NEURAL NETWORK INVERSION: DERIVING DECODER FROM ENCODER FOR TTS

LPC COEFFICIENT BASED TTS USING TEDLIUM DATA.

Eliezer Zimble

Uni: ez2313

ABSTRACT

Text to Speech (TTS) systems have been successful in producing high quality natural sounding audio from text input. Our project develops a lightweight TTS by using neural network inversion on an Encoder trained for ASR. The project does so by training an Encoder-Decoder network based on Linear Predictive Coding (LPC) coefficients extracted from the benchmark TEDLIUM3 corpus. The project develops a Frontend based on a BLSTM recurrent neural network that turns text into speech embeddings which in turn the Decoder network uses to produce LPC coefficients.

Index Terms— TTS, Encoder-Decoder, LPC

1. INTRODUCTION

The task of automatically converting Text to Speech (TTS) has important applications in a broad array of fields [1] including communications, artificial intelligence, speech processing and aid to those speech impaired. The goal of TTS platforms is to use text input to produce natural sounding speech, and the success of state of the art neural network based approaches for the single speaker have allowed research to expand to more complex tasks such multi-speaker TTS and low resource-TTS [1]. The problem formulation for modern TTS systems remains the production of natural sounding audio, with further goals including robustness to varied inputs and settings, and operation under constrained resources.

2. FORMULATION FROM THE STATE OF THE ART

The state of the art TTS platforms use fully end-to-end neural networks that generate waveform output directly from text [1], [2] or incorporate an acoustic model as part of the same training block [3]. These networks typically employ a transformer architecture and achieve low word error rate (WER) [4]. The TTS formulation in such systems is the training of a neural TTS with a single training process that incorporates all steps needed to take text as input and produce natural sounding audio in waveform. While effective, such models are typically costly as they require a large amount of training data and are composed of a very large number of weights.

3. APPROACH AND FORMULATION

Overall, our project researches the problem of low-resource TTS systems and approaches the problem though deriving a Decoder network for single-speaker TTS by means of inverting an LPC based Encoder neural network trained to perform speech to text. As a coding scheme, the output LPC coefficients are lightweight and can be easily stored or transmitted.

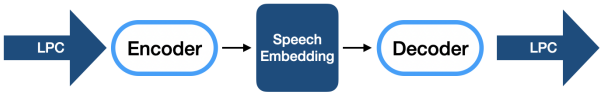
The approach of our project is to develop a TTS system that consists of 3 separate blocks: (1) a Frontend that takes in text as input and produces a speech embedding as its output, (2) a Decoder that takes turns the speech embedding into LPC coefficients, and (3) a vocoder that transforms the LPC coefficients into the desired audio. A high level overview of the overall system is presented in Figure 1. The choice of LPC coefficients as the feature type is due to the fact that LPC coefficients contain phase information [5] that will aid in the reconstruction of audio.

Fig. 1. High level structure of overall platform

The main focus and crux of our project is the development of the Decoder block. Our approach is to train an Encoder-Decoder network as a means of neural network inversion. The Encoder network is trained for Automatic Speech Recognition (ASR) and trained on LPC coefficients. The Encoder network through an intermediary level produces a speech embedding that is used as the input into the Decoder network. The construction of the decoder layers takes into consideration the detailed architecture of the Encoder network. The decoding layers are designed to perform the reverse of the corresponding encoding layers and thereby achieve TTS. The Encoder-Decoder network takes LPC coefficients and ivectors as input, produce an intermediary speech embedding layer, and then reverse the layers to produce predicted LPC coefficients back as output. The encoding layers have the learning rate frozen to 0 to force the decoder layers to learn to take speech embeddings as input and produce predicted LPC coefficients as output. A

high level overview of the Encoder-Decoder network is presented in Figure 2.

Fig. 2. High level structure of Encoder-Decoder network for LPC



Further, our project developed a Frontend block that takes text as input and produces speech embeddings to match the input of the Decoder network. The Frontend does so by using text processing, phoneme duration modeling and training a BLSTM to take phoneme integer vectors and output speech embeddings of the correct dimension.

Taken together the Frontend and Decoder constitute a pipeline that takes text as input and produces predicted LPC coefficients as output. Future work will use the LPC coefficients together with a Vocoder to reconstruct the audio and thereby achieve a low cost TTS platform.

3.1. Encoder Network Architecture

The Encoder network is an Automatic Speech Recognition system with the same architecture and setup as the TEDLIUM3 [6] recipe HMM-TDNN-f chain model for ASR. However, the Encoder network uses custom Kaldi scripts to train on and take order 20 LPC coefficients as input features, while the TEDLIUM model uses 40 dimension MFCC features.

The architecture of the Encoder network is composed of several types of layers from the input through the final outputs. The Encoder network begins by appending the LPC feature input with the neighboring time shifted frames (using $t = -1$ and $t = 1$) for overall input feature dimension of 60. Additionally the Encoder network takes input of dimension 100 ivectors. The Encoder network uses an LDA transform layer as a means of de-correlating the data (without dimension reduction) on the feature input and ivectors. The LDA layer therefore has input and output dimension of 160. The next layer is a TDNN layer with input and output dimension of 1024. The Encoder network follows with 12 TDNN-F layers with input and output dimension of 1024, bottleneck dimension of 128, and bypass scale of 0.66. For TDNN-F layers 2 and 3 a stride of 1 is used, for layer 4 a stride of 0, and for all remaining TDNN-F layers have a stride of 3.

The Encoder network has 2 outputs: the posterior probabilities (PDFs) for transcriptions and the log-prob (negative cross-entropy) loss as a linear objective function. For each output the Encoder network has a prefinal layer with input dimension 1024 and output dimension 256 before producing the final outputs each of dimension 3600 (corresponding to the PDFs for 3600 distinct tri-phones).

3.2. Encoder-Decoder Network Architecture

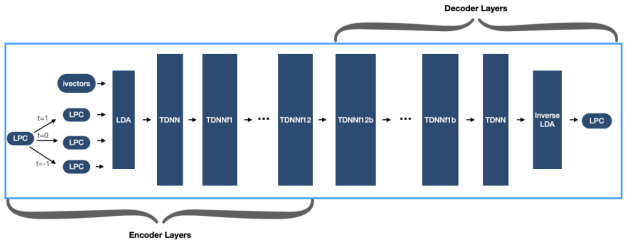
For construction of the Encoder-Decoder network, the encoding layers are formed by copying the input layer through the speech embedding layer of 12th TDNN-f layer from the Encoder network and setting the learning rate for all components to 0. The encoding layers overall takes input of 20 dimension (order) LPC coefficients (per frame) and 100 dimension ivectors (per speaker) and outputs a 1024 dimension speech embedding (per frame).

The decoding layers are added to the network as the reverse of the encoding layers. In particular, 12 TDNN-f layers are added mirroring the encoding layers with input and output dimension of 1024, bottleneck dimension of 128, and bypass scale of 0.66. The decoding TDNN-f layers are added in reverse such that the overall Encoder-Decoder network has a chiastic structure. The following layer is a TDNN layer with input and output dimension of 1024. A custom Kaldi binary is used to reverse the LDA transform (input and output dimension of 160) and extract the components corresponding to the unshifted ($t = 0$) order 20 LPC coefficients. The decoding layers thereby take an input of 1024 dimension speech embedding per frame and produces a 20 dimension (order) predicted LPC coefficients per frame.

Overall the Encoder-Decoder network takes in 20 dimension (order) LPC coefficients (per frame) and 100 dimension ivectors (per speaker), and outputs 20 dimension (order) predicted LPC coefficients (per frame).

For the Encoder-Decoder network, the objective function is mean squared error loss, following the Kaldi convention of $-\frac{1}{2}(x \cdot y)^T(x-y)$. A more detailed presentation of the layers of in Encoder-Decoder network architecture is presented in Figure 3.

Fig. 3. Encoder-Decoder Architecture

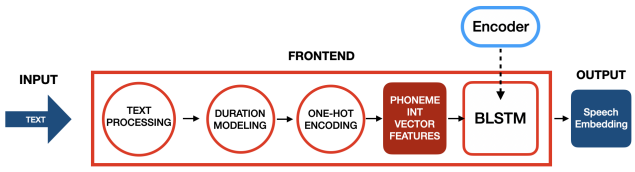


3.3. Frontend Design and Architecture

The Frontend block consists of several stages that serve as a pipeline to take raw text as input and generate speech embeddings as output. The frontend output will be passed as input into the Decoder network. The stages are (1) convert the text into phoneme integer vectors, (2) apply duration modeling, (3) transform and normalize the feature vectors with one-hot

encoding, (4) pass the new features through a BLSTM network to generate the desired 1024 dimensional speech embeddings. See Figure 4. for an overview of the frontend design.

Fig. 4. High level structure of Frontend block



3.3.1. Text to Phoneme Integer Vector

The TEDLIUM lexicon is used to convert words to phonemes. In turn, the TEDLIUM phoneme information is used to map each of the 184 monophones to its integer symbol. A word is thereby transformed into a phoneme integer representation. We compose the mappings and concatenate the output of words to achieve a straightforward process for transforming text into new features of phoneme integer vectors. The length of the vector is the number of phonemes in the text.

3.3.2. Duration modeling

The Frontend employs a basic phoneme frame duration model by using the phoneme frame level alignments of the training data to form a mapping from phoneme to average duration. A fixed phone in the input is concatenated for the phone's average frame duration length.

Further, the frames produced for each phone are concatenated by an additional multiplicative factor corresponding to the windowing used during feature extraction. The reason for this is that we want to get the frontend input to match the number of frames in terms of the number of LPC features for the utterance (such as used in Kaldi "utt2num_frames" files). The baseline repetition factor is set at 3 frames. The Frontend performs text processing to ensure that the new features are the correct length (in particular for during training). When the predicted duration modeling is shorter than the expected number of frames, the Frontend performs padding with the integer value for non-silence noise. When the predicted duration modeling is longer than the expected number of frames, the Frontend computes and applies the necessary "dilution factor" by which to reduce phoneme repetition values in order to close the gap.

The intermediary features after the duration modeling are vectors where the length is the predicted number of frames (in terms of LPC features for the utterance).

3.3.3. One-hot encoding

The Frontend transforms the phoneme integer vector features into one hot encoding vectors: for a fixed frame X with phoneme integer value x, the Frontend constructs a new feature of a 184 dimensional vector (corresponding to the number of distinct monophones) that has value 1 at coordinate x and is 0 otherwise. The features after the one-hot encoding have dimension (width) 184 and length corresponding to the predicted number of frames (in terms of LPC features for the utterance).

The one-hot encoding acts as a form of normalization. We want to prevent phones with higher valued integer mappings from receiving a disproportionate weight during the BLSTM training. The phoneme to integer mappings is merely a label and should not influence the values for the loss function. The one-hot encoding makes it so that every phone has a value of 1 for the "hot" component and therefore all phones receive equal weight in the BLSTM neural net training.

3.3.4. BLSTM

The Frontend finally uses the new features as input for a Bidirectional Long Short Term Memory RNN (BLSTM). The BLSTM network consists of 3 bi-directional LSTM levels of cell dimension 1024 with recurrent projection dimension 128 and non-recurrent projection dimension 128. The BLSTM produces speech embeddings of dimension 1024 for each frame of input. The final output of the Frontend block is therefore features of dimension (width) 1024 and length corresponding to the predicted number of frames (in terms of LPC features for the utterance).

4. EXPERIMENTS AND RESULTS

The data used by our project is the open source TEDLIUM 3 Legacy Corpus [6] which is split into training, development and test sets. The training set overall contains 452 hours of audio and automatically aligned transcriptions from TED talks, totaling 474,090,535 number of frames. The training set contains 2351 number of talks with 2028 unique speakers. The development set consists of 8 TED talks given by unique speakers with manual aligned transcriptions for a total of 574,327 number of frames. The test set consists of 11 TED talks given by unique speakers with manual aligned transcriptions for 939,889 number of frames. The TEDLIUM training, test and development sets were used for the training and evaluation of each of the Encoder Network, Encoder-Decoder network, and the Frontend block. The details are presented in the following sections.

4.1. Encoder network training

We used the Tedlium training data together with adapted and modified Kaldi scripts to extract LPC coefficients for

the training, test and development sets. The Encoder network training follows the TEDLIUM [6] recipe experiments through the development of an HMM-TDNN-f chain model using order 20 LPC coefficients as input features in place of MFCCs. Following the Tedlium recipe, the experiments begin by using LPC features to train, align and rescore monophone and triphone Gaussian Mixture Models (GMM). The triphone GMM training is repeated to produce a triphone GMM with context dependence and speaker-adaptation The encoder training then performs speed perturbation to the data and alignment with the cleaned tri-phone GMM. The speed perturbed data is used to generate high resolution LPC coefficients for the training, test and development sets. The high resolution LPC coefficients are used to extract ivectors of dimension 100 for the training, test and development sets. The GMM alignments are converted into lattices and the topology is built as a tree for use in the chain model training.

Finally, the Encoder HMM-TDNN-f chain model is trained using the training set high resolution order 20 LPC coefficients and 100 dimension ivectors as the training input, and alignment from the GMM lattices and topology to generate the neural network training output. The training runs for 6 epochs and uses minibatches of size 64.

4.2. Encoder network Results

The Encoder network was evaluated in terms of Word Error Rate (WER) on the TEDLIUM [6] original test and development sets and rescored test and development sets compared to the transcriptions. While the WER is a metric for Automatic Speech Recognition (ASR) and not TTS per se, the evaluation gives us insight into the performance of the Encoder network. The Encoder network results are compared to those of the original state of the art TEDLIUM3 HMM-TDNN-f trained on the same corpus (using MFCCs) and evaluated with the same test and development sets. The results are presented in Table 1. and indicate that the Encoder network achieved slightly worse but comparable results to that of the TEDLIUM3 chain model.

Table 1. Comparison of Encoder and Tedlium WER results

Encoder WER results				
Model	WER original		WER rescored	
	Dev	Test	Dev	Test
Encoder (LPC)	8.94%	8.88%	8.31%	8.36%
Tedlium Chain (MFCC)	8.22%	8.45%	7.60%	7.73%

4.3. Encoder Decoder Training

For the Encoder-Decoder network training, the learning factor rate for the encoding layers is set to 0 and Kaldi scripts are used for training a dense target neural network. The Encoder-Decoder network is trained with high resolution training LPC coefficients and the training set ivectors as input, and as the "ground truth" output the high resolution training LPC coefficients. We want the Encoder-Decoder network to learn to produce predicted LPC coefficients back out. The Encoder-Decoder network training uses minibatches of size 512 and runs for 2 epochs.

4.4. Encoder Decoder Results

The Encoder-Decoder network was evaluated using a custom Kaldi script to compute the mean square error for the high resolution test and dev sets. For each set the high resolution LPC coefficients and respective ivectors were passed through the Encoder-Decoder network and compared to the high resolution LPC coefficients using MSE. The results are presented in Table 2. The results demonstrate the effectiveness of the decoder layers as a means of inverting the Encoder network.

Table 2. MSE results for Encoder-Decoder model

Encoder-Decoder Results	MSE	
	Dev	Test
Encoder-Decoder	-0.0785	-0.0854
(Number of frames)	(574,327)	(939,889)

4.5. Frontend Training

The Frontend training consists of training the BLSTM model with custom examples and producing a basic duration model that maps phones to their average number of repeated frames (as measured by feature length per utterance).

4.5.1. BLSTM training

For the training examples for the BLSTM, we prepared custom phoneme integer vector features as the input and computed the associated ground truth speech embeddings as output. In particular, for the Frontend training output the high resolution training set LPC coefficients and associated ivectors were passed through the Encoder network to compute the ground truth speech embeddings. For the Frontend training input, Kaldi scripts were used to extract the phoneme alignments per frame for the training set from the Encoder (Chain) model alignments. As the alignments are inferred and utterance lengths do not necessarily match the true number of frames, padding and shortening were applied when needed.

The computed examples were used to train the BLSTM network for 6 epochs using MSE as the loss function. The chunk size was set at 20, and both left and right context were set to 10.

4.5.2. *Duration modeling development*

Using the extracted phoneme training set alignments, custom scripts determine the average number of frame repeats per phone as a model of phoneme duration. We store the information as a mapping for use as the duration modeling block of the Frontend. As above in the Frontend design, we further repeat the frames for each phoneme to correspond with the expected feature length (due to windowing during feature extraction).

4.6. Frontend Results

The results of the Frontend block are presented in Table 3. The large value for the MSE demonstrates that the predicted speech embeddings are not suitable as input for the Decoder. The Frontend therefore requires further development before it can be properly implemented in a TTS system.

Table 3. MSE results for Frontend block

Frontend Results	MSE	
	Dev	Test
Frontend (Number of frames)	-2214 (574,327)	-2191 (939,889)

5. CONCLUSION AND FUTURE WORK

5.1. Conclusion

The project demonstrates that neural network inversion is achievable and in particular that the Encoder-Decoder LPC coefficient network can be an effective platform for TTS. The Encoder network and Encoder-Decoder network both perform well and the Frontend, while not performing well enough, demonstrates the proof of concept for the overall TTS platform. The project establishes the feasibility of an LPC based light-weight TTS system.

5.2. Future Work

Future work will focus on two main direction: improving the Frontend block and integration with a Vocoder.

5.2.1. *Further Frontend Development*

Currently the performance of Frontend block is limiting the overall TTS system. Future work therefore requires further development of the Frontend block in order to achieve high quality speech embeddings to pass into the Decoder. Approaches to explore include more robust duration modeling and quinphone labels for the text input [7], and incorporation of prosody models [8], [9].

Additionally, replacement of the one-hot encoding with a probability distribution based approach will likely make the input features more robust and improve the resulting MSE. In particular, the current one-hot encoding yields sparse 184 dimensional feature vectors with a single non-zero entry and may be prone to instability. The features can achieve a similar result to the normalization done by the one-hot encoding while being more robust by replacing the one-encoding with a probabilistic weighting scheme where the true phone has its associated component receive most of the weight and all other components a small randomized weight.

Further, training of the Frontend block may be improved by using datasets specially designed for TTS training such as The LibriTTS [10] and CSTR VCTK [11] corpora instead of the currently used TEDLIUM dataset (which was design primarily for ASR).

Finally, different architectures may need to be explored in place of the current BLSTM configuration in order for the system to properly learn to produce the desired speech embeddings.

5.2.2. *Vocoder Integration*

Completion of the overall TTS system requires the integration of the Frontend and Decoder with a neural LPC coefficient based Vocoder [12] [13] [14] in addition to traditional reconstruction using the LPC coefficients. The Vocoder block will take the predicted LPC coefficients and produce the desired audio output.

Additionally, usage of LPC coefficients together with a Vocoder allows for cross synthesis. Different trained or recorded excitation signals can be combined with the LPC coefficients to produce the audio output in a variety of voices for a given text. The cross-synthesis includes the potential to train one's own voice profile which may be of particular benefit for those speech impaired.

In the fully developed TTS system the integration with the Vocoder block would both produce waveform output and also allow choice of voice profile.

6. REFERENCES

[1] Xu Tan, Tao Qin, Frank Soong, and Tie-Yan Liu, "A survey on neural speech synthesis," 2021.

[2] Ron J. Weiss, RJ Skerry-Ryan, Eric Battenberg, Soroosh Mariooryad, and Diederik P. Kingma, "Wave-tacotron: Spectrogram-free end-to-end text-to-speech synthesis," 2020.

[3] Yanqing Liu, Ruiqing Xue, Lei He, Xu Tan, and Sheng Zhao, "Delightfultts 2: End-to-end speech synthesis with adversarial vector-quantized auto-encoders," 2022.

[4] Ruiqing Xue, Yanqing Liu, Lei He, Xu Tan, Linquan Liu, Edward Lin, and Sheng Zhao, "Foundationtts: Text-to-speech for asr customization with generative language model," 2023.

[5] J. Makhoul, "Linear prediction: A tutorial review," *Proceedings of the IEEE*, vol. 63, no. 4, pp. 561–580, 1975.

[6] Franç ois Hernandez, Vincent Nguyen, Sahar Ghannay, Natalia Tomashenko, and Yannick Estève, "TED-LIUM 3: Twice as much data and corpus repartition for experiments on speaker adaptation," in *Speech and Computer*, pp. 198–208. Springer International Publishing, 2018.

[7] Blaise Potard, Matthew Aylett, David Braude, and Petr Motlicek, "Idlak tangle: An open source kaldi based parametric speech synthesiser based on dnn," 09 2016, pp. 2293–2297.

[8] J.F. Pitrelli, R. Bakis, E.M. Eide, R. Fernandez, W. Hamza, and M.A. Picheny, "The ibm expressive text-to-speech synthesis system for american english," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 4, pp. 1099–1108, 2006.

[9] Raul Fernandez, Asaf Rendel, Bhuvana Ramabhadran, and Ron Hoory, "Using deep bidirectional recurrent neural networks for prosodic-target prediction in a unit-selection text-to-speech system," 09 2015.

[10] Heiga Zen, Viet Dang, Rob Clark, Yu Zhang, Ron J. Weiss, Ye Jia, Zhifeng Chen, and Yonghui Wu, "Libritts: A corpus derived from librispeech for text-to-speech," 2019.

[11] K. MacDonald et al. C. Veaux, J. Yamagishi, "Superseded-cstr vctk corpus: English multi-speaker corpus for cstr voice cloning toolkit," 2016.

[12] Jean-Marc Valin and Jan Skoglund, "Lpcnet: Improving neural speech synthesis through linear prediction," 2019.

[13] Jean-Marc Valin, Umut Isik, Paris Smaragdis, and Arvindh Krishnaswamy, "Neural speech synthesis on a shoestring: Improving the efficiency of lpcnet," 2022.

[14] Krishna Subramani, Jean-Marc Valin, Umut Isik, Paris Smaragdis, and Arvindh Krishnaswamy, "End-to-end lpcnet: A neural vocoder with fully-differentiable lpc estimation," 2022.

[15] Lin Ai, Shih-Ying Jeng, and Homayoon Beigi, "A new approach to accent recognition and conversion for mandarin chinese," 2020.

[16] Yuxuan Wang, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc V. Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous, "Tacotron: A fully end-to-end text-to-speech synthesis model," *CoRR*, vol. abs/1703.10135, 2017.

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

[18] Garcia Fuentes C Cardoso W, Smith G, "Evaluating text-to-speech synthesizers," 2015.

[19] Anil C Kokaram "Andrew Hines, Jan Skoglund and Naomi Harte", "Visqol: an objective speech quality model," "2015".

[20] J. S. Chung, A. Nagrani, and A. Zisserman, "Voxceleb2: Deep speaker recognition," in *INTERSPEECH*, 2018.

[21] Zvi Kons, Slava Shechtman, Alex Sorin, Carmel Rabinovitz, and Ron Hoory, "High quality, lightweight and adaptable tts using lpcnet," 2019.