EFFICIENT AND SCALABLE NEURAL RESIDUAL WAVEFORM CODING WITH COLLABORATIVE QUANTIZATION

Kai Zhen[1,2]*, Mi Suk Lee*[3]*, Jongmo Sung*[3]*, Seungkwon Beack*[3]*, Minje Kim*[1,2]

[1]Indiana University, Luddy School of Informatics, Computing, and Engineering, Bloomington, IN
[2]Indiana University, Cognitive Science Program, Bloomington, IN
[3]Electronics and Telecommunications Research Institute, Daejeon, South Korea

zhenk@iu.edu, lms@etri.re.kr, jmseong@etri.re.kr, skbeack@etri.re.kr, minje@indiana.edu,

ABSTRACT

Scalability and efficiency are desired in neural speech codecs, which supports a wide range of bitrates for applications on various devices. We propose a collaborative quantization (CQ) scheme to jointly learn the codebook of LPC coefficients and the corresponding residuals. CQ does not simply shoehorn LPC to a neural network, but bridges the computational capacity of advanced neural network models and traditional, yet efficient and domain-specific digital signal processing methods in an integrated manner. We demonstrate that CQ achieves much higher quality than its predecessor at 9 kbps with even lower model complexity. We also show that CQ can scale up to 24 kbps where it outperforms AMR-WB and Opus. As a neural waveform codec, CQ models are with less than 1 million parameters, significantly less than many other generative models.

Index Terms— Speech coding, linear predictive coding, deep neural network, residual learning, model complexity

1. INTRODUCTION

Speech coding quantizes speech signals into a compact bit stream for efficient transmission and storage in telecommunication systems [1, 2]. The design of speech codecs is to address the trade-off among low bitrate, high perceptual quality, low complexity and delay, etc [3, 4]. Most speech codecs are classified into two categorizes, *vocoders* and *waveform* coders [5]. Vocoders use few parameters to model the human speech production process, such as vocal tract, pitch frequency, etc [6]. In comparison, waveform coders compress and reconstruct the waveform to make the decoded speech similar to the input as "perceptually" as possible. Conventional vocoders are computationally efficient and can encode speech at very low bitrates, while waveform coders support a much wider bitrate range with scalable performance and are more robust to noise.

In both conventional vocoders and waveform coders, linear predictive coding (LPC) [7], an all-pole linear filter, serves a critical component, as it can efficiently model power spectrum with only a few coefficients through Levinson-Durbin algorithm [6]. For vocoders, the LPC residual is then modeled as a synthetic excitation signal with a pitch pulse train or white noise component [8]. On the other hand, for waveform coders, such as Opus [9], Speex [10] and AMR-WB [11], the residual is directly compressed to the desired bitrate before being synthesized to the decoded signal.

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (2017-0-00072, Development of Audio/Video Coding and Light Field Media Fundamental Technologies for Ultra Realistic Tera-media).

LPC is useful in modern neural speech codecs, too. While generative autoregressive models, such as WaveNet, have greatly improved the synthesized speech quality [12], it comes at the cost of model complexity during the decoding process [13]. For example, vector quantized variational autoencoders (VQ-VAE) with WaveNet decoder achieves impressive speech quality at a very low bitrate of 1.6 kbps, yet with approximately 20 million trainable parameters [14]. To make such a system more efficient, LPC can still unload computational overheads from neural networks. LPCNet combines WaveRNN [15] and LPC to shrink down the complexity to 3 GFLOPS which enables real-time coding [16, 17]. Nevertheless, LPCNet, as a vocoder, provides a decent performance at 1.6 kbps, but does not scale up to transparent quality. In terms of the neural waveform coder, CMRL [18] uses LPC as a pre-processor and a variation of [19] to model the LPC residual to match the state-of-the-art speech quality with only 0.9 million parameters. However, both LPCNet and CMRL take LPC another blackbox shoehorned into advanced neural networks. Using LPC as a deterministic preprocessor can be sub-optimal, as its bit allocation is pre-defined and not integrated to model training.

To better incorporate LPC with neural networks towards scalable waveform coding with low model complexity, we propose a collaborative quantization (CQ) scheme where LPC quantization process is trainable. Coupled with the other neural network autoencoding modules for the LPC residual coding, the proposed quantization scheme learns the optimal bit allocation between the LPC coefficients and the other neural network code layers. With the proposed collaborative training scheme, CQ outperforms its predecessor at 9 kbps, and can scale up to match the performance of the state-of-the art codec at 24 kbps with a much lower complexity than many generative models. We first illustrate relevant techniques which CQ is based upon in Section 2, and then explain how they are tailored to our model design in Section 3. In Section 4, we evaluate the model in multiple bitrates in terms of objective and subjective measures. We conclude in Section 5.

2. PRELIMINARIES

2.1. End-to-end speech coding autoencoders

A 1D-CNN architecture on the time-domain samples serves the desired lightweight autoencoder (AE) for end-to-end speech coding, where the model complexity is a major concern [19, 18]. As shown in Table 1, the encoder part consists of four bottleneck ResNet stages [20], a downsampling convolutional layer to halve the feature map size in the middle, and then a channel compression layer to create

Table 1. Architecture of the 1D-CNN autoencoders. Input and output tensors sizes are represented by (width, channel), while the kernel shape is (width, in channel, out channel).

Layer	Input shape	Kernel shape	Output shape
Change channel	(512, 1)	(9, 1, 100)	(512, 100)
1st bottleneck	(512, 100)	(9, 100, 20) (9, 20, 20) (9, 20, 100) ×2	(512, 100)
Downsampling	(512, 100)	(9, 100, 100)	(256, 100)
2nd bottleneck	(256, 100)	(9, 100, 20) (9, 20, 20) (9, 20, 100) ×2	(256, 100)
Change channel	(256, 100)	(9, 100, 1)	(256, 1)
Change channel	(256, 1)	(9, 1, 100)	(256, 100)
1st bottleneck	(256, 100)	(9, 100, 20) (9, 20, 20) (9, 20, 100) ×2	(256, 100)
Upsampling	(256, 100)	(9, 100, 100)	(512, 50)
2nd bottleneck	(512, 50)	(9, 50, 20) (9, 20, 20) (9, 20, 50) ×2	(512, 50)
Change channel	(512, 50)	(9, 50, 1)	(512, 1)

a real-valued code vector of 256 dimensions. The decoder is with a mirrored architecture, but its upsampling layer recovers the original frame size (512 samples) from the reduced code length (256).

2.2. Soft-to-hard (softmax) quantization

To compress speech signals, a core component of this AE is the trainable quantizer which learns a discrete representation of the code layer in the AE. Out of the recent neural network-compatible quantization schemes, such as VQ-VAE [21] and soft-to-hard quantization [22], we focus on soft-to-hard quantization, namely *softmax* quantization as in the other end-to-end speech coding AEs [19, 18]. Given an input frame $\boldsymbol{x} \in \mathbb{R}^S$ of S samples, the output from the encoder is $\boldsymbol{h} = \mathcal{F}_{\text{Enc}}(\boldsymbol{x})$, each is a 16-bit floating-point value. Given $J = 32$ centroids represented as a vector $\boldsymbol{b} \in \mathbb{R}^J$, softmax quantization maps each sample in $\boldsymbol{h}$ to one of J centroids, such that each quantized sample can be represented by $\log_2 J$ bits (5 bits when $J = 32$).

This quantization process uses a hard assignment matrix $\boldsymbol{A}_{\text{hard}} \in \mathbb{R}^{I \times J}$, where I and J are the dimension of the code and the vector of centroids, respectively. It can be calculated based on the element-wise Euclidean distance matrix $\boldsymbol{D} \in \mathbb{R}^{I \times J}$.

$$\boldsymbol{A}_{\text{hard}}(i,j) = \begin{cases} 1 & \text{if } \boldsymbol{D}(i,j) = \min_{j'} \boldsymbol{D}(i,j') \\ 0 & \text{otherwise} \end{cases} . \quad (1)$$

Then, the quantization can be done by assigning the closest centroid to each of $\boldsymbol{h}$'s elements: $\bar{\boldsymbol{h}} = \boldsymbol{A}_{\text{hard}}\boldsymbol{b}$. However, this process is not differentiable and blocks the backpropagation error flow during training. Instead, a soft-to-hard assignment is adopted as follows:

(a) Calculate the distance matrix $\boldsymbol{D} \in \mathbb{R}^{I \times J}$ between the elements of $\boldsymbol{h}$ and $\boldsymbol{b}$.

(b) Calculate the soft-assignment matrix from the dissimilarity matrix using the softmax function $\boldsymbol{A}_{\text{soft}} = \text{softmax}(-\alpha \boldsymbol{D})$, where the softmax function applies to each row of $\boldsymbol{A}_{\text{soft}}$ to turn it into a probability vector, e.g., $\boldsymbol{A}_{\text{soft}}(i,j)$ holds the highest probability iff $\boldsymbol{h}_i$ is most similar to $\boldsymbol{b}_j$. Therefore, during the training phase $\boldsymbol{A}_{\text{soft}}\boldsymbol{b}$ approximates hard assignments and is fed to the decoder as the input code, while still differentiable. The additional variable α controls the softness of the softmax function, i.e., $\lim_{\alpha \to \infty} \boldsymbol{A}_{\text{soft}} = \boldsymbol{A}_{\text{hard}}$. We use $\alpha = 300$ to minimize the gap between $\boldsymbol{A}_{\text{soft}}$ and $\boldsymbol{A}_{\text{hard}}$.

(c) At testing time, $\boldsymbol{A}_{\text{hard}}$ replaces $\boldsymbol{A}_{\text{soft}}$ by turning the largest probability in a row into one and zeroing the others. $\boldsymbol{A}_{\text{hard}}\boldsymbol{b}$ creates the quantized code $\bar{\boldsymbol{h}}$.

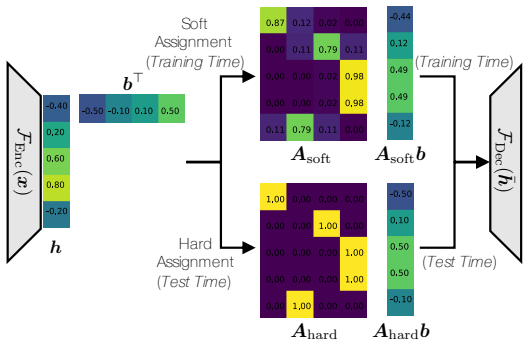

Fig. 1. An example of the softmax quantization process.

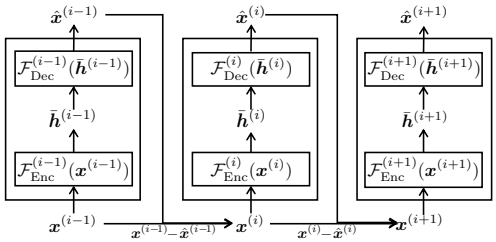

Fig. 2. The CMRL residual coding scheme.

Fig. 1 summarizes the softmax quantization process.

2.3. Cross-module residual learning (CMRL) pipeline

CMRL serializes a list of AEs as its building block modules to enable residual learning among them (Fig. 2). Instead of relying on one AE, CMRL serializes a list of AEs as building block modules, where the i-th AE takes its own input $\boldsymbol{x}^{(i)}$ and is trained to predict it $\hat{\boldsymbol{x}}^{(i)} \approx \boldsymbol{x}^{(i)}$. Except for the heading AE, the input of i-th AE $\boldsymbol{x}^{(i)}$ is the residual signal, or the difference between the input speech $\boldsymbol{x}$ and the sum of what has not been reconstructed by the preceding AEs: $\boldsymbol{x}^{(i)} = \boldsymbol{x} - \sum_{j=1}^{i-1} \hat{\boldsymbol{x}}^{(j)}$. CMRL decentralizes the effort of optimizing one gigantic neural network; lowers the model complexity in terms of trainable parameters to less than 1 million, which brings neural audio coding algorithms closer to smart devices with limited energy supply and storage space. The AEs in CMRL use the same model architecture in Section 2.1 and quantization scheme in Section 2.2.

3. COLLABORATIVE QUANTIZATION

In the CMRL pipeline, LPC module serves a pre-processor with a fixed bitrate of 2.4 kbps. While it can effectively model the spectral envelope, it may not fully benefit the consequent residual quantization. For example, for a frame that LPC cannot effectively model, CQ can weigh more on the following AEs to use more bits, and vice versa. In this section, we break down the LPC process to make its quantization module trainable, along with the other AE modules in CMRL which are to recover the LPC residual as best as possible.

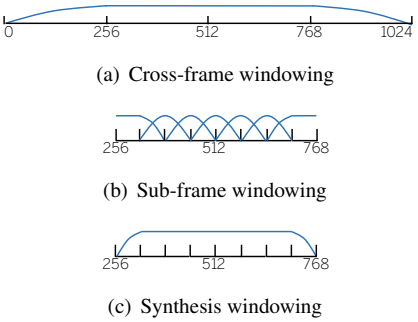

Fig. 3. LPC windowing schemes

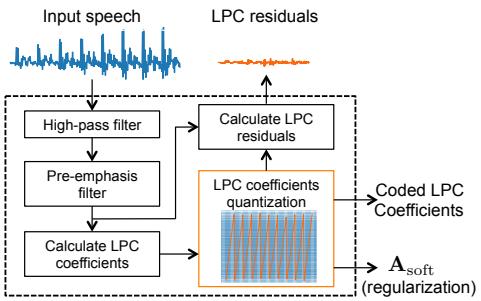

Fig. 4. The trainable LPC analyzer

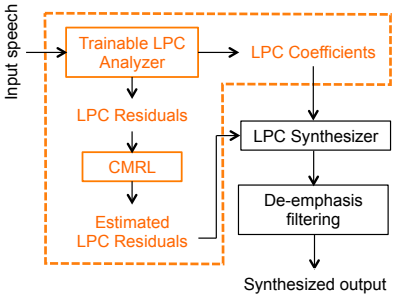

Fig. 5. Overview of the CQ system.

3.1. Trainable LPC analyzer

Our goal is to incorporate LPC analysis into the CMRL pipeline so that it outsources the LPC coefficient quantization to the neural network training algorithm. The trainable LPC analyzer is derived from AMR-WB [23] with several necessary adjustments to be compatible with neural network computational paradigm.

High-pass filtering and pre-emphasizing: Given the input speech, we first adopt high-pass filtering and pre-emphasizing as in [23]. A high-pass filter is employed with a cut off frequency of 50 Hz. The pre-emphasis filter is $H_{emp}(z) = 1 - 0.68z^{-1}$, and the de-emphasis filter is employed to remove artifacts in the high frequencies.

Data windowing for LPC coefficients calculation: The pre-emphasis filtered utterances are segmented to frames of 1024 samples. Each frame is windowed before LPC coefficients are calculated. As shown in Fig. 3 (a), the symmetric window has its weight emphasized on the middle 50% samples: first 25% part is the left half of a Hann window with 512 points; the middle 50% is a series of ones; and the rest 25% part is the right half of the Hann window. Then, the linear prediction is conducted on the windowed frame in time domain s. For the prediction of the t-th sample, $\hat{s}(t) = \sum_i a_i s(t-i)$, where a_i is the i-th LPC coefficient. The frames are with 50% overlap. The LPC order is set to be 16. We use Levinson Durbin algorithm [6] to calculate LPC coefficients. They are are represented as line spectral pairs (LSP) [24] which are more robust to quantization.

Trainable LPC quantization: We then employ the trainable softmax quantization scheme to LPC coefficients in LSP domain, to represent each coefficient with its closest centroid, as described in Section 2.2. For each windowed frame $\boldsymbol{x}$, $\boldsymbol{h}_{\text{LPC}} = \mathcal{F}_{LPC}(\mathbf{x})$ gives corresponding LPC coefficients in the LSP representation. The rest of the process is the same with the softmax quantization process, although this time the LPC-specific centroids $\boldsymbol{b}_{\text{LPC}}$ should be learned and be used to construct the soft assignment matrix. In practice, we set LPC order to be 16, and the number of centroids to be 256 (i.e., 8 bits). Hence, the size of the soft and hard assignment matrices is 16×256, each of whose rows is a probability vector and a one-hot vector, respectively.

Data windowing for LPC residual calculation: We use a sub-frame windowing technique to calculate residuals (Fig. 3 (b)). For a given speech frame and its quantized LPC coefficients, we calculate residuals for each sub-frame, individually. The middle 50% of the 1024 samples, for example, [256:768] for the first analysis frame that covers [0:1024] and [768:1280] for the second frame of [512:1536], is decomposed into seven sub-frames, each with the size 128 and 50% overlap. Out of the seven sub-frames, the middle five are windowed by a Hann function with 128 points; the first and last frames are asymmetrically windowed, as shown in Fig. 3 (b). The residual is calculated with the seven sub-frames on the middle 512 samples, which amount to 50% of the frame. Hence, given the 50% analysis frame overlap, there is no overlap between residual segments.

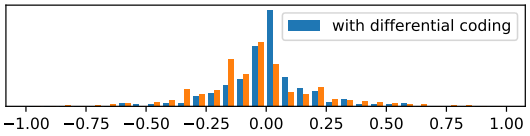

Fig. 6. Differential coding enables a more centralized distribution

3.2. Residual coding

The LPC residual, calculated from the trainable LPC analyzer (Fig. 4), is compressed by the 1D-CNN AEs as described in Section 2.3. In this work, we employ differential coding [25] to the output of encoders, $\boldsymbol{h} = [h_0, h_1, \cdots, h_{m-1}]$ where m is the length of code per frame for each AE. Hence, the input scalar to the softmax quantization is $\Delta h_i = h_i - h_{i-1}$. Consequently, the quantization starts from a more centralized real-valued "code" distribution (Fig.6). As illustrated in Fig. 5, both the quantization of LPC coefficients and residual coding with CMRL are optimized together. With this design, the purpose of LPC analysis is not just to minimize the residual signal energy as much as possible [26], but to find a pivot which also facilitates the residual compression from following CMRL modules.

3.3. Model training

According to the CMRL pipeline, the individual AEs can be trained sequentially by using the residual of the previous module as the input of the AE and the target of prediction. Once all the AEs are trained, finetuning step follows to improve the total reconstruction quality. This section discusses the loss function we used for training each of the AEs as well as for finetuning. The loss function consists of the reconstruction error terms and regularizers:

$$\mathcal{L} = \lambda_1 \mathcal{T}(y||\hat{y}) + \lambda_2 \mathcal{F}(y||\hat{y}) + \lambda_3 \mathcal{Q}(\boldsymbol{A}_{\text{soft}}) + \lambda_4 \mathcal{E}(\boldsymbol{A}_{\text{soft}}) \quad (2)$$

Given that the input of CQ is in time domain, we minimize the loss in both time and frequency domains. The time domain error, $\mathcal{T}(y||\hat{y})$, is measured by mean squared error (MSE). $\mathcal{F}(y||\hat{y})$ compensates what is not captured by the non-perceptual $\mathcal{T}(y||\hat{y})$ term by measuring the loss in mel-scale frequency domain. Four different mel-filter banks are specified with the size of 128, 32, 16 and 8, to enable a coarse-to-fine differentiation.

$\mathcal{Q}(\boldsymbol{A}_{\text{soft}})$ and $\mathcal{E}(\boldsymbol{A}_{\text{soft}})$ are regularizers for softmax quantization. For the soft assignment matrix $\boldsymbol{A}_{\text{soft}}$ as defined in Section 2.2, $\mathcal{Q}(\boldsymbol{A}_{\text{soft}})$ is defined as $\sum_{i,j}(\sqrt{\boldsymbol{A}_{\text{soft}}(i,j)} - 1)/I$, to