The article has been accepted for publication by IEEE. Full citation:
Kai Zhen, Mi Suk. Lee, Jongmo Sung, Seungkwon Beack and Minje Kim, "Psychoacoustic Calibration of Loss Functions for Efficient End-to-End Neural Audio Coding," in IEEE Signal Processing Letters, vol. 27, pp. 2159-2163, 2020.
DOI: 10.1109/LSP.2020.3039765.

Psychoacoustic Calibration of Loss Functions for Efficient End-to-End Neural Audio Coding

Kai Zhen, *Student Member, IEEE*, Mi Suk Lee, Jongmo Sung, Seungkwon Beack, Minje Kim, *Senior Member, IEEE*

***Abstract*—Conventional audio coding technologies commonly leverage human perception of sound, or psychoacoustics, to reduce the bitrate while preserving the perceptual quality of the decoded audio signals. For neural audio codecs, however, the objective nature of the loss function usually leads to suboptimal sound quality as well as high run-time complexity due to the large model size. In this work, we present a psychoacoustic calibration scheme to re-define the loss functions of neural audio coding systems so that it can decode signals more perceptually similar to the reference, yet with a much lower model complexity. The proposed loss function incorporates the global masking threshold, allowing the reconstruction error that corresponds to inaudible artifacts. Experimental results show that the proposed model outperforms the baseline neural codec twice as large and consuming 23.4% more bits per second. With the proposed method, a lightweight neural codec, with only 0.9 million parameters, performs near-transparent audio coding comparable with the commercial MPEG-1 Audio Layer III codec at 112 kbps.**

***Index Terms*—Audio coding, deep neural networks, psychoacoustics, network compression**

I. Introduction

AUDIO coding, a fundamental set of technologies in data storage and communication, compresses the original signal into a bitstream with a minimal bitrate (encoding) without sacrificing the perceptual quality of the recovered waveform (decoding) [1], [2]. In this paper we focus on the lossy codecs, which typically allow information loss during the process of encoding and decoding only in inaudible audio components. To this end, psychoacoustics is employed to quantify the audibility in both time and frequency domains. For example, MPEG-1 Audio Layer III (also known as MP3), as a successful commercial audio codec, achieves a near-transparent quality at 128 kbps by using a psychoacoustic model (PAM) [2]. Its bit allocation scheme determines the number of bits allocated to each subband by dynamically computing the masking threshold via a PAM and then allowing quantization error once it is under the threshold [3].

Recent efforts on deep neural network-based speech coding systems have made substantial progress on the coding gain [4]–[6]. They formulate coding as a complex learning process that converts an input to a compact hidden representation. This poses concerns for edge applications with the computational resource at a premium: a basic U-Net audio codec contains approximately 10 million parameters [7]; in [8], vector quantized variational autoencoders (VQ-VAE) [9] employs WaveNet [5] as a decoder, yielding a competitive speech quality at 1.6 kbps, but with 20 million parameters. In addition, recent neural speech synthesizers employ traditional DSP techniques, e.g., linear predictive coding (LPC), to reduce its complexity [10]. Although it can serve as a decoder of a speech codec, LPC does not generalize well to non-speech signals.

Perceptually meaningful objective functions have shown an improved trade-off between performance and efficiency. Some recent speech enhancement models successfully employed perceptually inspired objective metrics, e.g. perceptual attractors [11], energy-based weighting [12], perceptual weighting filters from speech coding [13], and global masking thresholds [14] [15], while they have not targeted audio coding and model compression. Other neural speech enhancement systems implement short-time objective intelligibility (STOI) [16] and perceptual evaluation of speech quality (PESQ) [17] as the loss [18], [19]. These metrics may benefit speech codecs, but do not faithfully correlate with subjective audio quality. Meanwhile, PAM serves as a subjectively salient quantifier for the sound quality and is pervasively used in the standard audio codecs. However, integrating the prior knowledge from PAM into optimizing neural audio codecs has not been explored.

In this paper, we present a psychoacoustic calibration scheme to improve the neural network optimization process, as an attempt towards efficient and high-fidelity neural audio coding (NAC). With the global masking threshold calculated from a well-known PAM [20], the scheme firstly conducts priority weighting making the optimization process focus more on audible coding artifacts in frequency subbands with the relatively weaker masking effect, while going easy otherwise. The scheme additionally modulates the coding artifact to ensure that it is below the global masking threshold, which is analogous to the bit allocation algorithm in MP3 [2]. This is, to our best knowledge, the first method to directly incorporate psychoacoustics to neural audio coding.

II. End-to-end neural audio coding

A. Lightweight NAC Module

Given that neural codecs can suffer from a large inference cost due to their high model complexity, one of our goals is

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

Kai Zhen is with the Department of Computer Science and Cognitive Science Program at Indiana University, Bloomington, IN 47408 USA. Mi Suk Lee, Jongmo Sung, and Seungkwon Beack are with Electronics and Telecommunications Research Institute, Daejeon, Korea 34129. Minje Kim is with the Dept. of Intelligent Systems Engineering at Indiana University (e-mails: zhenk@iu.edu, lms@etri.re.kr, jmseong@etri.re.kr, skbeack@etri.re.kr, minje@indiana.edu).

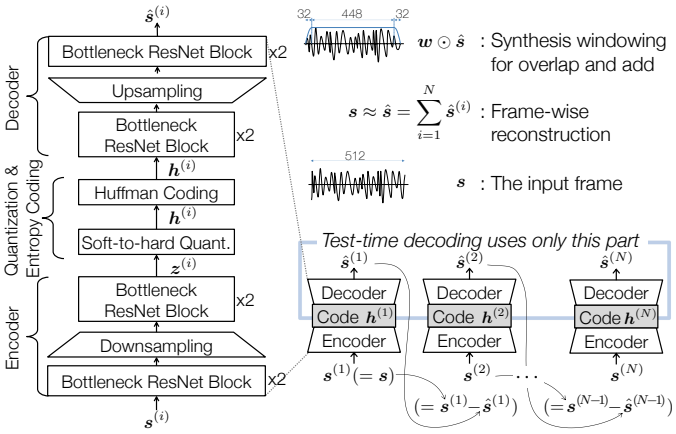


Fig. 1: Schematic diagrams for NAC. The residual coding pipeline for CMRL consists of multiple NAC autoencoding modules. Training and test-time encoding uses all blocks while the test-time decoding uses only the decoder portion.

TABLE I: The 1D-CNN NAC module architecture (Fig. 1). The shape of feature maps is (frame length, channel); the kernel shape is (kernel size, in channel, out channel).

System	Layer	Input shape	Kernel shape	Output shape
Encoder	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)
Soft-to-hard quantization & Huffman coding				
Decoder	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)

to demonstrate the advantage of the proposed pychoacoustic loss function on model compression. To that end, we choose a compact neural audio coding (NAC) module as the building block. The NAC module is a simplified version of a convolutional neural network (CNN)-based autoencoder [21] with only 450K parameters. As shown in Fig. 1, it consists of a stack of bottleneck blocks as in [22], each of which performs a ResNet-style residual coding [23]. The code vector produced by its encoder part is discretized into a bitstring via the soft-to-hard quantization process originally proposed in [24] for image compression. We detail the description as follows.

1) Encoder: The CNN encoder maps an input frame of T time-domain samples, $\boldsymbol{s} \in \mathbb{R}^T$ to the code vector, i.e., $\boldsymbol{z} \leftarrow \mathcal{F}_{\text{enc}}(\boldsymbol{s})$. Striding during the 1D convolution operation can downsample the feature map. For example, $\boldsymbol{z} \in \mathbb{R}^{T/2}$ when the stride is set to be 2 and applied once during encoding. The detailed architecture is summarized in TABLE I.

2) Soft-to-hard quantization: Quantization replaces each real-valued element of the code vector $\boldsymbol{z}$ with a kernel value chosen from a set of K representatives. We use soft-to-hard quantizer [24], a clustering algorithm compatible with neural networks, where the representatives are also trainable. During training, in each feedforward routine, the c-th code value z_c is assigned to the nearest kernel out of K, $\boldsymbol{\beta} \in \mathbb{R}^K$, which have been trained so far. The discrepancy between z_c and the chosen kernel $h_c \in \{\beta_1, \beta_2, \cdots, \beta_K\}$ (namely the quantization error) is accumulated in the final loss, and then reduced during training via backpropagation (i.e., by updating the means and assignments). Specifically, the cluster assignment is conducted by calculating the distance, $\boldsymbol{d} \in \mathbb{R}^K$, between the code value and all kernels, and then applying the softmax function to the negatively scaled distance to produce a probabilistic membership assignment: $\boldsymbol{a} \leftarrow \text{softmax}(-\alpha \boldsymbol{d})$. Although we eventually need a hard assignment vector $\mathbf{a}$, i.e., a one-hot vector that indicates the closest kernel, during training the quantized code $\boldsymbol{h}$ is acquired by a soft assignment, $\boldsymbol{a}^\top \boldsymbol{\beta}$, for differentiability. Hence, at the test time, $\mathbf{a}$ replaces $\boldsymbol{a}$ by turning on only the maximum element. Note that a larger scaling factor α makes $\boldsymbol{a}$ harder, making it more similar to $\mathbf{a}$. Huffman coding follows to generate the final bitstream.

3) Decoder: The decoder recovers the original signal from the quantized code vector: $\hat{\boldsymbol{s}} = \mathcal{F}_{\text{dec}}(\boldsymbol{h})$, by using an architecture mirroring that of the encoder (TABLE I). For upsampling, we use a sub-pixel convolutional layer proposed in [25] to recover the original frame length T.

4) Bitrate Analysis and Control: The lower bound of the bitrate is defined as $|\boldsymbol{h}|\mathcal{H}(\boldsymbol{h})$, where $|\boldsymbol{h}|$ is the number of down-sampled and quantized features per second. The entropy $\mathcal{H}(\boldsymbol{h})$ forms the lower bound of the average amount of bits per feature. While $|\boldsymbol{h}|$ is a constant given a fixed sampling rate and network topology, $\mathcal{H}(\boldsymbol{h})$ is adaptable during training. As detailed in [24], basic information theory calculates $\mathcal{H}(\boldsymbol{h})$ as $-\sum_k p(\beta_k) \log_2 p(\beta_k)$, where $p(\beta_k)$ denotes the occurrence probability of the k-th cluster defined in the soft-to-hard quantization. Therefore, during model training, $\mathcal{H}(\boldsymbol{h})$ is added to the loss function as a regularizer navigating the model towards the target bitrate. Initiated as 0.0, the blending weight increases by 0.015 if the actual bitrate overshoots the target and decreases by that amount otherwise. Because this regularizer is well defined in the literature [24] [21] [26], we omit it in following sections for simplicity purposes.

B. Cross-Module Residual Learning

To scale up for high bitrates, cross-module residual learning (CMRL) [26] implants the multistage quantization scheme [27] by cascading residual coding blocks (Fig. 1). CMRL decentralizes the neural autoencoding effort to a chain of serialized low complexity coding modules, with the input of i-th module being $\boldsymbol{s}^{(i)} = \boldsymbol{s} - \sum_{j=1}^{i-1} \hat{\boldsymbol{s}}^{(j)}$. That said, each module only encodes what is not reconstructed from preceding modules, making the system scalable. Concretely, for an input signal $\boldsymbol{s}$, the encoding process runs all N autoencoder modules in a sequential order, which yields the bitstring as a concatenation of the quantized code vectors: $\boldsymbol{h} = \left[\boldsymbol{h}^{(1)\top}, \boldsymbol{h}^{(2)\top}, \cdots, \boldsymbol{h}^{(N)\top}\right]^\top$. During decoding, all de-

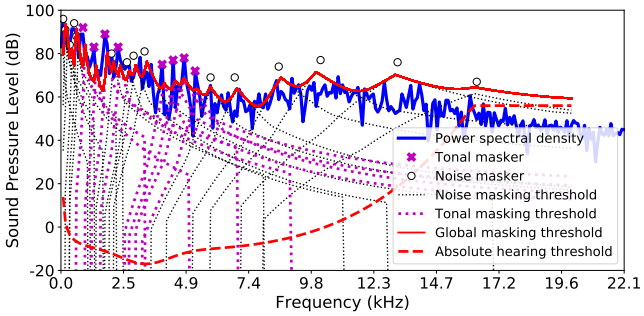


Fig. 2: Visualization of the masker detection, individual and global masking threshold calculation for an audio input.

coders, $\mathcal{F}_{\text{dec}}(\boldsymbol{h}^{(i)})\ \ \forall i$, run to produce the reconstructions that sum up to approximate the initial input signal as $\sum_{i=1}^{N}\hat{\boldsymbol{s}}^{(i)}$.

III. The Proposed Psychoacoustic Calibration

The baseline model uses the sum of squared error (SSE) defined in the time domain: $\mathcal{L}_1(\boldsymbol{s}||\hat{\boldsymbol{s}}) = \sum_{i=1}^{N}\sum_{t=1}^{T}\left(\hat{s}_t^{(i)} - s_t^{(i)}\right)^2$. In addition, another loss is defined in the mel-scaled frequency domain to weigh more on the low frequency area, as the human auditory system does, $\mathcal{L}_2(\boldsymbol{y}||\hat{\boldsymbol{y}}) = \sum_{i=1}^{N}\sum_{l=1}^{L}\left(y_l^{(i)} - \hat{y}_l^{(i)}\right)^2$, where $\boldsymbol{y}$ stands for a mel spectrum with L frequency subbands as proposed in [21].

A. Psychoacoustic Model-1

Without loss of generality, we choose a basic PAM that computes simultatenous masking effects for the input signal as a function of frequency, while the temporal masking effect is not integrated. According to PAM-1 defined in [2], for an input frame, it (a) calculates the logarithmic power spectral density (PSD) $\boldsymbol{p}$; (b) detects tonal and noise maskers, followed by decimation; (c) calculates masking threshold for individual tonal and noise maskers $\boldsymbol{U} \in \mathbb{R}^{F\times R}, \boldsymbol{V} \in \mathbb{R}^{F\times B}$, where R and B are the number of maskers. The global masking threshold at frequency bin f is accumulated from each individual masker in (c) along with the absolute hearing threshold $\boldsymbol{Q}$ [20], as $m_f = 10\log_{10}\left(10^{0.1Q_f} + \sum_r 10^{0.1U_{f,r}} + \sum_b 10^{0.1V_{f,b}}\right)$. Fig. 2 shows an example of $\boldsymbol{p}$ of a signal and its global masking threshold based on the simultaneous masking effect.

Global masking threshold as discussed is used in various conventional audio codecs to allocate minimal amount of bits without losing the perceptual audio quality. Typically, the bit allocation algorithm optimizes n_f/m_f (NMR), where n_f denotes the power of the noise (i.e., coding artifacts) in the subband f and m_f is the power of the global masking threshold. In an iterative process, each time the bit is assigned to the subband with the highest NMR until no more bit can be allocated [28]–[30]. The global masking curve acquired via PAM-1 comprises both input-invariant prior knowledge as in the absolute hearing threshold and input-dependent masking effects. We propose two mechanisms to integrate PAM-1 into NAC optimization: priority weighting and noise modulation.

B. Priority Weighting

During training we estimate the logarithmic PSD $\boldsymbol{p}$ out of an input frame $\boldsymbol{s}$, as well the global masking threshold $\boldsymbol{m}$ to define a perceptual weight vector, $\boldsymbol{w} = \log_{10}(\frac{10^{0.1\boldsymbol{p}}}{10^{0.1\boldsymbol{m}}} + 1)$: the log ratio between the signal power and the masked threshold, rescaled from decibel. Accordingly, we define a weighting scheme that pays more attention to the unmasked frequencies:

$$\mathcal{L}_3(\boldsymbol{s}||\hat{\boldsymbol{s}}) = \sum_i \sum_f w_f \left(x_f^{(i)} - \hat{x}_f^{(i)}\right)^2, \tag{1}$$

where $x_f^{(i)}$ and $\hat{x}_f^{(i)}$ are the f-th magnitude of the Fourier spectra of the input and the recovered signals for the i-th CMRL module. The intuition is that, if the signal's power is greater than its masking threshold at the f-th frequency bin, i.e. $p_f > m_f$, the model tries hard to recover this audible tone precisely: a large w_f enforces it. Otherwise, for a masked tone, the model is allowed to generate some reconstruction error. The weights are bounded between 0 and ∞, whose smaller extreme happens if, for example, the masking threshold is too large comparing to the sufficiently soft signal.

C. Noise Modulation

The priority weighting mechanism can accidentally result in audible reconstruction noise, exceeding the mask value m_f, when w_f is small. Our second psychoacoustic loss term is to modulate the reconstruction noise by directly exploiting NMR, n_f/m_f, where $\boldsymbol{n}$ is the power spectrum of the reconstruction error $\boldsymbol{s} - \sum_{i=1}^{N}\hat{\boldsymbol{s}}^{(i)}$ from all N autoencoding modules. We tweak the greedy bit allocation process in the MP3 encoder that minimizes NMR iteratively, such that it is compatible to the stochastic gradient descent algorithm as follows:

$$\mathcal{L}_4 = \max_f \left(\text{ReLU}\left(\frac{n_f}{m_f} - 1\right)\right). \tag{2}$$

The rectified linear units (ReLU) function excludes the contribution of the inaudible noise to the loss when $n_f/m_f - 1 < 0$. Out of those frequency bins where the noise is audible, the max operator selects the one with the largest NMR, which counts towards the total loss. The process as such resembles MP3's bit allocation algorithm, as it tackles the frequency bin with the largest NMR for each training iteration.

IV. Experiments

A. Experimental Setup

1) Data Preparation and Hyperparameters: Our training dataset consists of 1,000 single-channel clips of commercial music, spanning 13 genres. Each clip is about 20 seconds long, amounting to about 5.5 hours of play time. The sampling rate is 44.1 kHz and downsampled to 32 kHz for the lower bitrate setup. Each frame contains $T = 512$ samples with an overlap of 32 samples, where a Hann window is applied to the overlapping region. Note that the choice of frame size is to align the system's hyperparameters to the previous work [21], [26], [31], but it does not necessarily mean that 512 results in an enough frequency resolution for PAM-based lost terms. For training, hyperparameters are found based on validation with another 104 clips: 128 frames for the batch size; $\alpha = 300$ for the initial softmax scaling factor; 2×10^{-4} for the initial learning rate of the Adam optimizer [32], and 2×10^{-5}

Scalable and Efficient Neural Speech Coding: A Hybrid Design

Kai Zhen, *Student Member, IEEE*, Jongmo Sung, Mi Suk Lee, Seungkwon Beack, and Minje Kim, *Senior Member, IEEE*

***Abstract*—We present a scalable and efficient neural waveform coding system for speech compression. We formulate the speech coding problem as an autoencoding task, where a convolutional neural network (CNN) performs encoding and decoding as a neural waveform codec (NWC) during its feedforward routine. The proposed NWC also defines quantization and entropy coding as a trainable module, so the coding artifacts and bitrate control are handled during the optimization process. We achieve efficiency by introducing compact model components to NWC, such as gated residual networks and depthwise separable convolution. Furthermore, the proposed models are with a scalable architecture, cross-module residual learning (CMRL), to cover a wide range of bitrates. To this end, we employ the residual coding concept to concatenate multiple NWC autoencoding modules, where each NWC module performs residual coding to restore any reconstruction loss that its preceding modules have created. CMRL can scale down to cover lower bitrates as well, for which it employs linear predictive coding (LPC) module as its first autoencoder. The hybrid design integrates LPC and NWC by redefining LPC's quantization as a differentiable process, making the system training an end-to-end manner. The decoder of proposed system is with either one NWC (0.12 million parameters) in low to medium bitrate ranges (12 to 20 kbps) or two NWCs in the high bitrate (32 kbps). Although the decoding complexity is not yet as low as that of conventional speech codecs, it is significantly reduced from that of other neural speech coders, such as a WaveNet-based vocoder. For wide-band speech coding quality, our system yields comparable or superior performance to AMR-WB and Opus on TIMIT test utterances at low and medium bitrates. The proposed system can scale up to higher bitrates to achieve near transparent performance.**

***Index Terms*—Neural speech coding, waveform coding, representation learning, model complexity.**

Manuscript received March 26, 2021; revised June 28, 2021 and September 9, 2021; accepted November 8, 2021. Date of publication November 19, 2021; date of current version December 20, 2021. This work was supported by the Institute for Information and Communications Technology Promotion (IITP) funded by the Korea government (MSIT) under Grant 2017-0-00072 (Development of Audio/Video Coding, and Light Field Media Fundamental Technologies for Ultra Realistic Tera-Media). The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Tom Backstrom. *(Corresponding author: Minje Kim.)*

Kai Zhen is with the Department of Computer Science and Cognitive Science Program, Indiana University, Bloomington, IN 47408 USA (e-mail: zhenk@iu.edu).

Jongmo Sung, Mi Suk Lee, and Seungkwon Beack are with the Electronics and Telecommunications Research Institute, Daejeon 34129, Korea (e-mail: jmseong@etri.re.kr; lms@etri.re.kr; skbeack@etri.re.kr).

Minje Kim is with the Department of Intelligent Systems Engineering, Indiana University, Bloomington, IN 47408 USA (e-mail: minje@indiana.edu).

Digital Object Identifier 10.1109/TASLP.2021.3129353

I. Introduction

SPEECH coding can be implemented as an encoder-decoder system, whose goal is to compress input speech signals into the compact bitstream (encoder) and then to reconstruct the original speech from the code with the least possible quality degradation. Speech coding facilitates telecommunication and saves data storage among many other applications. There is a typical trade-off a speech codec must handle: the more the system reduces the amount of bits per second (bitrate), the worse the perceptual similarity between the original and recovered signals is likely to be perceived. In addition, the speech coding systems are often required to maintain an affordable computational complexity when the hardware resource is at a premium. For decades, speech coding has been intensively studied yielding various standardized codecs that can be categorized into two types: the vocoders and waveform codecs. A vocoder, also referred to as parametric speech coding, distills a set of physiologically salient features, such as the spectral envelope (equivalent to vocal tract responses including the contribution from mouth shape, tongue position and nasal cavity), fundamental frequencies, and gain (voicing level), from which the decoder *synthesizes* the speech. Typically, a vocoder operates at 3 kbps or below with high computational efficiency, but the synthesized speech quality is usually limited and does not scale up to higher bitrates [1]–[3]. On the other hand, a waveform codec aims to accurately reconstruct the input speech signal, which features up-to-transparent quality in a high bitrate range [4]. AMR-WB [5], for instance, can be seen as a hybrid waveform codec, because it employs speech modeling as in many other waveform codecs [6]–[8]. EVS [9], a recently standardized 3GPP voice and audio codec, has noticeably optimized frame error robustness, yielding a much-enhanced frame error concealment performance against than AMR-WB [10]. Similar to EVS, Opus, a waveform codec at its core, can also be applied to both speech and audio signals where it uses the LPC-based SILK algorithm for the speech-oriented model [11] and scales up to 510 kbps for transparent audio streaming and archiving.

Under the notion of unsupervised speech representation learning, deep neural network (DNN)-based codecs have revitalized the speech coding problem and provided different perspectives [12], [13]. The major motivation of employing neural networks to speech coding is twofold: to fill the performance gap between vocoders and waveform codecs towards a near-transparent speech synthesis quality; to use its trainable encoder

and learn latent representations which may benefit other DNN-implemented downstream applications, such as speech enhancement [14], [15], speaker identification [16] and automatic speech recognition [17], [18]. Having that, a neural codec can serve as a trainable acoustic unit integrated in future digital signal processing engines [13].

Recently proposed neural speech codecs have achieved high coding gain and reasonable quality by employing deep autoregressive models. The superior speech synthesis performance achieved in WaveNet-based models [19] has successfully transferred to neural speech coding systems, such as in [20], where WaveNet serves as a decoder synthesizing wideband speech samples from a conventional non-trainable encoder at 2.4 kbps. Although its reconstruction quality is comparable to waveform codecs at higher bitrates, the computational cost is significant due to the model size of over 20 million parameters.

Meanwhile, VQ-VAE [12] integrates a trainable vector quantization scheme into the variational autoencoder (VAE) [21] for discrete speech representation learning. While the bitrate can be lowered by reducing the sampling rate 64 times, the downside for VQ-VAE is that the prosody can be significantly altered. Although [22] provides a scheme to pass the pitch and timing information to the decoder as a remedy, it does not generalize to non-speech signals. More importantly, VQ-VAE as a vocoder does not address the complexity issue since it uses WaveNet as the decoder. Although these neural speech synthesis systems noticeably improve the speech quality at low bitrates, they are not feasible for real-time speech coding on the hardware with limited memory and bandwidth.

LPCNet [23] focuses on efficient neural speech coding via a WaveRNN [24] decoder by leveraging the traditional linear predictive coding (LPC) techniques. The input of the LPCNet is formed by 20 parameters (18 Bark scaled cepstral coefficients and 2 additional parameters for the pitch information) for every 10 ms frame. All these parameters are extracted from the non-trainable encoder, and vector-quantized with a fixed codebook. As discussed previously, since LPCNet functions as a vocoder, the decoded speech quality is not considered transparent [1].

In this paper, we propose a novel neural speech coding system, with a lightweight design and scalable performance. First, we design a generic neural waveform codec with only 0.35 million parameters where 0.12 million parameters belong to the decoder. Compared to our previous models in [25], [26] where the decoder has 0.23 million parameters, the current neural codec employs gated linear units to boost the gradient flow during model training and depthwise separable convolution to achieve further efficiency during decoding, as detailed in Section II. Based on this neural codec, our full system features two mechanisms to integrate speech production theory and residual coding techniques in Section III. Benefited from the residual-excited linear prediction (RELP) [27], we conduct LPC and apply the neural waveform codec to the excitation signal, which is illustrated in Section III-A. In this integration, a trainable quantizer bridges the encoding of linear spectral pairs and the corresponding LPC residual, making the speech coding pipeline end-to-end trainable. We also enable residual coding among neural waveform codecs to scale up the performance for high bitrates (Section III-B). In summary, the proposed system has following characteristics:

- *Scalability:* Similar to LPCNet [23], the proposed system is compatible with conventional spectral envelope estimation techniques. However, ours operates at a much wider bitrate range with comparable or superior speech quality to standardized waveform codecs.
- *Compactness:* The neural waveform codec in our system is with a much lower complexity than WaveNet [19] and VQ-VAE [12] based codecs. Our decoder contains only 0.12 million parameters which is $160\times$ more compact than a WaveNet counterpart. Our TensorFlow implementation's execution time to encode and decode a signal is only 42.44% of its duration on a single-core CPU in the low-to-medium bitrates and 80.21% in the high bitrate, facilitating real-time communications.
- *Trainability:* Our method is with a trainable encoder as in VQ-VAE, which can be integrated into other DNNs for acoustic signal processing. Besides, it is not constrained to speech, and can be generalized to audio coding with minimal effort as shown in [28].

TABLE I
CATEGORICAL SUMMARY OF RECENTLY PROPOSED NEURAL SPEECH CODING SYSTEMS. ✔ MEANS THE SYSTEM SUPPORTS THE FEATURE WHILE ✗ DOES NOT. ● MEANS IT IS NOT KNOWN

	WaveNet [20]	VQ-VAE [22]	LPCNet [23]	Proposed
Transparent coding	✔	●	✗	✔
Less than 1M parameters	✗	✗	✔	✔
Real-time communications	✗	✗	✔	✔
Encoder trainable	✔	✔	✗	✔

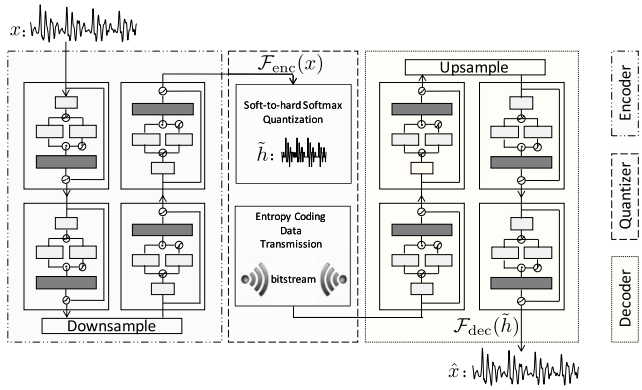
Table I highlights the comparison to the other existing neural speech codecs.

This paper is an extension of the authors' previous conference presentations [25], [26], where some initial ideas were already discussed. The new contributions presented this journal version are listed as follows:

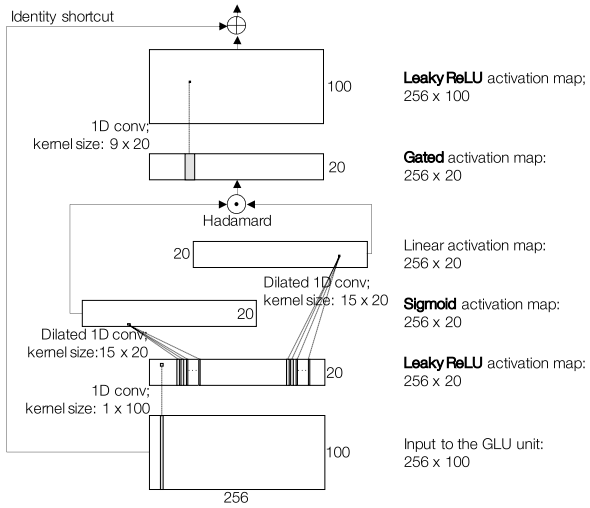
- *Novel Algorithmic Enhancements:* We propose new neural network architectures to form a new baseline autoencoder module and used it everywhere in our codecs. In our previous works, we have used a 1D convolutional neural network (CNN) that defines an autoencoder block with an identity shortcut as in the ResNet architecture [29]. While this architecture has been effective, in this journal paper, we propose to use the dilated gated linear units and depthwise separable convolution to reduce the kernel size without inducing any performance degradation. Consequently, our NWC is defined by 0.35 M parameters, whose decoder part accounts for only 0.12 M parameters. Compared to our previous models that are already small with only 0.45 M parameters, the newly introduced reduction amounts to 22.2%. If we only compare the decoder parts, it is a 47.8% reduction. Although the proposed architecture is more compact than our previous models or the WaveNet-based codecs, since neural codecs' complexity is much larger than the traditional speech codecs, the additional model

complexity reduction with no degradation of performance is promising. The architectural improvement are presented in Section II-A.

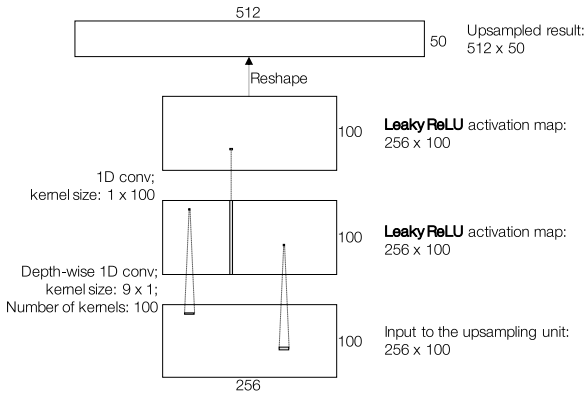
- *Extensive Experimental Validation:* In our previous works, the experimental validation was to prove the initial concepts individually proposed each paper. In this time, we conduct an extensive and thorough experiments to provide the readers with a full view to the whole building-blocks of the neural speech coding. To this end, we define four candidate systems, from Model-I to IV, by incrementally adding new modules, such as LPC, the trainable LPC quantizer, and multiple concatenated neural autoencoders. The objective and subjective tests validate each of these additions in a full view (Table III and Fig. 7).
- *Additional Analyses and Ablation Tests:* We also provide detailed experimental validation for most of the claims made in the paper by designing and performing separate experiments, which were missing in the previous papers.
  - Section IV-D1 provides experimental verification that the proposed compact neural architecture does not induce performance loss.
  - Section IV-D2 presents a detailed analysis of the behavior of the cascaded autoendoers and the impact of different training strategies.
  - Section IV-F1 explores contribution of different loss terms in our training objective by performing ablation tests, and then proposes an optimal combination of hyperparameters.
  - Section IV-F2 also conducts an ablation test to empirically verify that the proposed trainable LPC quantization algorithm improves speech quality at the same bitrate.
  - Section IV-F3 and IV-F4 analyze the bit allocation behavior among the different submodules. Since the bit allocation strategy is decided by the learning algorithm, these analyses provide evidence that our models dynamically adapt to the characteristics of the signals given the limited bit budget.
  - Section IV-G presents additional analyses on computational complexity and execution time ratios to discuss the potential of the neural codecs in real-time applications.
  - Last but not least, in Section IV-G3 we discuss the implementation issues and the limitations of the proposed system in the context of real-world application scenarios.



(a) The high-level structure of proposed neural waveform codec



(b) Dilated gated linear unit (GLU)



(c) Depthwise separable 1D convolution for upsampling

Fig. 1. The proposed architecture for lightweight NWC.

II. End-to-End Neural Waveform Codec (NWC)

The neural waveform codec (NWC), is an end-to-end autoencoder that forms the base of our proposed coding systems. NWC directly encodes the input waveform $\boldsymbol{x} \in \mathbb{R}^T$ using a convolutional neural network (CNN) encoder $\mathcal{F}_{\text{enc}}(\cdot)$, i.e., $\boldsymbol{h} \leftarrow \mathcal{F}_{\text{enc}}(\boldsymbol{x})$. Then, the quantization process $\mathcal{Q}(\cdot)$ converts the encoding into a bitstring $\tilde{\boldsymbol{h}} \in \mathbb{R}^N$, which is followed by lossless data compression and bitstream transmission. On the receiver side, the decoder reconstructs the waveform as $\boldsymbol{x} \approx \hat{\boldsymbol{x}} \leftarrow \mathcal{F}_{\text{dec}}(\tilde{\boldsymbol{h}})$. Fig. 1(a) depicts NWC's overall system architecture. The structure is detailed in Table II. It serves as a basic component in the proposed speech coding system in Section III. In this section, we first introduce the architectural improvement that reduced our model's complexity. Next, we also introduce two strategies

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

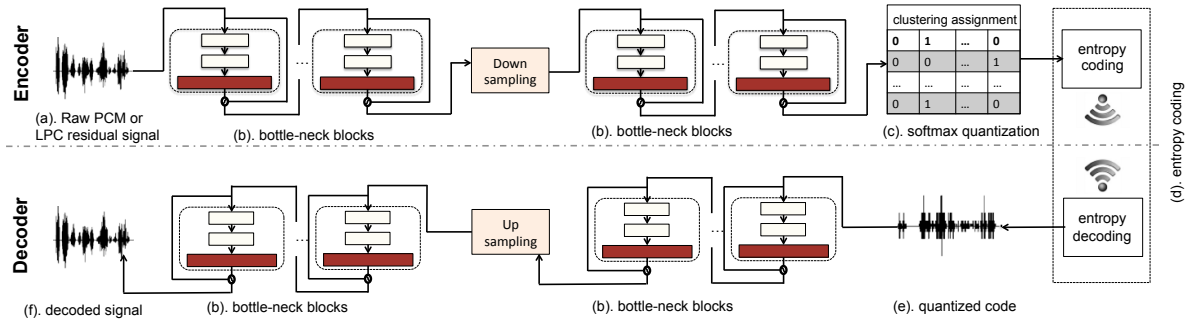


Figure 1: *A schematic diagram for the end-to-end speech coding component module: some channel change steps are omitted.*

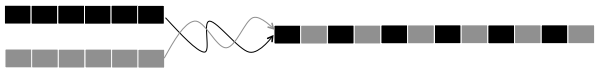


Figure 2: *The interlacing-based upsampling process.*

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 $\mathbb{W}^{(i)}$.

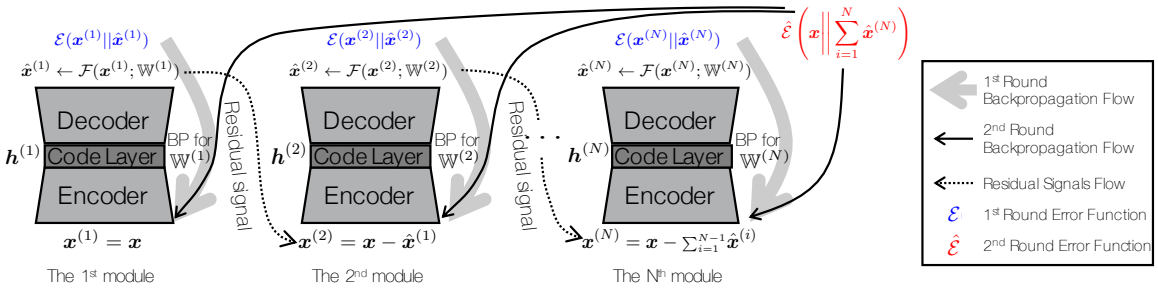


Figure 3: *Cross-module residual learning pipeline*

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

Interspeech 2022
18-22 September 2022, Incheon, Korea

Sub-8-Bit Quantization Aware Training for 8-Bit Neural Network Accelerator with On-Device Speech Recognition

Kai Zhen[†] *Hieu Duy Nguyen*[†] *Raviteja Chinta*[⋆]
Nathan Susanj[†] *Athanasios Mouchtaris*[†] *Tariq Afzal*[⋆] *Ariya Rastrow*[†]

Alexa AI, Amazon, USA[†]
Hardware Compute Group, Amazon, USA[⋆]
{zhenk, hieng, ravitec}@amazon.com
{nsusanj, mouchta, tafzal, arastrow}@amazon.com

Abstract

We present a novel sub-8-bit quantization-aware training (S8BQAT) scheme for 8-bit neural network accelerators. Our method is inspired from Lloyd-Max compression theory with practical adaptations for a feasible computational overhead during training. With the quantization centroids derived from a 32-bit baseline, we augment training loss with a Multi-Regional Absolute Cosine (MRACos) regularizer that aggregates weights towards their nearest centroid, effectively acting as a pseudo compressor. Additionally, a periodically invoked hard compressor is introduced to improve the convergence rate by emulating runtime model weight quantization. We apply S8BQAT on speech recognition tasks using Recurrent Neural Network-Transducer (RNN-T) architecture. With S8BQAT, we are able to increase the model parameter size to reduce the word error rate by 4-16% relatively, while still improving latency by 5%.
Index Terms: on-device speech recognition, sub-8-bit quantization, INT8 neural network accelerator, Lloyd-Max quantizer

1. Introduction

Latency reduction without introducing noticeable accuracy degradation is critical to on-device automatic speech recognition (ASR) systems in various scenarios, such as in in-car units and on portable devices, where Internet connectivity can be intermittent. Ubiquitous as end-to-end ASR models are [1, 2, 3, 4], deploying them on edge can be challenging due to the strict limit of bandwidth and memory of edge devices [5, 6]. Hence, it is highly necessary to improve model efficiency, i.e. model memory size vs. model accuracy, through applying model optimization and compression techniques when running real-time ASR on-device.

Efficient neural network inference can be approached by various ways. The teacher-student training paradigm for knowledge distillation simplifies the network topology, although the student may fail to mimic the teacher's behavior if the model capacity is too low [7]. Furthermore, it usually requires additional network compression techniques prior to the hardware deployment [8]. Alternatively, enforcing model sparsity can significantly reduce memory footprint [9, 10]. However, one can only realize the inference speedup if the sparsity pattern matches the specific memory design of the hardware [11].

Neural network quantization can be effectively employed to compress 32-bit weights down to 8-bit, or activations to 5-bit [12, 13], via applying a simple post-training quantization step [14, 15] or a more involved QAT mechanism [16, 17]. Several quantization methods have been proposed to lower the bit-depth to 4 [18, 19, 20], or even 1 for proof-of-concept bit-wise neural networks [21]. Nevertheless, these methods can be heuristic which rely on an extensive hyper-parameter tuning to maintain the accuracy level. Furthermore, such sub-8-bit quantization methods require sub-8-bit operators on neural network accelerators (NNAs), which often have inferior performance compared to their 8-bit counterpart due to the reduced numerical accuracy. Consequently, sub-8-bit NNAs are less adopted and thus there is no real latency measurement for existing sub-8-bit approaches [16, 18, 20, 21]. The most prevalent type of NNAs are rather based on 8-bit arithmetic operators, i.e. addition/multiplication/etc, accepting 8-bit inputs and computing the outputs via bitwise operations. For the rest of the paper, we will refer to this type as INT8 NNA.

In this work, we propose a novel sub-8-bit quantization aware training (S8BQAT) that integrates with INT8-based runtime NNAs. It differs from our previous linear-QAT method [17] in twofold. Firstly, S8BQAT distills quantization centroids from a pre-trained 32-bit baseline via a mechanism derived from Lloyd-Max scalar quantization theory. We introduce Multi-Regional Absolute Cosine (MRACos) regularizer, which is INT8 compatible and computationally efficient. The MRACos regularizer penalizes off-the-centroid weights and aggregates them towards their nearest quantization centroids. Additionally, the MRACos regularizer is accompanied by a periodic compressor that assigns each model weight to that nearest quantization centroid, ensuring quantization convergence and therefore minimizing runtime quantization-induced performance degradation. Throughout the paper, we also refer to the MRACos regularizer and periodic compressors as soft and hard compressors, respectively. The soft compressor affects the gradient calculation only through the use of the regularization term, while the hard compressor quantizes each model weight to the exact centroid that are also used at runtime inference. We apply our S8BQAT into the ASR task and measure the performance in terms of word error rate (WER) and on-device runtime user perceived latency (UPL) on various runtime settings. Results show that the proposed S8BQAT achieves superior WER-UPL tradeoff compared to an 8-bit baseline. In particular, we increase the number of model parameters by 10.3%, thus reducing WER by 4-16% relative while reducing UPL by 5% with S8BQAT.

The rest of the paper is structured as follows. The proposed S8BQAT and the associated data loading mechanism are introduced in Sec.2. In Sec.3, we conduct runtime validation including the measurement of model accuracy and on-device UPL from general-purposed RNN-T models for speech recognition, along with an ablation study. Sec. 4 concludes the paper.

10.21437/Interspeech.2022-874

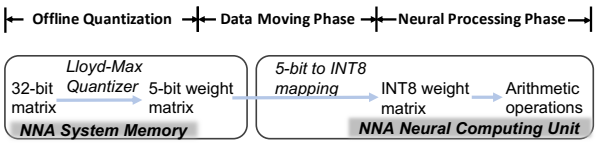


Figure 1: *NNA loads 32-bit model weights in the sub-8-bit form and decompresses them for integer arithmetic operations.*

2. Algorithm description

2.1. Sub-8-bit data loading mechanism in NNA

The proposed algorithm compresses deep learning models that are hosted on an NNA. At runtime, the NNA loads the model weights from the system memory into the neural computing unit's local memory buffer (data moving phase) to perform bit-wise arithmetic operations. This data moving phase needs to be accounted for each model inference call as often NNAs have limited on-chip memory to fully cache model weights locally.

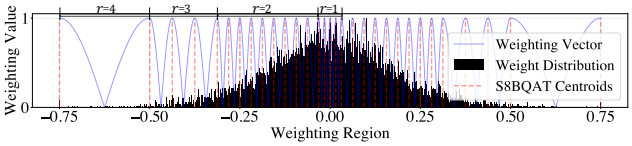
High efficiency can be achieved by accelerating matrix related operations, such as matrix multiplication on NNAs. Yet, the data moving phase is time consuming due to the constrained available memory bandwidth on-device. To reduce the bandwidth and consequently the latency, we quantize model weights into sub-8-bit (offline quantization), then transfer them to the NNA's neural computing unit where sub-8-bit weights are decompressed into INT8 format for the neural processing phase (see Figure 1). Consider a weight matrix with the shape of $(1024, 4096)$. With Lloyd-Max quantizer representing all weights by 32 distinct values or 5-bit, the compressed matrix requires 2.5MB, instead of 4MB for 8-bit, thus reducing the in-memory size and data transfer latency by 37.5%.

2.2. Lloyd-Max scalar quantization theory

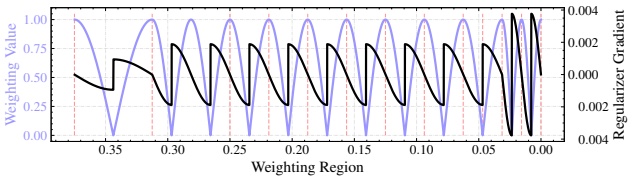
The problem of compressing a set of weights into another set with a smaller cardinality is solved by S. Lloyd and J. Max, which is often referred to as Lloyd-Max scalar quantization theorem [22, 23]. Let $\{c_1, ..., c_k\}$ be k partitions of the model weights $\boldsymbol{w} = \{w_1, ..., w_n\}$, and $\{m_1, ..., m_k\}$ be the prototypes (or quantization centroids) for the corresponding k partitions. Lloyd Max algorithm minimizes the mean squared error between the model weights and corresponding centroids: $\mathcal{H} = \sum_{k=1}^{K} \sum_{w_i \in c_k} ||w_i - m_k||^2$. The solutions of $\{m_1, ..., m_k\}$ can be derived in closed-form [22, 23], or via an iterative method which is also used for K-Means clustering [24]: for each iteration, we 1) assign each w_i to the nearest m_k: $c_k^{\text{new}} = \{w_i : \arg\min_{k'} ||w_i - m_{k'}||^2 = k\}$; and 2) update prototypes by setting $m_k^{\text{new}} = \frac{1}{|c_k^{\text{new}}|} \sum_{w_i \in c_k^{\text{new}}} w_i$, where both steps decrease $\mathcal{H}$ unless the algorithm has converged. With all weights $w \in c_i$ represented by $\{m_1, ..., m_k\}$, they are quantized into $\lceil \log_2 k \rceil$ bits.

2.3. Lloyd-Max quantizer-like regularization

Directly applying Lloyd-Max scalar quantizer to sub-8-bit QAT can be problematic. Firstly, the quantization centroids are not guaranteed to conform to INT8 format. Consequently, at runtime, when they are updated to $k/128$, where the integer $k \in [-128, 127]$, the model performance is subject to degradation. Secondly, executing the Lloyd-Max algorithm per training step is computationally expensive and memory consuming.



(a) Multi-regional absolute cosine regularizer with 31 peaks



(b) Gradient decays when the cosine frequency gets smaller

Figure 2: *Proposed soft compressor and its gradient*

To mitigate this issue, we propose multi-regional absolute cosine (MRACos) regularizer

$$\mathcal{L}_{\text{MRACos-reg}}(\boldsymbol{w}) = \sum_{w_i \in \boldsymbol{w}} \sum_{r=1}^{R} \delta_r \lambda_r (1 - |cos(\pi \theta_r w_i)|), \quad (1)$$

where $|cos(\pi\theta_r w_i)|$, λ_r and θ_r define the weighting vector, regularization weight coefficient and frequency of the cosine function in region r, while $\delta_r(w_i)$=1 for all $w_i \in \boldsymbol{w}$ that are inside the range of the r-th region, and $\delta_r(w_i)$=0 otherwise. It approximates the Lloyd-Max quantization centroids per region, e.g. r=3 in Figure 2 (a). Note that the frequency of the cosine function for all regions in Eq.1 is chosen such that regularizer maxima have INT8 format $k/128$, in which $k \in [-128, 127]$, and closest to the Lloyd-Max quantizer. The weighting vector penalizes model weights by how far they are off the nearest centroid: as shown in Eq.1, the further the model weight is off, the larger penalty is applied; the weight receives no penalty when it's on the centroid, as it leads to no extra degradation when being quantized. Weights outside all regions are clipped.

The gradient of MRACos regularizer can be calculated as,

$$\frac{\partial \mathcal{L}_{\text{MRACos-reg}}}{\partial w_i} = \sum_{r=1}^{R} \theta_r \lambda_r (-1)^{\xi} \pi \delta_r sin(\pi \theta_r w_i), \quad (2)$$

in which ξ=0 if $w \in [\frac{t-2}{\theta_r}, \frac{t}{\theta_r}]$ with t=$\{\cdots, -3, -1, 1, 3, \cdots\}$ and ξ=1, otherwise. Note that the cosine frequency θ_r in Eq. 2 becomes a decay factor in the gradient of the regularizer. For a region with a relatively small cosine frequency, the gradient there will be comparably small (see Figure 2 (b)). Admittedly, with a relatively large λ_r, weights will be more aggressively aggregated toward quantization centroids for faster quantization convergence. However, setting a high regularization weight will negatively affect the model performance, as discussed in Sec.3.

2.4. Periodic hard compressor

Instead of increasing the regularization weight to address the gradient decay issue, we introduce a periodic hard compressor that performs runtime quantization during model training per τ epochs (Figure 3). We measure the quantization convergence rate of weights for the k-th partition c_k, and its centroid m_k as

$$\gamma_k = \frac{|||w_i - m_k||^2 < \epsilon|}{|w_i \in c_k|}, \quad (3)$$

where the threshold ϵ is relatively small. As discussed in Sec.3, γ_k can be boosted effectively with the hard compressor to reduce the quantization induced degradation at runtime.

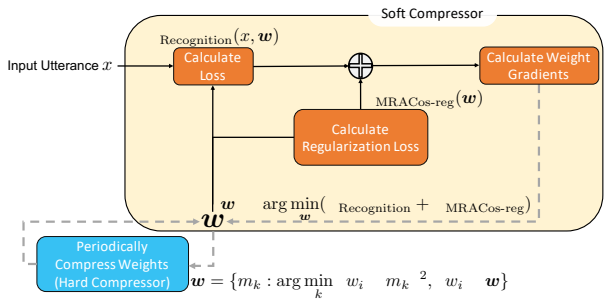


Figure 3: *Data flow during training with S8BQAT: the soft compressor operates at the gradient level while the hard compressor mimics runtime/post-training quantization.*

Table 1: *Architecture for M-II and M-III. S8BQAT can be enabled layer-wise: layers of M-II in gray are with 5-bit S8BQAT while others in 8-bit. M-III applies 5-bit S8BQAT to all layers.*

	Layer	Kernel	Recurrent Kernel	#Params
Encoder	L0	(192, 4096)	(1024, 4096)	47.8
	L1	(1024, 4096)	(1024, 4096)	
	L2	(2048, 4480)	(1120, 4480)	
	L3, L4	(1120, 4480)	(1120, 4480)	
	Projection	(1120, 512)	–	
Decoder	L0	(512, 4352)	(1088, 4352)	17.0
	L1	(1088, 4352)	(1088, 4352)	
	Projection	(1088, 512)	–	
Jointer	L0	(512, 2501)	–	2.6
	Decoder Embedding	(2501, 512)	–	

3. Experiment

3.1. Experimental setup

To validate the effectiveness of the proposed S8BQAT method, we apply it into the ASR task using the Recurrent Neural Network - Transducer (RNN-T) [25] architecture and measure two performance metrics: the word error rate (WER) for accuracy and user perceived latency (UPL). We consider several RNN-T variants, and compare the performance against a linear quantization-aware training baseline method [17]. Essentially, the proposed method differs from its baseline in two aspects: the soft compressor approximates Lloyd-Max scalar quantization to regularize model weights in a non-linear space while the hard compressor is introduced and invoked periodically to emulate runtime quantization.

We consider 3 RNN-T model settings, all with 5 encoding layers, 2 decoding layers, 1 joint layer and 2.5k word pieces: M-I is the baseline including 61.0M parameters, and trained with the 8-bit linear QAT mechanism [17]. M-II and M-III adopt the topology of M-I but with a slightly larger number of hidden units: specifically, the number of hidden units are increased from 1024 to 1120 and 1088 for Encoder L2-L4/Projection layer and Decoder, respectively (see Table 1). Consequently, the number of parameters for M-II and M-III is 67.3M, or 10.3% larger than M-I. Under S8BQAT, the process is as follows. We first train M-II and M-III for a pre-determined duration, e.g. 10k out of 850k steps; then extract the weight distribution and sub-8-bit configurations for all layers of M-III, and all but L0 layers of M-II; then apply the proposed S8BQAT mechanism (see Figure 3) to train M-II and M-III for the rest of the epochs; and finally, compress all sub-8-bit layers of M-II and M-III after the training finishes.

M-I, II and III are trained with a de-identified far-field dataset including 100k hours of human transcribed utterances and 40k hours of utterances generated by self-supervised learning speech model. All models are trained for 850k steps. We consider 4 far-field test sets for the WER evaluation: Frequent, Appliances, Entertainment, Rare, each with 50K utterances. Frequent and Rare datasets include tasks that are frequently or rarely queried, while Appliances and Entertainment contain queries from specific supported Appliances and Entertainment tasks. We benchmark latency metrics on 6k test utterances.

For reproducible purposes, we also train RNN-T models with S8BQAT on LibriSpeech corpus [26] with 960k hours of training data for 120k steps, and measure WERs with 5.4k, 5.3k, 5.4k, and 5.1k hours of dev-clean, dev-other, test-clean and test-other data, respectively.

3.2. Experimental results

3.2.1. Accuracy and latency analysis

We compare the accuracy and latency performance among M-I, M-II and M-III in Table 2. We normalize the WER numbers by the WER of M-I for Frequent test set. The latency numbers are normalized by the latency of M-I for latency test set. The latency is observed to be reduced even when the number of parameters increases thanks to the employment of S8BQAT: concretely, compared to the 8-bit baseline (M-I), M-II lowers the UPL-P50 by 5% and UPL-P90 by 3%; a more aggressive quantization configuration in M-III further brings down the latency with 6% UPL-P50 reduction and 5% UPL-P90 reduction, respectively. Moreover, M-II and M-III consistently show the accuracy improvement over the 8-bit baseline from all 4 test sets, thanks to their slightly increased number of parameters.

The mechanism of S8BQAT is depicted in Figure 4, where we extract 4 snapshots of M-II during training and plot their weight distributions. With the MRACos regularizer, weights are effectively driven to quantization centroids even with only 50k training steps (Figure 4 (a)). However, peaks in the tail of the distribution (grey circle in Figure 4 (a)) are far less spiky (indicating a lower quantization convergence rate, γ). This is mitigated at step 500k (see Figure 4 (c)), as all 5-bit regularized weights are hard quantized via the periodic hard compressor at step 350k (see Figure 4 (b)). Had the hard compressor not been invoked, the convergence rate at the tail of the weight distribution would have still been low, due to the decayed gradient issue as discussed in Sec.2. On the other hand, utilizing only the hard compressor results in large performance fluctuation and fundamentally worse WERs [17]. Thanks to the quantization-aware training with MRACos regularizer, the weights are gradually pushed towards the quantization centroids, leading to smaller fluctuations in the validation loss (see Figure 4 (d)).

3.2.2. Ablation analysis

We also conduct the ablation analysis (Table 3) by alternating the regularization weight (λ) for the soft regularizer and the hard compression period (τ). The baseline models have M-I architecture and linear-QAT with 8, 5, and 4-bit weights, i.e. the model is QAT trained and post-training quantized with x bits in which x=$\{8, 5, 4\}$. For S8BQAT models, we consider 4-bit and 5-bit, training/compressing and post-training quantizing all layers. All models are trained on LibriSpeech dataset with all other training hyperparameters being the same.

As shown in Table 3, the linear-QAT method [17] performs well in 8-bit, but fails to preserve the accuracy level in sub-8-bit modes. In contrast, the proposed soft compressor leverages

ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 978-1-7281-7605-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICASSP39728.2021.9413678

SOURCE-AWARE NEURAL SPEECH CODING FOR NOISY SPEECH COMPRESSION

Haici Yang[1], *Kai Zhen*[1], *Seungkwon Beack*[2], *Minje Kim*[1]

[1]Indiana University, Department of Intelligent Systems Engineering, Bloomington, IN, USA
[2]Electronics and Telecommunications Research Institute, Daejeon, South Korea

ABSTRACT

This paper introduces a novel neural network-based speech coding system that can process noisy speech effectively. The proposed source-aware neural audio coding (SANAC) system harmonizes a deep autoencoder-based source separation model and a neural coding system, so that it can explicitly perform source separation and coding in the latent space. An added benefit of this system is that the codec can allocate a different amount of bits to the underlying sources, so that the more important source sounds better in the decoded signal. We target a new use case where the user on the receiver side cares about the quality of the non-speech components in the speech communication, while the speech source still carries the most important information. Both objective and subjective evaluation tests show that SANAC can recover the original noisy speech better than the baseline neural audio coding system, which is with no source-aware coding mechanism, and two conventional codecs.

Index Terms— Speech enhancement, speech coding, source separation

1. INTRODUCTION

Breakthroughs made in deep learning for the past decade have shown phenomenal performance improvements in various pattern recognition tasks, including media compression and coding. Seminal works are proposed in the lossy image compression domain, where autoencoders are a natural choice. With an autoencoder, the encoder part converts the input signal into a latent feature vector, followed by the decoder that recovers the original input [1, 2]. Compression is achieved when the number of bits used to represent the latent vector (or code) is smaller than that of the raw input signal. With increased computational complexity, the deep autoencoders have shown superior compression performance to traditional technology.

Neural speech coding is an emerging research area, too. Autoregressive models, such as WaveNet [3], have shown a transparent perceptual performance at a very low bitrate [4, 5], surpassing that of traditional coders. Another branch of neural speech coding systems takes a frame-by-frame approach, feeding time-domain waveform signals to an simpler end-to-end autoencoding network. Kankanahalli proposes a model that consists of fully convolutional layers to integrate dimension reduction, quantization, and entropy control tasks [6]. Cross-module residual learning (CMRL) inherits the convolutional pipeline and proposes a cascading structure, where multiple autoencoders are concatenated to work on the residual signal produced by the preceding ones [7]. In [8], CMRL is coupled with a trainable linear predictive coding (LPC) module as a preprocessor. It further improves the performance and lowers the model complexity down to 0.45 million parameters, eventually outperforming AMR-WB [9].

In this work, we widen the scope of speech coding applications by taking into account noisy speech as the input. Additional sound sources often accompany real-world speeches. However, traditional speech codecs are mostly based on the speech production models [10, 9], thus lacking the ability to model the non-speech components mixed in the input signals. Efforts to address the problem are partly reflected in the MPEG unified speech and audio coding (USAC) standard [11, 12]. USAC tackles speech signals in the mixture condition by switching between different tools defined for different kinds of signals, such as speech and music. However, the switching decision does not consider the mixed nature within the frame, which requires explicit source separation. Meanwhile, AMR-WB's discontinuous transmission (DTX) mode also considers the mixed nature of input speech by deactivating the coding process for the non-speech periods [9]. Lombard et al. improved DTX by generating artificial comfort noise that smooths out the discontinuity [13]. Still, for the frames where both speech and non-speech sources co-exist, it is difficult to effectively control the bitrate using DTX. Similar ideas have been used in transform coders for audio compression, where the dynamic bit allocation algorithm based on psychoacoustic models can create a spectral hole in low bitrate cases. Intelligent noise gap filling can alleviate the musical noise generated from this quantization process [14, 15], while it is to reduce the artifact generated from the coding algorithm, rather than to model the non-stationary noise source separately from the main source.

To that end, we propose source-aware neural audio coding (SANAC) to control the bit allocation to multiple sources differently. SANAC does not seek a speech-only reconstruction, e.g., denoising the mixture input and coding it simultaneously [16]. Instead, we target the use case where the user still wants the code to convey the non-speech components to better understand the transmitter's acoustic environment. We empirically show that the sources in the mixture can be assigned with unbalanced bitrates depending on their perceptual or applicational importance and entropy in the latent space, leading to a better objective and subjective quality.

2. MODEL DESCRIPTION

The proposed SANAC system harmonizes a source separation module into the neural coding system. Our model performs explicit source separation in the feature space to produce source-specific codes, which are subsequently quantized and decoded to recover the respective sources. The source-specific code vectors are learned using a masking-based approach as in TasNets [17, 18], while we utilize the orthogonality assumption between the source-specific code vectors to withdraw the separator module in the TasNet architec-

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).

ture and reduce the encoder complexity. Soft-to-hard quantization [2] quantizes the real-valued source-specific codes. Bitrate control works on each sources as well.

2.1. Orthogonal code vectors for separation

As a codec system, the model consists of an encoder that converts time-domain mixture frame $\boldsymbol{x} \in \mathbb{R}^N$ into code vector $\boldsymbol{z} \in \mathbb{R}^D$: $\boldsymbol{z} \leftarrow \mathcal{F}_{\text{enc}}(\boldsymbol{x})$. To marry the source separation concept, we assume K mask vectors $\boldsymbol{m}^{(k)} \in \mathbb{R}^D$ that can decompose the code vector into K components: $\sum_{k=1}^{K} m_d^{(k)} = 1$. Note that $m_d^{(k)}$ is the probability of d-th code value belonging to k-th source. In addition, we further assume that this probabilistic code assignments to the sources are actually determined by one-hot vectors, so that the masking process assigns each code value to only one source, i.e.,

$$m_d^{(k)} = \begin{cases} 1 & \text{if } \arg\max_j m_d^{(j)} = k \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The TasNet models estimate similar masking vectors via a separate neural network module, which led to the state-of-the-art separation performance. In there, the estimated mask values are drastically distributed to either near zero or one, making the masked code vectors nearly orthogonal from each other. However, the sigmoid-activated masks in the TasNet architecture do not specifically assume a hard assignment.

From now on, we assume orthogonal code vectors per source as a result of hard masking, i.e., $\boldsymbol{z}^{(1)} \perp \boldsymbol{z}^{(2)} \perp \cdots \perp \boldsymbol{z}^{(K)}$, where $\boldsymbol{z}^{(k)} = \boldsymbol{m}^{(k)} \odot \boldsymbol{z}$, k-th source's code vector defined by the Hadamard product $\odot$ between the mask and the mixture code.

The proposed orthogonality leads us to a meaningful structural innovation. Instead of estimating the mask vector for every input frame, we can use structured masking vectors that force the code values to be grouped into K exclusive and consecutive subsets. For example, for a two-source case with $D = 8$, $\boldsymbol{m}^{(1)} = [1, 1, 1, 1, 0, 0, 0, 0]^\top$. Hence, we can safely discard the masked-out elements (the latter four elements), by defining its truncated version as $\mathbf{z}^{(k)} \in \mathbb{R}^{D/K}$. The concatenation of the truncated code vectors determines the final code vector: $\boldsymbol{z} = [\mathbf{z}^{(1)\top}, \mathbf{z}^{(2)\top}, \cdots, \mathbf{z}^{(K)\top}]^\top$.

In practice, we implement the encoder as a 1-d convolutional neural network (CNN) of which output is a $2L \times P$ matrix, where $2L$ is the number of output channels (see Figure 1, where $L = 6$, $P = 256$). We collect the first L channels of this feature map (dark red bars) as our codes for speech, i.e., $\mathbf{z}^{(1)}$ corresponds to the vectorized version of the upper half of the feature map of size $L \times P$, or $LP = D/K$, where D should be an integer multiple of K. The other half is for the noise source. Since the decoders are learned to predict individual sources, this implicit masking process can still work for source separation.

2.2. Soft-to-hard quantization

Quantization is a mapping process that replaces a continuous variable with its closest discrete representative. Since it is not a differentiable process, incorporating it into a neural network requires careful consideration. Soft-to-hard quantization showed successful performance both in image and speech compression models [2, 6, 7]. The idea is to formulate this cluster assignment process as a softmax classification during the feedforward process, which finds the nearest one among M total representatives $\boldsymbol{\mu}_m$ for the given code vector $\boldsymbol{y}$ as follows:

$$d_m = \mathcal{E}(\boldsymbol{y} || \boldsymbol{\mu}_m), \quad \boldsymbol{p} = \text{Softmax}(-\alpha \boldsymbol{d}), \tag{2}$$

$$\text{Testing: } \bar{\boldsymbol{y}} = \boldsymbol{\mu}_{\arg\max_m p_m}, \quad \text{Training: } \bar{\boldsymbol{y}} = \sum_{m=1}^{M} p_m \boldsymbol{\mu}_m,$$

where the algorithm first computes the Euclidean distance vector $\boldsymbol{d}$ against all the representatives (i.e., the cluster means). The negative distance values work as similarity scores for the softmax function. Using the softmax result, the probability vector $\boldsymbol{p}$ of the cluster membership, we can construct the quantized code vector $\bar{\boldsymbol{y}}$: during test time, simply choosing the closest one will do a proper quantization. Since the $\arg\max$ operation is not differentiable, for training, we do a convex combination of the cluster centroids to represent the quantized code, as a differentiable surrogate of the hard assignment process. The discrepancy between training and testing is reduced by controlling the scaling hyperparameter α, which makes the softmax probabilities more drastic once it is large enough (i.e., a one-hot vector in the extreme case). Note that it also learns the cluster centroids $\boldsymbol{\mu}$ as a part of the learnable network parameters rather than employing a separate clustering process to define them.

In previous works the quantization has been on scalar variables, i.e., $\boldsymbol{y} \in \mathbb{R}^1$ [6, 7, 8]. In this work, the soft-to-hard quantization performs vector quantization (VQ). We denote the CNN encoder output by $\boldsymbol{Z} \in \mathbb{R}^{2L \times P}$, which consists of $K = 2$ code blocks: $\boldsymbol{Z} = [\boldsymbol{Z}^{(1)}; \boldsymbol{Z}^{(2)}]$. Then, each code vector for quantization is defined by the p-th feature spanning over L channels: $\boldsymbol{y} = \boldsymbol{Z}_{1:L,p}^{(k)}$, having $L = 6$ as the VQ dimension in our case.

2.3. Source-wise entropy control

The theoretical lower bound of the bitrate, as a result of Huffman coding, can be defined by the entropy of the quantized codes. The frequency of the cluster means defines the entropy of the source-specific codes: $\mathcal{H}\big(\boldsymbol{\mu}^{(k)}\big) = -\sum_{m=1}^{M} q_m^{(k)} \log q_m^{(k)}$, where $q_m^{(k)}$ denotes the frequency of m-th mean for the k-th source. Meanwhile, the entropy for the code of the mixture signal is smaller than or equal to the sum of the entropy of all sources: $\mathcal{H}(\ddot{\boldsymbol{\mu}}) \leq \sum_{k=1}^{K} \mathcal{H}\big(\boldsymbol{\mu}^{(k)}\big)$, where $\ddot{\boldsymbol{\mu}}$ is the set of quantization vector centroids learned in a source-agnostic way, i.e., directly from the mixture signals. Therefore, in theory, SANAC cannot achieve a better coding gain than a codec that works directly on the mixture.

However, SANAC can still benefit from the source-wise coding, especially by exploiting the perceptual factors. Our main assumption in this work is that the perceptual importance differs from source-by-source, leading to a coding system that can assign different bitrates to different sources. For noisy speech, for example, we will try to assign more bits to the speech source. Consequently, although the user eventually listens to the recovered mixture of speech and noise (a) the perceptual quality of the speech component is relatively higher (b) the codec can achieve a better coding gain if the noise source' statistical characteristics is robust to low bitrates.

Our argument is based on the codec's ability to control the entropy of the source-specific codes. In SANAC, we adopt the entropy control mechanism proposed in [2], but by setting up a per-source loss between the target $\xi^{(k)}$ and the actual entropy values: $\big(\xi^{(k)} - \mathcal{H}\big(\boldsymbol{\mu}^{(k)}\big)\big)^2$. While this loss does not guarantee the exact bitrate during the test time, in practice, we observe that the actual bitrate is not significantly different from the target.

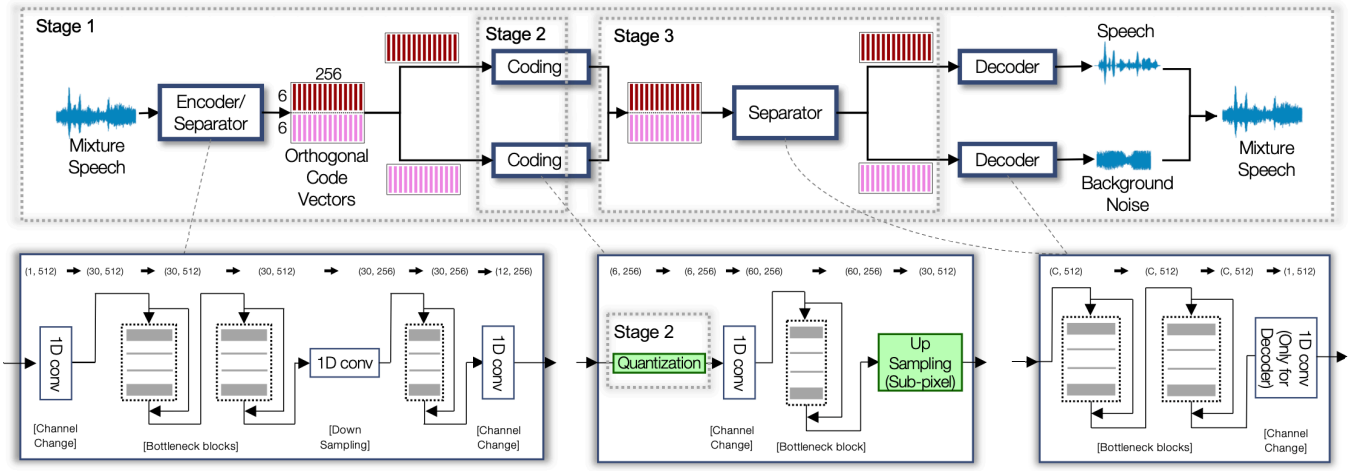


Fig. 1: Schematic diagram of source-aware speech coding. At Stage 1, the Stage 2 and 3 modules are not trained, while Stage 2 updates all parameters except for the Stage 3 modules. At Stage 3, all parameters are optimized.

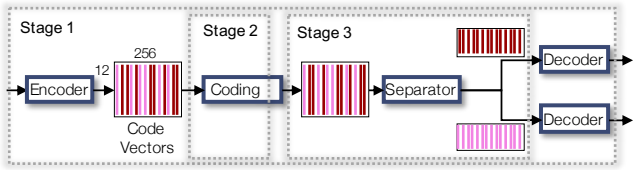


Fig. 2: Schematic diagram of DNN-based baseline model

2.4. Decoding and the final loss

The source-specific truncated codes, after the quantization, $\bar{\boldsymbol{Z}}^{(k)}$, are fed to the decoder part of the network. The decoder function works similarly to Conv-TasNet [18] in that the decoder runs K times to predict K individual source reconstructions from K sourse-specific feature maps as the input. However, SANAC's decoding is different from Conv-TasNet's as the decoder input is the quantized codes. In addition, our model cares about the quality of the recovered mixture, not only the separation quality.

Our training loss considers all these goals, consisting of the main mean squared error (MSE)-based reconstruction term and the entropy control terms. More specifically, for the noisy speech case $\boldsymbol{x} = \boldsymbol{s} + \boldsymbol{n}$ ($k = 1$ for speech and $k = 2$ for noise), the MSE loss is for the speech source reconstruction $\hat{\boldsymbol{s}}$ and the mixture reconstruction $\hat{\boldsymbol{x}}$, while the noise source reconstruction $\hat{\boldsymbol{n}}$ is implied in there. We regularize the total entropy as well as the ratio between the two source-wise entropy values:

$$\begin{aligned}\mathcal{L} = {} & \lambda_{\text{MSE}}\big(\mathcal{E}_{\text{MSE}}(\boldsymbol{s}||\hat{\boldsymbol{s}}) + \mathcal{E}_{\text{MSE}}(\boldsymbol{x}||\hat{\boldsymbol{x}})\big) \\ & + \lambda_{\text{EntTot}}\Big(\xi - \mathcal{H}\big(\boldsymbol{\mu}^{(1)}\big) - \mathcal{H}\big(\boldsymbol{\mu}^{(2)}\big)\Big)^2 \\ & + \lambda_{\text{Ratio}}\left(\psi - \frac{\mathcal{H}\big(\boldsymbol{\mu}^{(1)}\big)}{\mathcal{H}\big(\boldsymbol{\mu}^{(2)}\big)}\right)^2,\end{aligned} \tag{3}$$

where ξ and ψ are the target total entropy and the target ratio, respectively.

3. EXPERIMENT

3.1. Dataset

500 and 50 utterances are randomly selected from training and test set of TIMIT corpus [19]. We generate 10 contaminated samples out of each clean utterance by adding 10 different non-stationary background sources, {*bird singing, casino, cicadas, typing, chip eating, frogs, jungle, machine gun, motorcycle, ocean*} used in [20]. Every contaminated speech waveform was segmented into frames of 512 samples (32ms), with overlap of 64 samples. We apply a Hann window of size 128 samples only to the overlapping regions. Since there are $16000/448$ frames per second and each frame produces P code vectors for VQ, for the entropy of a source-specific codebook ξ, the bitrate is $16000P\xi/488$, e.g., 9.14kbps when $P = 256$ and $\xi = 1$.

3.2. Training process

Adam optimizer with an initial learning rate 0.0001 trains the models [21]. Both SANAC and the baseline are trained in three stages. Every jump to next stage is triggered when the validation loss stops improving in 3 consecutive epochs. We stop the training updates after validation loss does not improve for 20 epochs.

• *Stage 1*: For the first three epochs, the model trains the encoder to separate the input into the speech and background sources, that are represented by the two orthogonal code vectors. No quantization is involved in yet, but this stage better initializes the parameters for the quantization process. The encoder consists of a few bottleneck blocks that are commonly used in ResNet [22]. With the bottleneck structure, the input and output feature maps can be connected via an identity shortcut with less filters to learn in between. The encoder module also employs a 1-d convolution layer to downsample the feature map from 512 to 256, followed by another bottleneck block and a channel changing layer to yield two sets of code maps of 6×256 each. The learned source-specific feature maps are fed directly to the channel changer with no quantization, followed by an upsampler and the final decoder. In terms of upsampling, we interlace two adjacent channels into one, doubling the number of features up to 512 while halving the number of channels as introduced in [23], and then adopted for neural speech coding [6, 7].

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 $\bar{\boldsymbol{h}}$.

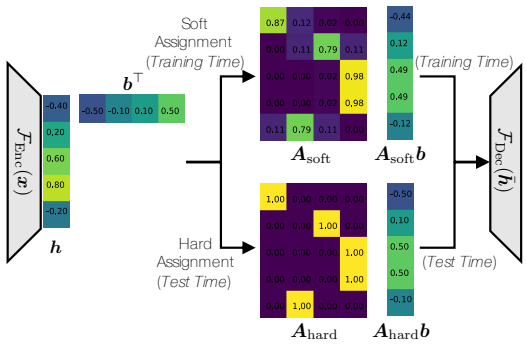


Fig. 1. An example of the softmax quantization process.

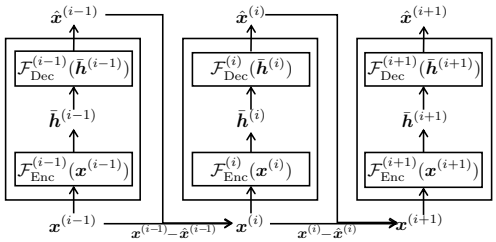


Fig. 2. The CMRL residual coding scheme.

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.

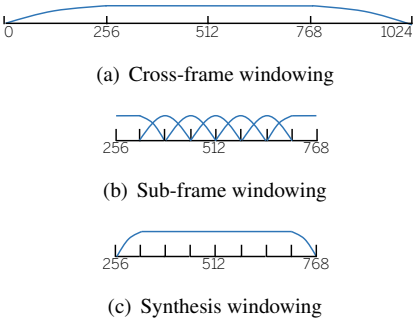


Fig. 3. LPC windowing schemes

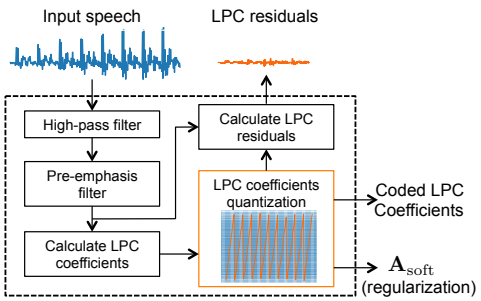


Fig. 4. The trainable LPC analyzer

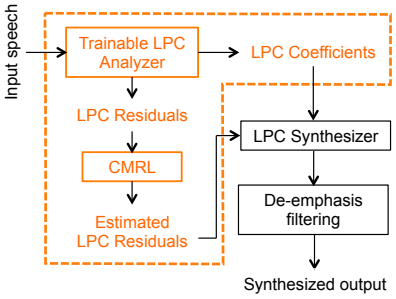


Fig. 5. Overview of the CQ system.

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.

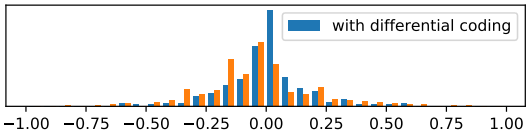


Fig. 6. Differential coding enables a more centralized distribution

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

ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 978-1-7281-7605-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICASSP39728.2021.9413844

SPARSIFICATION VIA COMPRESSED SENSING FOR AUTOMATIC SPEECH RECOGNITION

Kai Zhen[1*], *Hieu Duy Nguyen*[2], *Feng-Ju Chang*[2], *Athanasios Mouchtaris*[2], *and Ariya Rastrow*[2]

[1]Indiana University Bloomington
[2]Alexa Machine Learning, Amazon, USA
zhenk@indiana.edu, {hieng, fengjc, mouchta, arastrow}@amazon.com

ABSTRACT

In order to achieve high accuracy for machine learning (ML) applications, it is essential to employ models with a large number of parameters. Certain applications, such as Automatic Speech Recognition (ASR), however, require real-time interactions with users, hence compelling the model to have as low latency as possible. Deploying large scale ML applications thus necessitates model quantization and compression, especially when running ML models on resource constrained devices. For example, by forcing some of the model weight values into zero, it is possible to apply zero-weight compression, which reduces both the model size and model reading time from the memory. In the literature, such methods are referred to as sparse pruning. The fundamental questions are when and which weights should be forced to zero, i.e. be pruned. In this work, we propose a compressed sensing based pruning (CSP) approach to effectively address those questions. By reformulating sparse pruning as a sparsity inducing and compression-error reduction dual problem, we introduce the classic compressed sensing process into the ML model training process. Using ASR task as an example, we show that CSP consistently outperforms existing approaches in the literature.

Index Terms— Model pruning, automatic speech recognition (ASR), sparsity, Recurrent Neural Network Transducer (RNN-T), compressed sensing.

1. INTRODUCTION

Automatic Speech Recognition (ASR) is an important component of a virtual assistant system. The main focus of ASR is to convert users' voice command into transcription, based on which further processing will act upon. Recently, end-to-end (E2E) approaches have attracted much attention due to their ability of directly transducing audio frame features into sequence outputs [1]. Without explicitly imposing/injecting domain knowledge and manually tweaking intermediate components (such as lexicon model), building and maintaining E2E ASR system is much more efficient than a hybrid deep neural network (DNN)-Hidden Markov Model (HMM) model.

In order to provide the best user experience, an ASR system is required to achieve not only high accuracy but also small user-perceived latency. This motivates the trend of moving the processing from Cloud/remote servers to users' device to reduce the latency further. More often than not, the hardware limitations impose strict constraints on the model complexity [2, 3]. Firstly, the hardware may only support integer arithmetic operations for run-time inference, which necessitates model quantization. Secondly, a hardware often performs multiple tasks supported by different models in sequence. The hardware, with limited memory size, thus needs to move multiple models in and out of the processing units.

To compress the model for the hardware, two widely applied methods are (a) model selection/structured pruning, i.e. choosing a model structure with pruned layers/channels and small performance degradation [4, 5], and (b) zero-weight compression/sparse pruning, i.e. pruning small-value weights to zero [6, 7]. Model selection differs from sparse pruning in that it deletes entire channels or layers, showing a more efficient speedup during inference, yet with a more severe performance degradation [4, 5]. These two types of methods are usually complementary: after being structurally pruned, a model can also undergo further zero-weight compression to improve the inference speed. In this study, we focus on sparse pruning, targeting at lowering the memory storage and bandwidth requirement as it largely contributes to the latency for on-device ASR models.

A naïve approach for sparse pruning is to push the weight values smaller than a threshold to zero after training, which often leads to significant performance degradation [6]. To mitigate this problem, Tibshirani *et. al.* applied *LASSO* regularization to penalize large-value model weights [8]. The drawbacks are twofold: firstly, it does not exert an explicit specification of the target sparsity level; secondly, it is subject to the gradient vanishing issue when the model has more and more layers. Gradual pruning approach [6] resolves those concerns by defining a sparsity function that maps each training step to a corresponding intermediate sparsity level. During the model training, the pruning threshold is adjusted gradually according to the function to eventually reach the target sparsity level. However, gradual pruning assumes that pruning the values smaller than the threshold will lead to the least degradation, which is heuristic and sub-optimal. Consequently, gradual pruning techniques provide a guidance on "when to prune and by how much", but a less satisfying answer for "which (weights) to prune", thus leading to inefficiency.

In this work, we propose a compressed sensing (CS)-based pruning method, referred to as CSP subsequently, that is sparse-aware and addresses both "*when to prune*" and "*which to prune*". CSP reformulates the feedforward operations in machine learning architectures, such as Long Short Term Memory (LSTM) or Fully-Connected (FC) cells, as a sensing procedure with the inputs and hidden states being random sensing matrices. Under that perspective, a sparsification process is to enhance the sparsity and reduce the compression error, due to pruning, simultaneously. Following [9], we adopt the ℓ_1 regularization to enforce sparsity and the ℓ_2 regularization to mitigate the compression loss, and reformulate the

*This work was conducted during Kai's internship in Amazon Pittsburgh, PA, USA.

sparsification procedure as an optimization problem. We demonstrate the effectiveness of our method by compressing recurrent neural network transducer (RNN-T), one of the E2E ASR models. The RNN-T model is sparsified via a hybrid training mechanism in which CSP is conducted during the feedforward stage, along with the back propagation for the global optimization. Our proposed method constantly outperforms the state-of-the-art gradual pruning approach in terms of the word error rate (WER) under all settings. In particular, with a sparisty ratio of 50% where half of the weights are 0, CSP yields little to no performance degradation on LibriSpeech dataset.

The rest of the paper is structured as follows. In Sec. 2, we briefly review related pruning methods. Our CSP method is introduced in Sec. 3. Sec. 4 describes our experiment setup and results. Finally, we conclude our work with some remarks in Sec. 5.

2. RELATED WORK

One of the most straightforward approaches for sparse-aware training is applying ℓ_k regularization, where $k = 0, 1, 2$, etc. There has been a rich literature in comparing various forms of sparsity regularizers. Consider the model training: $\mathbb{W} \leftarrow \arg\min_{\mathbb{W}} \mathcal{L}_{\text{accuracy}}(\mathbb{W}) + ||\mathbb{W}||_1$, where ℓ_1 norm is used on the regularization. The fundamental idea is to optimize the model prediction while penalizing large weight values. In DNN model compression, the regularization is usually implemented as an extra loss term for the training.

This training-aware sparsity regularization leads to promising pruning results especially for convolutional neural networks (CNN) [10] with residual learning techniques [11, 12], but may not apply well to models employing recurrent neural network (RNN) components such as LSTM. The error due to a global sparsity constraint ℓ_1 will be propagated to all time steps. Additionally, such drawback is much more severe for architectures, such as RNN-T, which contains feedback loop from one part of the model to the others.

Another well-known, state-of-the-art, pruning method for ML models is gradual-pruning [6]. This method does not resort to ℓ_1 regularization for sparsity, but dynamically updates the pruning threshold during model training, as is indicated by its name. To answer the question "when to prune", the authors defines a sparsity function parametrized by the target sparsity $\boldsymbol{s}_f$ at $\boldsymbol{t}_n$ step with an initial pruning step $\boldsymbol{t}_0$. Concretely, at training step $\boldsymbol{t}$, the pruning threshold is adjusted to match the sparsity $\boldsymbol{s}_t$ calculated in Eq. 1. The main complication is to adjust the pruning procedure such that the model weights are relatively converged and the learning rate is still sufficiently large to reduce the pruning-induced loss.

$$\boldsymbol{s}_t = \boldsymbol{s}_f * \left(1 - \left(1 - \frac{\boldsymbol{t} - \boldsymbol{t}_0}{\boldsymbol{t}_n - \boldsymbol{t}_0}\right)^3\right) \tag{1}$$

One concern is that finding an optimal setup for these hyperparameters can be hard without going through a rigorous ablation analysis. Furthermore, with gradual pruning, gradient-updating back-propagation is the only mechanism to limit the degradation. Most importantly, gradual pruning only addresses the question "*when to prune*" but not "*which (weights) to prune*". At each time step, the weights below the (gradually increased) threshold are to be pruned. This is based on the premise that the smaller the weight, the less important it is, which is heuristic and sub-optimal.

3. COMPRESSED SENSING BASED PRUNING

3.1. Adapting Compressed Sensing for Sparse Pruning

CS aims to compressing potentially redundant information into a sparse representation and reconstructing it efficiently [13, 14], which has facilitated a wide scope of engineering scenarios, such as medical resonance imaging (MRI) and radar imaging. For example, in MRI [15], high resolution scanned images are generated per millisecond or microsecond, leading to significant storage cost and transmission overhead. CS learns a sparse representation of each image with which, during the decoding time, CS can recover the reference image almost perfectly. Assume an image compression task with a reference image $\boldsymbol{x} \in \mathbb{R}^n$, where $\boldsymbol{x}$ is usually decomposed into an orthogonal transformation basis ψ and the activation $\boldsymbol{s}$, as $\boldsymbol{x} = \psi * \boldsymbol{s}$. Given that $\boldsymbol{s}$ satisfies $\mathcal{K}$-sparse property, CS is capable of locating those $\mathcal{K}$ salient activation elements. Concretely, CS introduces a sensing matrix ϕ to project $\boldsymbol{x}$ into $\boldsymbol{y}$, as $\boldsymbol{y} = \phi * \boldsymbol{x}$. In [9, 16], it has been proved that by optimizing the ℓ_2 loss in the sensing dimensionality while exerting the ℓ_1 norm regularizer to $\boldsymbol{s}$, a $\mathcal{K}$-sparse solution of $\boldsymbol{s}$, denoted as $\hat{\boldsymbol{s}}$, can be found in polynomial time. Consequently, $\psi * \hat{\boldsymbol{s}}$ can estimate the original image $\boldsymbol{x}$ with high fidelity and relatively small latency.

In this work, we investigate the effectiveness of CS based pruning for ML models. We consider the ASR task, i.e. converting audio speech to transcriptions, using an RNN-T architecture. Due to the space constraint, we only describe the transformation of LSTM cell, which is a major building block in various E2E ASR models. It is straightforward to extend the transformations to other architectures/layers like the fully-connected (FC) network and CNN.

Consider a vanilla LSTM cell: the element-wise multiplication between the input at time step t, $\boldsymbol{x}^{(t)}$, and kernels is given in Eq. 2, while that of hidden states from the previous step $\boldsymbol{h}^{(t-1)}$ and recurrent kernels is in Eq. 3. Here, $\boldsymbol{W}_f, \boldsymbol{W}_c, \boldsymbol{W}_i$, and $\boldsymbol{W}_o$ (correspondingly $\boldsymbol{U}_f, \boldsymbol{U}_c, \boldsymbol{U}_i$, and $\boldsymbol{U}_o$) denote the kernels (correspondingly recurrent kernels) weights of the cell (c), the input gate (i), output gate (o), and forget gate (f), respectively. All gating mechanisms to update the cell and hidden states are encapsulated in Eq. 4, where $\boldsymbol{C}^{(t)}$ is the cell state vector at time t and $\mathcal{G}$ denotes the transformation of $\boldsymbol{z}_x, \boldsymbol{z}_h, \boldsymbol{C}^{(t-1)}$ into $\boldsymbol{C}^{(t)}, \boldsymbol{h}^{(t)}$. The bias terms are omitted for ease of presentation.

$$\boldsymbol{z}_x = [\boldsymbol{W}_f, \boldsymbol{W}_c, \boldsymbol{W}_i, \boldsymbol{W}_o] \odot [\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t)}] \tag{2}$$

$$\boldsymbol{z}_h = [\boldsymbol{U}_f, \boldsymbol{U}_c, \boldsymbol{U}_i, \boldsymbol{U}_o] \odot [\boldsymbol{h}^{(t-1)}, \boldsymbol{h}^{(t-1)}, \boldsymbol{h}^{(t-1)}, \boldsymbol{h}^{(t-1)}] \tag{3}$$

$$[\boldsymbol{C}^{(t)}, \boldsymbol{h}^{(t)}] = \mathcal{G}(\boldsymbol{z}_x, \boldsymbol{z}_h, \boldsymbol{C}^{(t-1)}), \tag{4}$$

To prune all kernel weights (denoted as $\mathbb{W} = [\boldsymbol{W}_f, \boldsymbol{W}_c, \boldsymbol{W}_i, \boldsymbol{W}_o]$) in LSTM cells, we adapt and reformulate a CS-like pruning procedure by adopting ℓ_1 regularization for sparsity-inducing and ℓ_2 regularization for compression-error reduction, as outlined in Fig. 1. The procedure starts by a midway inference to collect the activation inputs $[\boldsymbol{z}_x, \boldsymbol{z}_h]$ as in Eq. 2 and Eq. 3. When those kernels are sparsified, the activation inputs will be consequently updated as $[\boldsymbol{z}'_x, \boldsymbol{z}'_h]$ (see Fig. 1). The goal is to sparsify and prune the kernels while preserving the value of $[\boldsymbol{z}_x, \boldsymbol{z}_h]$ to minimize the pruning-induced loss. To that end, the ℓ_1 regularizer is applied to the input kernels while the ℓ_2 regularizer controls the reconstruction loss on $\boldsymbol{z}_x$. Hence, our CS solver is embedded in a local optimizer triggered periodically by feedforward steps in a stochastic manner. As illustrated in the restricted isometry property (RIP), the sensing matrix ϕ is expected to be random for an accurate signal reconstruction [16]. The proposed CS solver satisfies the RIP

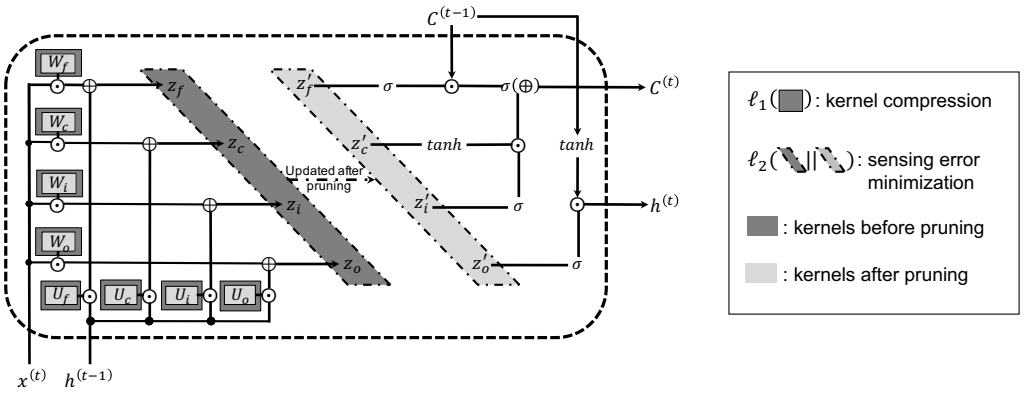


Fig. 1: A CSP-LSTM cell with the local sparse optimizer

with high probability, since the sensing matrices (the input $\boldsymbol{x}^{(t)}$ and hidden state $\boldsymbol{h}^{(t)}$) in LSTM cells vary at different time steps.

Our CS solver is kernel-wise, i.e. in each LSTM layer, there is one CS solver for each of the input-kernel and recurrent-kernel sparsification process. A general CS loss for the local optimizer is defined in Eq. 5. By adjusing the sensing coefficient, the local optimizer is capable of calibrating the model to the target sparsity by balancing the ℓ_1 and ℓ_2 regularizers,

$$\mathcal{L}_{CS}(\mathbb{W}, \boldsymbol{y}, \boldsymbol{h}) = \lambda|\mathbb{W}|_1 + ||\boldsymbol{y} - \mathbb{W} \odot \boldsymbol{h}||_2, \tag{5}$$

where the hyperparameter λ in Eq. 5 is referred to as the sensing coefficient subsequently. $\mathbb{W}$, $\boldsymbol{h}$, and $\boldsymbol{y}$ denote kernel weights, inputs, and activation input, respectively. To remove the manual-tuning, we dynamically update λ via Eq. 6.

$$\lambda \leftarrow \begin{cases} \max(\lambda_{\text{lower}}, \lambda - \epsilon), & \text{if } \boldsymbol{s} > \boldsymbol{s}_t \\ \min(\lambda_{\text{upper}}, \lambda + \epsilon), & \text{otherwise.} \end{cases} \tag{6}$$

During training, if the actual sparsity overshoots the target sparsity at a certain step, we reduce λ by ϵ. Otherwise, the λ is increased by ϵ instead. In our setup, λ is constrained between $\lambda_{\text{lower}} = 0.001$ and $\lambda_{\text{upper}} = 1.0$ with ϵ being 0.005. The pruning threshold, ρ, initialized as 0.002, is also updated as in [6] to adjust the sparsity. Algorithm 1 summarizes our proposed CSP procedure for input kernels. The CSP for recurrent kernels $\mathbb{U} = [\boldsymbol{U}_f, \boldsymbol{U}_c, \boldsymbol{U}_i, \boldsymbol{U}_o]$ is executed similarly and is omitted for brevity.

Algorithm 1 Proposed CSP for an LSTM cell

Inputs: input data at time step t, $\boldsymbol{x}^{(t)}$
the hidden state from the previous step, $\boldsymbol{h}^{(t-1)}$
the cell state from the previous step, $\boldsymbol{C}^{(t-1)}$
Outputs: updated hidden state at time step t, $\boldsymbol{h}^{(t)}$
updated cell state at time step t, $\boldsymbol{C}^{(t)}$
if sparsity level $<$ the target **then**
  Midway inference: $[\boldsymbol{z}_x, \boldsymbol{z}_h] = \mathcal{F}(\mathbb{W}, \boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t)})$
  Sensing: $\mathbb{W}' \leftarrow \arg\min_{\mathbb{W}} \mathcal{L}_{CS}(\mathbb{W}, \boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t)}, \boldsymbol{z}_x, \boldsymbol{z}_h)$
  Pruning: $\mathbb{W}^p_{(i,j)} \leftarrow \begin{cases} 0, & \text{if } \left|\mathbb{W}'_{(i,j)}\right| < \rho; \\ \mathbb{W}'_{(i,j)}, & \text{otherwise.} \end{cases}$
  Update coefficient and threshold λ and ρ
end if
Update cell outputs: $[\boldsymbol{z}_x, \boldsymbol{z}_h] = \mathcal{F}(\mathbb{W}^p, \boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t)})$
$[\boldsymbol{C}^{(t)}, \boldsymbol{h}^{(t)}] = \mathcal{G}(\boldsymbol{z}_x, \boldsymbol{z}_h, \boldsymbol{C}^{(t-1)})$

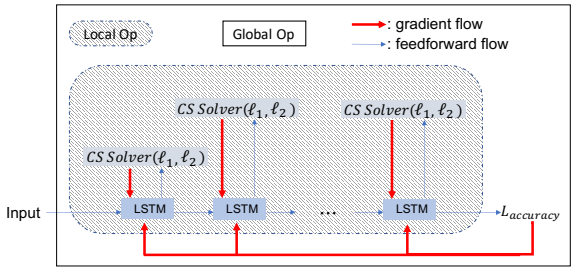


Fig. 2: Sparse-aware training scheme for CSP

3.2. Hybrid Sparse-Aware Training Scheme

The proposed CSP sparsification optimizer is combined with the conventional backpropagation algorithm in a sparse-aware training scheme (see Fig. 2). Since CSP is conducted kernel-wise for each layer, the local optimization is triggered during the feedforward stage as the samples passes through the first encoder layer to the last decoder layer sequentially. When the model makes the prediction, the global loss is calculated to update parameters in all preceding layers through backpropagation. The hybrid training scheme is not subjected to gradient vanishing thanks to the local ℓ_1 regularization, and compatible with global optimization.

4. EXPERIMENTS

4.1. Experimental Setup

We consider the ASR task with an RNN-T architecture. Comparing to the listen-attend-spell (LAS) model [17], RNN-T features online streaming capability while not requiring a separate lexicon/pronunciation system as in the connectionist temporal classification model (CTC) [18]. To rigorously evaluate CSP along with existing sparsification methods, we experiment with 4 different RNN-T based topologies, all with 5 encoding layers and 2 decoding layers. Model M-I, M-III and M-IV all have 768 units per layer (UpL) and 4000 word-pieces (WPs) while M-II uses 1024 UpL with 2500 WPs. Among 4 models, only M-III includes a joint network (J-N), which combines the outputs of RNN-T encoder and decoder to achieve better performance. M-I and M-II are trained on a far-field dataset with 25k hours of English audio, while M-III and M-IV are trained on the LibriSpeech dataset with 960 hours [19]. Note that these models are reasonably small comparing with counterparts in the literature, to highlight the effect of sparse pruning on model performance. Please refer to Table 1 for the total number of parameters of each model.

A DUAL-STAGED CONTEXT AGGREGATION METHOD TOWARDS EFFICIENT END-TO-END SPEECH ENHANCEMENT

Kai Zhen[1,2]*, Mi Suk Lee*[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@indiana.edu, lms@etri.re.kr, minje@indiana.edu

ABSTRACT

In speech enhancement, an end-to-end deep neural network converts a noisy speech signal to a clean speech directly in the time domain without time-frequency transformation or mask estimation. However, aggregating contextual information from a high-resolution time domain signal with an affordable model complexity still remains challenging. In this paper, we propose a densely connected convolutional and recurrent network (DCCRN), a hybrid architecture, to enable dual-staged temporal context aggregation. With the dense connectivity and cross-component identical shortcut, DCCRN consistently outperforms competing convolutional baselines with an average STOI improvement of 0.23 and PESQ of 1.38 at three SNR levels. The proposed method is computationally efficient with only 1.38 million parameters. The generalizability performance on the unseen noise types is still decent considering its low complexity, although it is relatively weaker comparing to Wave-U-Net with 7.25 times more parameters.

Index Terms— End-to-end, speech enhancement, context aggregation, residual learning, dilated convolution, recurrent network

1. INTRODUCTION

Monaural speech enhancement can be described as a process to extract the target speech signal by suppressing the background interference in the speech mixture in the single-microphone setting. There have been various classic methods, such as spectral subtraction [1], Wiener-filtering [2] and non-negative matrix factorization [3], to remove the noise without leading to objectionable distortion or adding too much artifacts, such that the denoised speech is of decent quality and intelligibility. Recently, the deep neural network (DNN), a data-driven computational paradigm, has been extensively studied thanks to its powerful parameter estimation capacity and correspondingly promising performance [4][5][6][7].

DNNs formulate monaural speech enhancement either as mask estimation [8] or end-to-end mapping [9]. In terms of mask estimation, DNNs usually take acoustic features in time-frequency (T-F) domain to estimate a T-F mask, such as ideal binary mask (IBM) [10], etc. In comparison, both the input and output of end-to-end speech enhancement DNNs can be T-F spectrograms, or even time domain signals directly without any feature engineering.

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).

In both mask estimation and end-to-end mapping DNNs, dilated convolution [11] serves a critical role to aggregate contextual information with the enlarged receptive field. Gated residual network (GRN) [12] employs dilated convolutions to accumulate context in temporal and frequency domains, leading to a better performance than a long short-term memory (LSTM) cell-based model [13]. In end-to-end setting, WaveNet [14] and its variations also adopt dilated convolution in speech enhancement.

For real-time systems deployed in resource-constrained environment, however, the oversized receptive field from dilated convolution can cause a severe delay issue. Although causal convolution can enable real-time speech denoising [15], it performs less well comparing to the dilated counterpart [12]. Besides, when the receptive field is too large, the amount of padded zeroes in the beginning of the sequence and a large buffer size for online processing can be a burdensome spatial complexity for a small device. Meanwhile, recurrent neural networks (RNN) can also aggregate context through a frame-by-frame processing without relying on the large receptive field. However, the responsiveness of a practical RNN system, such as LSTM [13], comes at the cost of the increased number of model parameters, which is neither as easy to train nor resource-efficient. There has been effort to apply dilated DenseNet [16] or a hybrid architecture to source separation [17], the mechanism to enable dual-staged context aggregate through the heterogeneous model topology has not been addressed.

To achieve efficient end-to-end monaural speech enhancement, we propose a densely connected convolutional and recurrent network (DCCRN), which conducts dual-level context aggregation. The first level of context aggregation in DCCRN is achieved by a dilated 1D convolutional neural network (CNN) component, encapsulated in the DenseNet architecture [18]. It is followed by a compact gated recurrent unit (GRU) component [19] to further utilize the contextual information in the "many-to-one" fashion. Note that we also employ a cross-component identical shortcut linking the output of DenseNet component to the output of GRU component to reduce the complexity of the GRU cells. We also propose a specifically designed training procedure for DCCRN that trains the CNN and RNN components separately, and then finetune the entire model. Experimental results show that the hybrid architecture of dilated DenseNet and GRU in DCCRN consistently outperforms other CNN variations with only one level of context aggregation on untrained speakers. Our model is computationally efficient and provides reasonable generalizability to untrained noises with only 1.38 million parameters.

We describe the proposed method in Section 2, and then provide experimental validation in Section 3. We conclude in Section 4.

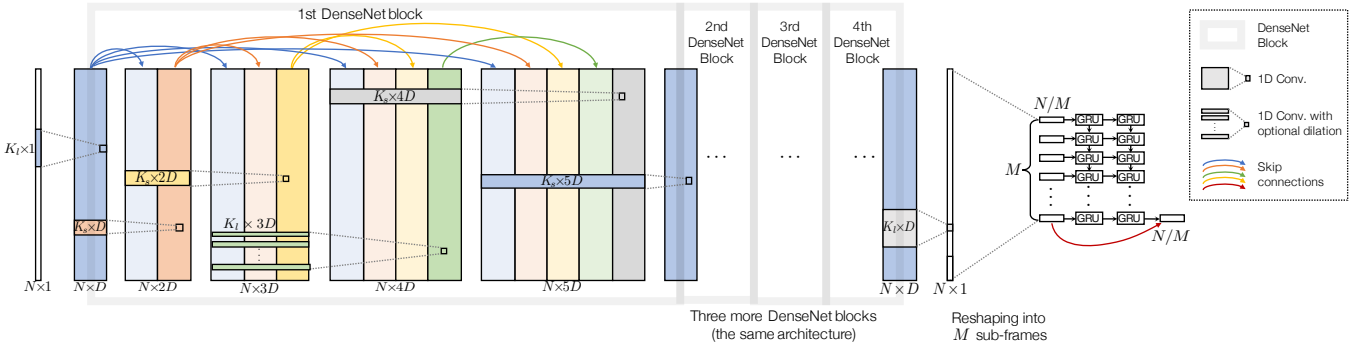


Fig. 1: A schematic diagram of the DCCRN training procedure including dilated DenseNet and GRU components.

2. MODEL DESCRIPTION

2.1. Context aggregation with dilated DenseNet

Residual learning has become a critical technique to tackle the gradient vanishing issue when tuning a deep convolutional neural network (CNN), such that the deep CNN can achieve better performance but with a lower model complexity. ResNet illustrates a classic way to enable residual learning by adding identical shortcuts across bottleneck structures [20]. Although the bottleneck structure includes direct paths to feedforward information from earlier layers to later layers, it does not extend to its full capacity of the information flow. Therefore, ResNet is usually found to be accompanied by a gating mechanism, a technique heavily used in RNNs, such as LSTM or GRU, to further facilitate the gradient propagation in convolutional networks [12].

In comparison, DenseNet [18] resolves the issue by redefining the skip connections. The dense block differs from the bottleneck structure in that each layer takes concatenated outputs from all preceding layers as its input, while its own output is fed to all subsequent layers (Figure 1). Consequently, DenseNet requires fewer model parameters to achieve a competitive performance.

In fully convolutional architectures, the dilated convolution is a popular technique to enlarge the receptive field to cover longer sequences [14], which has shown promising results in speech enhancement [12]. Because of the lower model complexity, dilated convolution is considered as a cheaper alternative to the recurrence operation. Our model adapts this technique sparingly with a receptive field size that does not exceed the frame size.

We use $\mathcal{F}^{(l)}$ to denote a convolution operation between the input $\boldsymbol{X}^{(l)}$ and the filter $\boldsymbol{H}^{(l)}$ in the l-th layer with a dilation rate $\gamma^{(l)}$:

$$\boldsymbol{X}^{(l+1)} \leftarrow \mathcal{F}^{(l)}(\boldsymbol{X}^{(l)}, \boldsymbol{H}^{(l)}, \gamma^{(l)}) \tag{1}$$

$$\boldsymbol{X}^{(l+1)}_{\tau,d} = \sum_{n+k\gamma^{(l)}=\tau} \boldsymbol{X}^{(l)}_{n,d} \boldsymbol{H}^{(l)}_{k,d}, \tag{2}$$

where n, τ, d, k are the indices of the input features, output features, channels, and filter coefficients, respectively. Note that k is an integer with a range $k \leq \lfloor K/2 \rfloor$, where K is the 1D kernel size. In our system we have two kernel sizes: $K_s = 5$ and $K_l = 55$. As DCCRN is based on 1D convolution, the tensors are always in the shape of (features)×(channels). Zero padding keeps the number of features the same across layers. Given the dilation rate $\gamma^{(l)} > 1$ [11], the convolution operation is defined in (2) with the dilation being activated.

In DCCRN, a dense block combines five such convolutional layers. In each block, the input to the l-th layer is a channel-wise concatenated tensor of all preceding feature maps in the same block, thus substituting (1) with

$$\boldsymbol{X}^{(l+1)} \leftarrow \mathcal{F}^{(l)}\Big(\big[\boldsymbol{X}^{(l)}, \boldsymbol{X}^{(l-1)}, \cdots, \boldsymbol{X}^{(l_b)}\big], \boldsymbol{H}^{(l)}, \gamma^{(l)}\Big), \tag{3}$$

where $\boldsymbol{X}^{(l_b)}$ denotes the first input feature map to the b-th block. Note that in this DenseNet architecture, $\boldsymbol{H}^{(l)}$ grows its depth accordingly, i.e., $\boldsymbol{H}^{(l)} \in \mathbb{R}^{K \times (l-l_b+1)D}$ with a growing rate D, the depth of $\boldsymbol{X}^{(l_b)}$. In the final layer of a block, the concatenated input channels collapse down to D, which forms the input to the next block. The first dense block in Figure 1 depicts this process. We stack four dense blocks with the dilation rate of the middle layer in each block to be 1, 2, 4 and 8, respectively. Different from the original DenseNet architecture, we do not apply any transition in-between blocks, except for the very first layer, prior to the stacked dense blocks, expanding the channel of the input from 1 to D, and another layer right after the stacked dense blocks to reduce it back to 1. This forms our fully convolutional DenseNet baseline. In all the convolutional layers, we use leaky ReLU as the activation.

2.2. Context aggregation with gated recurrent network

DCCRN further employs RNN layers following the dilated DenseNet component (Figure 1). Among LSTM and GRU, two most well-known RNN variations, DCCRN chooses GRU for its reduced computational complexity compared to LSTM. The information flow within each unit is outlined as follows:

$$\boldsymbol{h}(t) = \big(1 - \boldsymbol{z}(t)\big) \odot \boldsymbol{h}(t-1) + \boldsymbol{z}(t) \odot \tilde{\boldsymbol{h}}(t) \tag{4}$$

$$\tilde{\boldsymbol{h}}(t) = \tanh\Big(\boldsymbol{W}_h \boldsymbol{x}(t) + \boldsymbol{U}_h\big(\boldsymbol{r}(t) \odot \boldsymbol{h}(t-1)\big)\Big) \tag{5}$$

$$\boldsymbol{z}(t) = \sigma\big(\boldsymbol{W}_z \boldsymbol{x}(t) + \boldsymbol{U}_z \boldsymbol{h}(t-1)\big) \tag{6}$$

$$\boldsymbol{r}(t) = \sigma\big(\boldsymbol{W}_r \boldsymbol{x}(t) + \boldsymbol{U}_r \boldsymbol{h}(t-1)\big), \tag{7}$$

where t is the index in the sequence. $\boldsymbol{h}$ and $\tilde{\boldsymbol{h}}$ are the hidden state and the newly proposed one, which are mixed up by the update gate $\boldsymbol{z}$ in a complementary fashion as in (4). The GRU cell computes the tanh unit $\tilde{\boldsymbol{h}}$ by using a linear combination of the input $\boldsymbol{x}$ and the gated previous hidden state $\boldsymbol{h}$ as in (5). Similarly, the gates are estimated using another sigmoid units as in (6) and (7). In all linear operations, GRU uses corresponding weight matrices, $\boldsymbol{W}_h, \boldsymbol{W}_z, \boldsymbol{W}_r, \boldsymbol{U}_h, \boldsymbol{U}_z, \boldsymbol{U}_r$. We omit bias terms in the equations.

The GRU component in this work follows a "many-to-one" map-

Algorithm 1 The feedforward procedure in DCCRN

1: **Input:** N samples from the noisy utterance, $\boldsymbol{x}$
2: **Output:** The last M/N samples of the denoised signal, $\hat{\boldsymbol{s}}$
3: DenseNet denoising: $\boldsymbol{X}^{(L)} \leftarrow \mathcal{D}(\boldsymbol{x}; \mathbb{W}^{\text{CNN}})$
4: Reshaping: $\bar{\boldsymbol{X}}^{(L)} \leftarrow \big[\boldsymbol{X}^{(L)}_{1:N/M}, \boldsymbol{X}^{(L)}_{N/M+1:2N/M}, \cdots,$
5: $\qquad\qquad \boldsymbol{X}^{(L)}_{N-N/M+1:N}\big]$
6: GRU denoising: $\hat{\boldsymbol{s}} \leftarrow \mathcal{G}(\bar{\boldsymbol{X}}^{(L)}; \mathbb{W}^{\text{RNN}})$
7: Post windowing: $\hat{\boldsymbol{s}} \leftarrow \text{Hann}(\hat{\boldsymbol{s}})$ {# at test time only}

ping style for an additional level of context aggregation. During training, it looks back M time steps and generates the output corresponding to the last time step. To this end, DCCRN reshapes the output of the CNN part, the $N \times 1$ vector, into M sub-frames, each of which is an N/M-dimensional input vector to the GRU cell. We have two GRU layers, one with 32 hidden units and the other one with N/M units to match the output dimensionality of the system. Furthermore, to ease the optimization and to limit the model complexity of the GRU layers, we pass the last N/M sub-frame output of the DenseNet component to the output of GRU component via a skipping connection, which is additive as in the ResNet architecture—the denoised speech is the sum of the output from both components. With the dilated DenseNet component well-tuned, its output will already be close to the clean speech, which leaves less work for GRU to optimize, as detailed in Section 2.5.

2.3. Data flow

During training, as illustrated in Figure 1, the noisy frame is first fed to the DenseNet component $\mathcal{D}(\boldsymbol{x}; \mathbb{W}^{\text{CNN}})$ (line 3 in Algorithm 1). It comprises of L consecutive convolutional layers that are grouped into four dense blocks, where $\mathbb{W}^{\text{CNN}} = \{\boldsymbol{W}^{(1)}, \cdots, \boldsymbol{W}^{(L)}\}$. The output frame of DenseNet, containing N samples, is then reformulated to a sequence of N/M dimensional vectors, $\bar{\boldsymbol{X}}^{(L)} \in \mathbb{R}^{N/M \times M}$, which serve as the input of the GRU component: $\hat{\boldsymbol{s}} \leftarrow \mathcal{G}(\bar{\boldsymbol{X}}^{(L)}; \mathbb{W}^{\text{RNN}})$. The cleaned-up signal $\hat{\boldsymbol{s}}$ corresponds to the final state of the GRU with the dimension of N/M.

At test time, the output sub-frame of DCCRN is weighted by Hann window with 50% overlap by its adjacent sub-frames. Note that to generate the last N/M samples, DCCRN only relies on the current and past samples, up to N within that frame, without seeing future samples, which is similar to causal convolution. Hence, the delay of DCCRN is the sub-frame size ($256/16,000 = 0.016$ second). If it were just for the DenseNet component only, such as those convolutional baselines compared in Section 3, the Hann window with the same overlap rate would still be applied, but the model output would be all N samples for the corresponding frame, instead of the last N/M samples.

Table 1 summarizes the network architecture. The current topology is designed for speech sampled at 16kHz.

2.4. Objective function

It is known that the mean squared error (MSE) itself cannot directly measure the perceptual quality nor the intelligibility, both of which are usually the actual metrics for evaluation. To address the discrepancy, the MSE can be replaced by a more intelligibly salient measure, such as short-time objective intelligibility (STOI) [21]. However, the improved intelligibility does not guarantee a better perceptual quality. The objective function in this work is defined in (8),

Table 1: Architecture of DCCRN: for the CNN layers, the data tensor sizes are represented by (size in samples, channels), while the CNN kernel shape is (size in samples, input channels, output channels). For the GRU layers, an additional dimension for the data tensors defines the length of the sequence, $M = 4$, while the kernel sizes define the linear operations (input features, output features). The middle layer of each dense block, marked by a dagger, is with larger kernel size $K_l = 55$ and an optional dilation with the rate of 1, 2, 4, and 8, for the four dense blocks, respectively.

Components	Input shape	Kernel shape	Output shape
Change channel	(1024, 1)	(55, 1, 32)	(1024, 32)
DenseNet	(1024, 32)	(5, 32, 32) (5, 64, 32) (55, 96, 32)† (5, 128, 32) (5, 160, 32) $\times 4$	(1024, 32)
Change channel	(1024, 32)	(55, 32, 1)	(1024, 1)
Reshape	(1024, 1)	-	(4, 256, 1)
GRU	(4, 256, 1)	(256+32, 32)×3 (32+256, 256)×3	(256, 1)

which is still based on MSE, but accompanied by a regularizer that compares mel spectra between the target and output signals. The TF domain regularizer compensates the end-to-end DNN that would only operate in time domain, otherwise. Empirically, it is shown to achieve better perceptual quality, as proposed in [22].

$$\mathcal{E}(\boldsymbol{s}||\hat{\boldsymbol{s}}) = \text{MSE}(\boldsymbol{s}||\hat{\boldsymbol{s}}) + \lambda \text{MSE}\big(\text{Mel}(\boldsymbol{s})||\text{Mel}(\hat{\boldsymbol{s}})\big). \tag{8}$$

2.5. Model training scheme

We train the CNN and RNN components separately, and then finetune the combined network.

- ***CNN training:*** First, we train the CNN component to minimize the error $\arg\min_{\mathbb{W}^{\text{CNN}}} \mathcal{E}(\boldsymbol{y}||\mathcal{D}(\boldsymbol{x}; \mathbb{W}^{\text{CNN}}))$.
- ***RNN training:*** Next, we train the RNN part by minimizing $\arg\min_{\mathbb{W}^{\text{RNN}}} \mathcal{E}(\boldsymbol{s}||\mathcal{G}(\bar{\boldsymbol{X}}^{(L)}; \mathbb{W}^{\text{RNN}}))$, while $\mathbb{W}^{\text{CNN}}$ is locked.
- ***Integrative finetuning:*** Once both CNN and RNN components are pretrained, we unlock the CNN parameter and finetune both components to minimize the final error: $\arg\min_{\mathbb{W}^{\text{CNN}}, \mathbb{W}^{\text{RNN}}} \mathcal{E}(\boldsymbol{s}||\mathcal{G}\Big(\mathcal{D}\big(\boldsymbol{x}; \mathbb{W}^{\text{CNN}}\big); \mathbb{W}^{\text{RNN}}\Big)$. Note that the learning rate for integrative finetuning should be smaller.

3. EXPERIMENTS

3.1. Experimental setup

In this paper, the experiment runs on TIMIT corpus [23]. We consider two experimental settings. For the model training, we randomly select 1000 utterances from TIMIT training subset. 5 types of non-stationary noise (birds, cicadas, computer keyboard, machine guns and motorcycles) from [24] are used to create mixtures. Concretely, each clean signal is mixed with a random cut of each of these noise types at a SNR level randomly drawn from the set of integers with the range of $[-5, +5]$ dB. Therefore, 5,000 noisy speech samples, totaling 3.5 hours, are used for model training. At test time, we randomly select 100 unseen utterances from TIMIT test subset,

SUB-8-BIT QUANTIZATION FOR ON-DEVICE SPEECH RECOGNITION: A REGULARIZATION-FREE APPROACH

Kai Zhen, Martin Radfar, Hieu Nguyen, Grant P. Strimel, Nathan Susanj, Athanasios Mouchtaris

Amazon Alexa AI

ABSTRACT

For on-device automatic speech recognition (ASR), quantization aware training (QAT) is ubiquitous to achieve the trade-off between model predictive performance and efficiency. Among existing QAT methods, one major drawback is that the quantization centroids have to be predetermined and fixed. To overcome this limitation, we introduce a regularization-free, "soft-to-hard" compression mechanism with self-adjustable centroids in a μ-Law constrained space, resulting in a simpler yet more versatile quantization scheme, called General Quantizer (GQ). We apply GQ to ASR tasks using Recurrent Neural Network Transducer (RNN-T) and Conformer architectures on both LibriSpeech and de-identified far-field datasets. Without accuracy degradation, GQ can compress both RNN-T and Conformer into sub-8-bit, and for some RNN-T layers, to 1-bit for fast and accurate inference. We observe a 30.73% memory footprint saving and 31.75% user-perceived latency reduction compared to 8-bit QAT via physical device benchmarking.

***Index Terms*—** On-device speech recognition, quantization aware training, RNN-T, conformer, model efficiency

1. INTRODUCTION

Improving the efficiency of neural automatic speech recognition (ASR) models via quantization is critical for on-device deployment scenarios. For neural network accelerator (NNA) embedded devices, where memory and bandwidth are at a premium, quantization can reduce the footprint and lower the bandwidth consumption of ASR execution, which will not only afford a faster model inference but also facilitate model deployment to various portable devices where a stable network connection is limited.

Existing quantization methods can be post-training quantization (PTQ) or in-training / quantization aware training (QAT). PTQ is applied after the model training is complete by compressing models into 8-bit representations and is relatively well supported by various libraries [1, 2, 3, 4, 5, 6], such as TensorFlow Lite [7] and AIMET [8] for on-device deployment. However, almost no existing PTQ supports customized quantization configurations to compress machine learning (ML) layers and kernels into sub-8-bit (S8B) regimes [9]. Moreover, the performance drop is inevitable as the model is unaware of the loss of precision when being quantized at test time. In contrast, QAT performs bit-depth reduction of model weights (for example, from 32-bit floating point to 8-bit integer) during training which usually yields superior performance over PTQ [10][11]. The QAT mechanism can be in the forward pass (FP-QAT) or the backward pass (BP-QAT), with the difference being whether regularization is used in the loss function. FP-QAT [9] quantizes the model weights during forward propagation to pre-defined quantization centroids. BP-QAT [12, 13, 14] relies on customized regularizers to gradually force weights to those quantization centroids (i.e., "soft quantization" via gradient) during training before hard compression performs in the late training phase. As model weights are informed by the customized regularizers to move closer to where they are quantized at runtime per training step, the predictive performance is often well preserved. Therefore, the focus of this work is on QAT.

Under both FP- and BP-QAT, it is essential that the quantization centroids are defined and specified before model training. As such, the demerit is the low feasibility when quantizing models in S8B mode because one needs to select the proper quantization centroids and their configurations for each kernel in each layer to ensure minimal runtime performance degradation. Consequently, applying existing QAT methods to Conformer [15] becomes quite challenging, as it usually contains more than hundreds of kernels.

In this work, we propose General Quantizer (GQ), a regularization-free, model-agnostic quantization scheme with a mixed flavor of both FP- and BP-QAT. GQ is "general" in that it does not augment the objective function by introducing any regularizer as in BP-QAT but determines the appropriate quantization centroids during model training for a given bit depth, and it can be simply applied in a plug-and-play manner to an arbitrary ASR model. Unlike FP-QAT, GQ features a soft-to-hard quantization during training, allowing model weights to hop around adjacent partitions more easily. Under GQ, quantization centroids are self-adjustable but in a μ-Law constrained space. As a proof-of-concept, we adopt the ASR task and conduct experiments on both the LibriSpeech and de-identified far-field datasets to evaluate GQ on three major end-to-end ASR architectures, namely conventional Recurrent Neural Network Transducer (RNN-

978-1-6654-7189-3/22/$31.00

T) [16], Bifocal RNN-T [17], and Conformer [18][19]. Our results show that in all three architectures, GQ yields little to no accuracy loss when compressing models to S8B or even sub-5-bit (5-bit or lower). We also present performance optimization strategies from ablation studies on bit-allocation and quantization frequency. Our contributions are as follows:

- We propose GQ, inspired by both FP- and BP-QAT approaches. GQ enables on-centroid weight aggregation without augmented regularizers. Instead, it leverages Softmax annealing to impose soft-to-hard quantization on centroids from the μ-Law constrained space.
- GQ supports different quantization modes for a wide range of granularity: different bit depths can be specified for different kernels/layers/modules.
- With GQ, we losslessly compress a lightweight streaming Conformer into sub-5-bit with more than $6\times$ model size reduction. To our best knowledge, this is among the first sub-5-bit Conformer models for on-device ASR. Without accuracy degradation, our GQ-compressed 5-bit Bifocal RNN-T reduces the memory footprint by 30.73% and P90 user-perceived latency (UPL) by 31.30%.

We describe the problem in Sec. 2 and GQ in Sec. 3. The experimental settings and results are detailed in Sec. 4. We conclude in Sec. 5 with some final remarks.

2. PRELIMINARIES

2.1. Problem Formulation

Consider a general deep neural network architecture with K layers, $\mathcal{F} = \mathcal{F}_1 \circ \cdots \circ \mathcal{F}_K$, mapping the input from $\mathbb{R}^{d_1}$ to the output in $\mathbb{R}^{d_{K+1}}$ as $\mathcal{F} : \mathbb{R}^{d_1} \longmapsto \mathbb{R}^{d_{K+1}}$, where the input and output of an arbitrary k-th layer are $\boldsymbol{x}^{(k+1)} := \mathcal{F}_k(\boldsymbol{x}^{(k)})$. Under supervised learning, the training data $\mathcal{X} = \{\boldsymbol{x_1}, \ldots, \boldsymbol{x}_N\}$ and $\mathcal{Y} = \{\boldsymbol{y_1}, \ldots, \boldsymbol{y}_N\}$ are used for updating model weights $\mathbb{W} = [\boldsymbol{W}_1, \ldots, \boldsymbol{W}_K]$ for K layers in $\mathcal{F}$. Usually the optimization process is over the training objective function $\mathcal{L}(\mathcal{X}, \mathcal{Y}, \mathcal{F}, \mathbb{W}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\mathcal{F}(\boldsymbol{x_i}), \boldsymbol{y_i}) + \lambda R(\mathbb{W})$, where i is the data batch index, ℓ is the major loss term measuring model accuracy and $R(\mathbb{W})$ is the regularizer blended to the objective function via a coefficient λ.

Network quantization aims at discretizing model weights. For scalar quantization, it is to convert each weight, $w \in \mathbb{W}$, to a quantization centroid, $z \in \boldsymbol{z}$, where $\boldsymbol{z} = [z_1, ...z_m]$ to ensure the network is compressed into $\lceil log_2 m \rceil$-bit. For S8B quantization, centroids are from a subset of `INT8` values.

2.2. Related QAT Approaches

BP-QAT counters model weight continuity via regularization. For example, it introduces weight regularizers on model weights, i.e., $R(\mathbb{W})$, measuring the point-wise distance between each weight and m quantization centroids in the centroid vector $\boldsymbol{z} = [z_1, ...z_m]$. Note that the quantization weight regularizer in the loss function, as $R(\mathbb{W})$, must be gradient descent compatible. Consequently, $R(\mathbb{W})$ cannot enforce each weight to be replaced by the closest centroid in $\boldsymbol{z}$ as $w = \arg\min_i ||w - z_i||$, for $w \in \mathbb{W}$ and $z_i \in \boldsymbol{z}$, because the $\min$ operator is not differentiable. Recent BP-QAT methods force weights to approach the centroid in $\boldsymbol{z}$ using $R(\mathbb{W}) = \sum_{w \in \mathbb{W}} \mathcal{D}(w, \boldsymbol{z})$, where the differentiable dissimilarity function $\mathcal{D}$ is based on a cosine function in [12, 13].

In contrast, FP-QAT can be regularizer free [9, 20]. Usually, the process is to use a "fake quantizer" or equivalent operations during training, hard quantizing weights to a specific range and bit-depth; and then at runtime, converting the model to `INT8` format via TFLite [21]. The study [9] uses native quantization operators with which, during training, the weights are quantized and then converted to the integer type for model deployment. However, FP-QAT is essentially hard compression recurring during training with severely dropped performance when applied to S8B quantization. Consequently, finetuning is usually needed, which prolongs the model training time [22, 23].

Both FP-QAT and BP-QAT require specifying appropriate quantization centroids before model training. While the centroids for `INT8` model compression are pre-defined, for S8B quantization, the optimal set of centroids is usually kernel/layer specific. For models, such as Conformer [15], where there are usually hundreds of kernels, current S8B QAT methods become less tractable.

In this work, we combine the merit from both FP- and BP-QAT and propose General Quantizer (GQ) that navigates weights to the corresponding quantization centroids without introducing augmented regularizers but via feedforward-only operators. Our work is inspired by a continuous relaxation of quantization [24] also used for speech representation learning [25, 26, 27, 28, 29], and μ-Law algorithm for 8-bit pulse-code modulation (PCM) digital telecommunication [30].

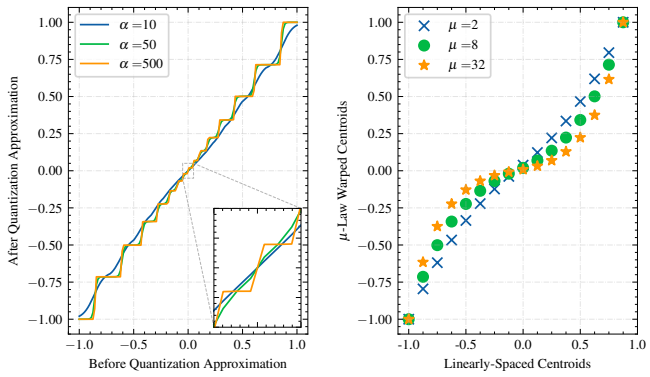
3. METHODS

3.1. Centroid Selection via Softmax-Based Dissimilarity Matrices

For any weight value $w_i \in \boldsymbol{w}$ where $|\boldsymbol{w}| = n$, and the quantization centroid vector $\boldsymbol{z} = [z_1, ...z_m]$, we define the point-wise dissimilarity matrix in Eq. 1

$$\boldsymbol{A}_{\text{soft}} = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}, \tag{1}$$

where a_{ij} is the probability of representing w_i by z_j. Each row in $\boldsymbol{A}_{\text{soft}}[i\cdot]$ is summed to 1 with the largest probability



(a) Weight value before and after quantization approximation. Here, the larger the softmax temperature α is, the closer to true quantization that the weight transformation becomes.

(b) Relationship between the wrapped quantization centroids and μ. A large μ means more quantization centroids are allocated near 0 and vice versa.

Fig. 1: Hyperparameters in GQ: α adjusts the quantization temperature which increases gradually during training, and μ modulates the non-linearity of quantization centroids.

going to the closest centroid. This is achieved when the pointwise distance $||w_i - z_j||_1$ is scaled by a negative number $-\alpha$ and wrapped by a Softmax function in Eq. 2

$$a_{ij} = \frac{e^{-\alpha||w_i - z_j||_1}}{\sum_{j=1}^{m} e^{-\alpha||w_i - z_j||_1}}. \tag{2}$$

Here, $\alpha \in [1, \infty)$ serves as the Softmax temperature for quantization annealing. When α is relatively small, w_i will be approximated by all centroids in $\boldsymbol{z}$ (see Eq.3); when $\alpha \to \infty$, $\boldsymbol{A}_{\text{soft}}[i\cdot]$ becomes a one-hot vector $\boldsymbol{A}_{\text{hard}}[i\cdot]$ and the weight will be the closest centroid.

$$\overline{w_i} = \boldsymbol{z} \times \boldsymbol{A}_{\text{soft}}[i\cdot]^T. \tag{3}$$

For simplicity, during training, we set the initial and target scalar values to be α_{start} and α_{end}, and allow α to gradually and linearly increase from s_{start} to s_{end}, as shown in Eq. 4.

$$\alpha = \alpha_{\text{start}} + (s - s_{\text{start}}) \times \frac{\alpha_{\text{end}} - \alpha_{\text{start}}}{s_{\text{end}} - s_{\text{start}}}. \tag{4}$$

As a result, the QAT effect is gradually intensified. At $\alpha = 10$, a rather small value, weights after being approximated by quantization centroids in $\boldsymbol{z}$ roughly preserve their original values; however, as α gradually increased to 500, the near-linear line almost becomes a step function, aggregating weights to just a few centroids (see Fig. 1 (a)). This forms a soft-to-hard QAT and allows model weights to be updated via gradients with barely any extra constraint during the early stage of training before driving weights to a certain centroid.

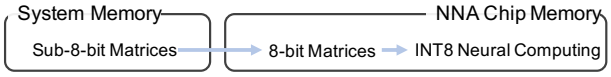


Fig. 2: Loading weights from system memory to chip memory on NNA is faster in S8B format, although the arithmetic operation is still in `INT8` format.

3.2. Adjusting Centroids with μ-Law Expanding

We assume weight distribution symmetry from any kernel in a trained 32-bit neural network in which the absolute values of most weights are small. Consequently, the imposed m quantization centroids in $\boldsymbol{z}$ should also be symmetric ($|z_i| = |z_{m+1-i}|$ where $i \in [1, m]$) with most centroids close to 0. To specify and adjust the level of non-linearity of $\boldsymbol{z}$ per kernel during training, we resort to μ-Law algorithm, mainly used in 8-bit PCM telecommunication (similar to A-Law algorithm standardized in Europe). The motivation for using μ-Law function is that it accents samplings from small (soft) values, reducing the quantization error and increasing signal-to-quantization-noise-error (SQNR) for data transmission. Hence, we employ the μ-Law algorithm in GQ to improve the quantization robustness of ASR models.

In μ-Law expanding function (Eq. 5), $\boldsymbol{z}$, linearly spaced values within the range of -1 and 1, are warped as $\boldsymbol{z'}$ in which the values are driven closer to 0, except for the boundary poles. As shown in Fig. 1 (b), when μ increases, the linearly spaced values are more noticeably warped in the μ-Law transformed space: by adjusting the value of μ that minimizes the quantization error $\arg\min_{\mu} ||\boldsymbol{z'} \times \boldsymbol{A}_{\text{soft}} - \boldsymbol{w}||_2$, quantization centroids in $\boldsymbol{z}$ can be re-distributed to better reflect the dynamic weight range of a specific neural component. A larger μ means the weight distribution is concentrated near 0; therefore, we allocate more quantization centroids near the origin. Smaller μ values indicate the weight distribution is tail-heavy.

$$\boldsymbol{z'} = \text{sng}(\boldsymbol{z}) \left(\frac{(1+\mu)^{|\boldsymbol{z}|} - 1}{1 + \mu} \right). \tag{5}$$

3.3. S8B Model for 8-Bit Computing

Due to limited chip memory size and bandwidth of the NNA, weights are loaded from system memory to the chip memory per matrix, which is time consuming. Hence, compressing the model into S8B can achieve inference speedup, even though NNA uses `INT8` for neural computing (see Fig.2).

Nonetheless, we map $\boldsymbol{z'}$ in Eq. 5 to the closest value in $[\cdots k/128 \cdots]$, where the integer $k \in [-128, 127]$, such that in-training and runtime quantization centroids are consistent.

3.4. Callback "Is All You Need"

A callback is a set of functions to be invoked at certain training stages. Under GQ, the callback is all you need: For any