for Indian languages.

[5] Balijekar, P. Speech Synthesis from Found Data

[6] Bojar, O. et. al. (2014) HindEnCorp – Hindi-English and Hindi-only Corpus for Machine Translation

[7] Census of India (2011) Office of the Registrar General, India. http://censusindia.gov.in/2011Census

[8] Indic TTS. https://www.iitm.ac.in/donlab/tts/

[9] Librosa. https://librosa.github.io/librosa/

[10] Mahajan, A. Attention, I am trying to Speak

[11] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior and Koray Kavukcuoglu (2016). WaveNet: A generative model for raw audio

[12] Pallavi, B. (2014) Speech Synthesis from Found Data

[13] Ping, W., Peng, K., Gibiansky, A., Arik, S. O., Kannan, A., Narang, S., Raiman, J., and Miller, J. (2018) Deep Voice 3: 2000-speaker neural text-to-speech.

[14] Park, K. Tensorflow Implementation of Deep Voice 3 https://github.com/Kyubyong/deepvoice3

[15] Pytorch. https://pytorch.org/

[16] PytorchNLP. https://pytorchnlp.readthedocs.io/en/latest/source/torchnlp.word_to_vector.html

[17] Skerry-Ryan, R., Battenberg, E., Xiao, Y., Wang, Y., Stanton, D., Shor, J., Weiss, R. J., Clark, R., and Saurous, R. A. (2018) Towards end-to-end prosody transfer for expressive speech synthesis with Tacotron

[18] Shen, J. et. al (2018) Natural TTS synthesis by conditioning wavenet on Mel Spectrogram predictions

[19] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015) Highway networks. ICML 2015

[20] Tachibana, H., Uenoyama, K., and Aihara, S. (2018) Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention

[21] TSNE. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

[22] Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., et al. (2017) Tacotron: A fully end-to-end text-to-speech synthesis model

[23] Wikipedia. https://en.wikipedia.org/wiki/Hindi, https://en.wikipedia.org/wiki/Devanagari, https://en.wikipedia.org/wiki/Languages_of_India

[24] Yamamoto, R. DeepVoice3 PyTorch and DC TTS Implementation. https://github.com/r9y9/deepvoice3_pytorch

[25] Yann Dauphin, Angela Fan, Michael Auli, and David Grangier. (2016) Language modeling with gated convolutional networks.