INTERSPEECH 2019
September 15–19, 2019, Graz, Austria

Cascaded Cross-Module Residual Learning towards Lightweight End-to-End Speech Coding

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

[1]Indiana University, 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, jmseong@etri.re.kr, lms@etri.re.kr, skbeack@etri.re.kr, minje@indiana.edu

Abstract

Speech codecs learn compact representations of speech signals to facilitate data transmission. Many recent deep neural network (DNN) based end-to-end speech codecs achieve low bitrates and high perceptual quality at the cost of model complexity. We propose a cross-module residual learning (CMRL) pipeline as a module carrier with each module reconstructing the residual from its preceding modules. CMRL differs from other DNN-based speech codecs, in that rather than modeling speech compression problem in a single large neural network, it optimizes a series of less-complicated modules in a two-phase training scheme. The proposed method shows better objective performance than AMR-WB and the state-of-the-art DNN-based speech codec with a similar network architecture. As an end-to-end model, it takes raw PCM signals as an input, but is also compatible with linear predictive coding (LPC), showing better subjective quality at high bitrates than AMR-WB and OPUS. The gain is achieved by using only 0.9 million trainable parameters, a significantly less complex architecture than the other DNN-based codecs in the literature.

Index Terms: speech coding, deep neural network, entropy coding, residual learning

1. Introduction

Speech coding, where the encoder converts the speech signal into bitstreams and the decoder synthesizes reconstructed signal from received bitstreams, serves an important role for various purposes: to secure a voice communication [1][2], to facilitate data transmission [3], etc. There have been various conventional speech coding methodologies, including linear predictive coding (LPC) [4], adaptive encoding [5], and perceptual weighting [6] among other domain specific knowledge about the speech signals, that are used to construct classic codecs, such as AMR-WB [7] and OPUS [8] with high perceptual quality.

Since the last decade, data-driven approaches have vitalized the use of deep neural networks (DNN) for speech coding. A speech coding system can be formulated by DNN as an autoencoder (AE) with a code layer discretized by vector quantization (VQ) [9] or bitwise network techniques [10], etc. Many DNN methods [11][12] take inputs in time-frequency (T-F) domain from short time Fourier transform (STFT) or modified discrete cosine transform (MDCT), etc. Recent DNN-based codecs [13][14][15][16] model speech signals in time domain directly without T-F transformation. They are referred to as end-to-end methods, yielding competitive performance comparing with current speech coding standards, such as AMR-WB [7].

While DNN serves a powerful parameter estimation paradigm, they are computationally expensive to run on smart devices. Many DNN-based codecs achieve both low bitrates and high perceptual quality, two main targets for speech codecs [17][18][19], but with a high model complexity. A WaveNet based variational autoencoder (VAE) [16] outperforms other low bitrate codecs in the listening test, however, with 20 millions parameters, a too big model for real-time processing in a resource-constrained device. Similarly, codecs built on SampleRNN [20][21] can also be energy-intensive.

Motivated by DNN based end-to-end codecs [14] and residual cascading [22][23], this paper proposes a "cross-module" residual learning (CMRL) pipeline, which can lower the model complexity while maintaining a high perceptual quality and compression ratio. CMRL hosts a list of less-complicated end-to-end speech coding modules. Each module learns to recover what is failed to be reconstructed by its preceding modules. CMRL differs from other residual learning networks, e.g. ResNet [24], in that rather than adding identical shortcuts between layers, CMRL cascades residuals across a series of DNN modules. We introduce a two-round model training scheme to train CMRL models. In addition, we also show that CMRL is compatible with LPC by having it as one of the modules. With LPC coefficients being predicted, CMRL recovers the LPC residuals which, along with the LPC coefficients, synthesize the decoded speech signal at the receiver side.

The evaluation of the propose method is threefold: objective measures, subjective assessment and model complexity. Comparing with AMR-WB, OPUS, and the recently proposed end-to-end system [14], CMRL showed promising performance both in objective and subjective quality assessments. As for complexity, CMRL contains only 0.9 million model parameters, significantly less complicated than the WaveNet based speech codec [16] and the end-to-end baseline [14].

2. Model description

Before introducing CMRL as a module carrier, we describe the component module to be hosted by CMRL.

2.1. The component module

Recently, an end-to-end DNN speech codec (referred to as Kankanahalli-Net) has shown competitive performance comparable to one of the standards (AMR-WB) [14]. We describe our component model derived from Kankanahalli-Net that consists of bottleneck residual learning [24], soft-to-hard quantization [25], and sub-pixel convolutional neural networks for upsampling [26]. Figure 1 depicts the component module.

http://dx.doi.org/10.21437/Interspeech.2019-1816

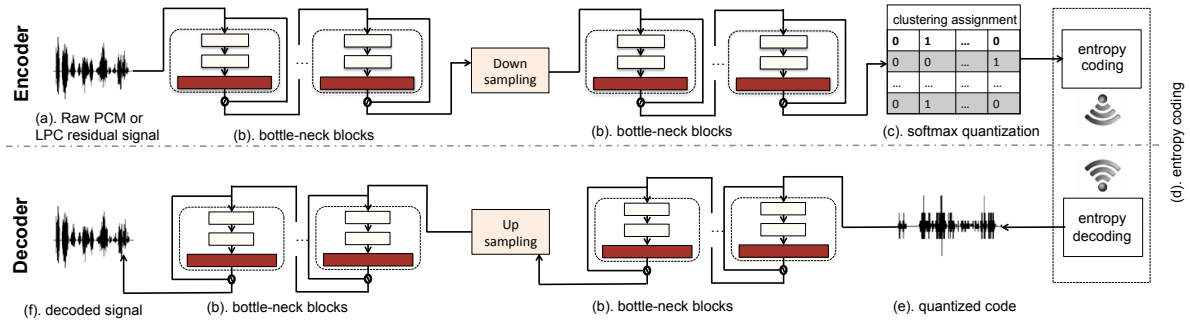

Figure 1: *A schematic diagram for the end-to-end speech coding component module: some channel change steps are omitted.*

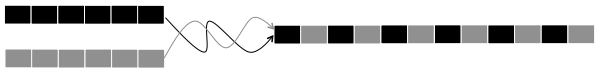

Figure 2: *The interlacing-based upsampling process.*

2.1.1. Four non-linear mapping types

In the end-to-end speech codec, we take $S = 512$ time domain samples per frame, 32 of which are windowed by the either left or right half of a Hann window and then overlapped with the adjacent ones. This forms the input to the first 1-D convolutional layer of C kernels, whose output is a tensor of size $S \times C$.

There are four types of non-linear transformations involved in this fully convolutional network: downsampling, upsampling, channel changing, and residual learning. The downsampling operation reduces S down to $S/2$ by setting the stride d of the convolutional layer to be 2, which turns an input example $S \times C$ into $S/2 \times C$. The original dimension S is recovered in the decoder with recently proposed sub-pixel convolution [25], which forms the upsampling operation. The super-pixel convolution is done by interlacing multiple feature maps to expand the size of the window (Figure 2). In our case, we interlace a pair of feature maps, and that is why in Table 1 the upsampling layer reduces the channels from 100 to 50 while recovers the original 512 dimensions from 256.

In this work, to simplify the model architecture we have identical shortcuts only for cross-layer residual learning, while Kankanahalli-Net employs them more frequently. Furthermore, inspired by recent work in source separation with dilated convolutional neural network [27], we use a "bottleneck" residual learning block to further reduce the number of parameters. This can lower the amount of parameters, because the reduced number of channels within the bottleneck residual learning block decreases the depth of the kernels. See Table 1 for the size of our kernels. Likewise, the input $S \times 1$ tensor is firstly converted to a $S \times C$ feature map, and then downsampled to $S/2 \times C$. Eventually, the code vector shrinks down to $S/2 \times 1$. The decoding process recovers it back to a signal of size $S \times 1$, reversely.

2.1.2. Softmax quantization:

The coded output from each encoder is still a real-valued vector of size $S/2$. Softmax quantization [25] performs scalar quantization by assigning each real value to the nearest representative (Figure 1 (c)). In the proposed system, softmax quantization maps the input scalar to one of the 32 clusters, or quantization levels, which requires $\log_2 32 = 5$ bits per dimension. Huffman coding further reduces the bitrate [28].

2.2. The module carrier: CMRL

Figure 3 shows the proposed cascaded cross-module residual learning (CMRL) process. In CMRL, each module does its best to reconstruct its input. The procedure in the i-th module is denoted as $\mathcal{F}(\boldsymbol{x}^{(i)}; \mathbb{W}^{(i)})$, which estimates the input as $\hat{\boldsymbol{x}}^{(i)}$. The input for the i-th module is defined as

$$\boldsymbol{x}^{(i)} = \boldsymbol{x} - \sum_{j=1}^{i-1} \hat{\boldsymbol{x}}^{(j)}, \tag{1}$$

where the first module takes the input speech signal, i.e., $\boldsymbol{x}^{(1)} = \boldsymbol{x}$. The meaning is that each module learns to reconstruct the residual which is not recovered by its preceding modules. Note that module homogeneity is not required for CMRL: for example, the first module can be very shallow to just estimate the envelope of MDCT spectral structure while the following modules may need more parameters to estimate the residuals.

Each AE decomposes into the encoder and decoder parts:

$$\boldsymbol{h}^{(i)} = \mathcal{F}_{\text{enc}}(\boldsymbol{x}^{(i)}; \mathbb{W}_{\text{enc}}^{(i)}), \quad \hat{\boldsymbol{x}}^{(i)} = \mathcal{F}_{\text{dec}}(\boldsymbol{h}^{(i)}; \mathbb{W}_{\text{dec}}^{(i)}), \tag{2}$$

where $\boldsymbol{h}^{(i)}$ denotes the part of code generated by the i-th encoder, and $\mathbb{W}_{\text{enc}}^{(i)} \cup \mathbb{W}_{\text{dec}}^{(i)} = \mathbb{W}^{(i)}$.

The encoding process: For a given input signal $\boldsymbol{x}$, the encoding process runs all N AE modules in a sequential order. Then, the bistring is generated by taking the encoder outputs and concatenating them: $\boldsymbol{h} = \left[\boldsymbol{h}^{(1)\top}, \boldsymbol{h}^{(2)\top}, \cdots, \boldsymbol{h}^{(N)\top}\right]^{\top}$.

The decoding process: Once the bitstring is available on the receiver side, all the decoder parts of the modules, $\mathcal{F}_{\text{dec}}(\boldsymbol{x}^{(i)}; \mathbb{W}_{\text{dec}}^{(i)})\ \forall N$, run to produce the reconstructions which are added up to approximate the initial input signal with the global error defined as

$$\hat{\mathcal{E}}\left(\boldsymbol{x} \middle|\middle| \sum_{i=1}^{N} \hat{\boldsymbol{x}}^{(i)}\right). \tag{3}$$

2.2.1. The two-round training scheme

Intra-module greedy training: We provide a two-round training scheme to make CMRL optimization tractable. The first round adopts a *greedy* training scheme, where each AE tries its best to minimize the error: $\arg\min_{\mathbb{W}^{(i)}} \mathcal{E}(\boldsymbol{x}^{(i)} || \mathcal{F}(\boldsymbol{x}^{(i)}; \mathbb{W}^{(i)}))$. The greedy training scheme echoes a divide-and-conquer manner, leading to an easier optimization for each module. The thick gray arrows in Figure 3 show the flow of the backpropagation error to minimize the individual module error with respect to the module-specific parameter set $\mathbb{W}^{(i)}$.

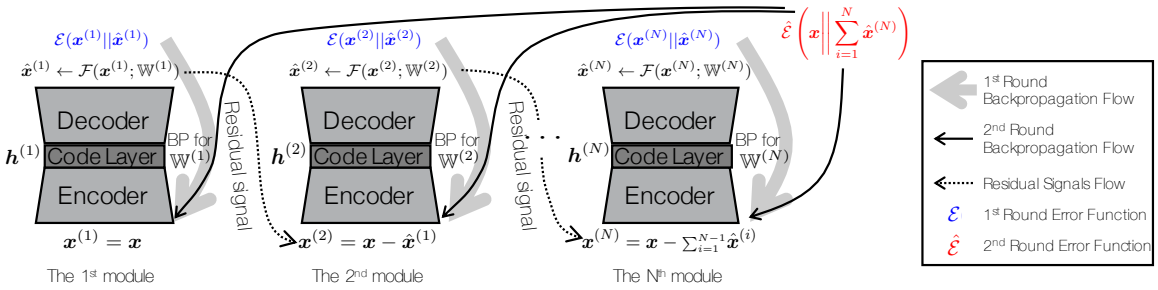

Figure 3: *Cross-module residual learning pipeline*

Cross-module finetuning: The greedy training scheme accumulates module-specific error, which the earlier modules do not have a chance to reduce, thus leading to a suboptimal result. Hence, the second-round cross-module finetuning follows to further improve the performance by reducing the total error:

$$\underset{\mathbb{W}^{(1)}\dots\mathbb{W}^{(N)}}{\arg\min}\ \hat{\mathcal{E}}\left(\boldsymbol{x}\middle\|\sum_{i=1}^{N}\mathcal{F}(\boldsymbol{x}^{(i)};\mathbb{W}^{(i)})\right). \tag{4}$$

During the finetuing step, we first (a) initialize the parameters of each module with those estimated from the greedy training step (b) perform cascaded feedforward on all the modules sequentially to calculate the total estimation error in (3) (c) backpropagate the error to update parameters in all modules altogether (thin black arrows in Figure 3). Aside from the total reconstruction error (3), we inherit Kankanahalli-Net's other regularization terms, i.e., perceptual loss, quantization penalty, and entropy regularizer.

2.3. Bitrate and entropy coding

The bitrate is calculated from the concatenated bitstrings from all modules in CMRL. Each encoder module produces S/d quantized symbols from the softmax quantization process (Figure 1 (e)), where the stride size d divides the input dimensionality. Let $c^{(i)}$ be the average bit length per symbol after Huffman coding in the i-th module. Then, $c^{(i)}S/d$ stands for the bits per frame. By dividing the frame rate, $(S-o)/f$, where o and f denote the overlap size in samples and the sampling rate, respectively, the bitrates per module add up to the total bitrate: $\xi_{\text{LPC}} + \sum_{i=1}^{N}\frac{fcS}{(S-o)d}$, where the overhead to transmit LPC coefficients is ξ_{lpc}=2.4kbps, which is 0 for the case with raw PCM signals as the input.

By having the entropy control scheme proposed in Kankanahalli-Net as the baseline to keep a specific bitrate, we further enhance the coding efficiency by employing the Huffman coding scheme on the vectors. Aside from encoding each symbol (i.e., the softmax result) separately, encoding short sequences can further leverage the temporal correlation in the series of quantized symbols, especially when the entropy is already low [29] [30]. We found that encoding a short symbol sequence of adjacent symbols, i.e., two symbols, can lower down the average bit length further in the low bitrates.

3. Experiments

We first show that for the raw PCM input CMRL outperforms AMR-WB and Kankanahalli-Net in terms of objective metrics in the experimental setup proposed in [14], where the use of LPC was not tested. Therefore, for the subjective quality, we perform MUSHRA tests [31] to show that CMRL with an LPC residual input works better than AMR-WB and OPUS at high bitrates.

Table 1: *Architecture of the component module as in Figure 1. 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)

3.1. Experimental setup

300 and 50 speakers are randomly selected from TIMIT [32] training and test datasets, respectively. We consider two types of inputs in time-domain: raw PCM and LPC residuals. For the raw PCM input, the data is normalized to have a unit variance, and then directly fed to the model. For the LPC residual input, we conduct a spectral envelope estimation on the raw signals to get LPC residuals and corresponding coefficients. The LPC residuals are modeled by the proposed end-to-end CMRL pipeline, while the LPC coefficients are quantized and sent directly to the receiver side at 2.4 kbps. The decoding process recovers the speech signal based on the LPC synthesis procedure using the LPC coefficients and the decoded residual signals.

We consider four bitrate cases: 8.85 kbps, 15.85 kbps, 19.85 kbps and 23.85 kbps. All convolutional layers in CMRL use 1-D kernel with the size of 9 and the Leaky Relu activation. CMRL hosts two modules: each module is with the topology as in Table 1. Each residual learning block contains two bottleneck structures with the dilation rate of 1 and 2. Note that for the lowest bitrate case, the second encoder downsamples each window to 128 symbols. The learning rate is 0.0001 to train the first module, and 0.00002 for the second module. Finetuning uses 0.00002 as the learning rate, too. Each window contains 512 samples with the overlap size of 32. We use Adam optimizer [33] with the batch size of 128 frames. Each module is trained for 30 epochs followed by finetuning until the entropy is