INCREMENTAL BINARIZATION ON RECURRENT NEURAL NETWORKS FOR SINGLE-CHANNEL SOURCE SEPARATION

Sunwoo Kim, Mrinmoy Maity, Minje Kim

Indiana University
Department of Intelligent Systems Engineering
Bloomington, IN 47408
kimsunw@indiana.edu, mmaity@iu.edu, minje@indiana.edu

ABSTRACT

This paper proposes a Bitwise Gated Recurrent Unit (BGRU) network for the single-channel source separation task. Recurrent Neural Networks (RNN) require several sets of weights within its cells, which significantly increases the computational cost compared to the fully-connected networks. To mitigate this increased computation, we focus on the GRU cells and quantize the feedforward procedure with binarized values and bitwise operations. The BGRU network is trained in two stages. The real-valued weights are pretrained and transferred to the bitwise network, which are then incrementally binarized to minimize the potential loss that can occur from a sudden introduction of quantization. As the proposed binarization technique turns only a few randomly chosen parameters into their binary versions, it gives the network training procedure a chance to gently adapt to the partly quantized version of the network. It eventually achieves the full binarization by incrementally increasing the amount of binarization over the iterations. Our experiments show that the proposed BGRU method produces source separation results greater than that of a real-valued fully connected network, with 11-12 dB mean Signal-to-Distortion Ratio (SDR). A fully binarized BGRU still outperforms a Bitwise Neural Network (BNN) by 1-2 dB even with less number of layers.

Index Terms— Speech Enhancement, Recurrent Neural Networks, Gated Recurrent Units, Bitwise Neural Networks

1. INTRODUCTION

Neural network-based approaches to source separation tasks have been becoming more prevalent [1, 2, 3]. Fully connected deep neural networks (DNN) have shown to be capable of learning complex mapping functions from a large set of noisy signals and their corresponding ideal binary mask (IBM) target outputs [4, 5, 6]. Recurrent neural networks (RNN), which are structured to be more effective in applications involving sequential or temporal data, have also shown to excel in the same task [7, 8, 9, 10, 11]. The RNN is able to attain the superior performance by utilizing a shared hidden state and gates within its hidden cells that guide the memory and learning over a sequence of inputs [12]. The most practical method to train RNNs is with truncated Backpropagation Through Time (BPTT) [13]. This bounded-history approximation method simplifies computation by limiting itself to a fixed scope of T timesteps [14].

The most efficient cell structure that is robust to the gradient vanishing problem is the Gated Recurrent Unit (GRU) cell [15]. The computation within each GRU cell is:

$$\begin{aligned}
\mathbf{r}^{(l)}(t) &= \sigma\Big(\mathbf{W}_r^{(l)}\mathbf{x}^{(l-1)}(t) + \mathbf{U}_r^{(l)}\mathbf{h}^{(l)}(t-1)\Big) \\
\mathbf{z}^{(l)}(t) &= \sigma\Big(\mathbf{W}_z^{(l)}\mathbf{x}^{(l-1)}(t) + \mathbf{U}_z^{(l)}\mathbf{h}(t-1)\Big) \\
\tilde{\mathbf{h}}^{(l)}(t) &= \phi\Big(\mathbf{W}_h^{(l)}\mathbf{x}^{(l-1)}(t) + \mathbf{U}_h^{(l)}\big(\mathbf{r}^{(l)}(t) \odot \mathbf{h}^{(l)}(t-1)\big)\Big) \\
\mathbf{h}^{(l)}(t) &= \mathbf{z}^{(l)}(t) \odot \mathbf{h}^{(l)}(t-1) + (1 - \mathbf{z}^{(l)}(t)) \odot \tilde{\mathbf{h}}^{(l)}(t)
\end{aligned} \tag{1}$$

where $l = \{1, \ldots, L+1\}$ denotes the layer index and $t = \{1, \ldots, T\}$ is the time index. $\mathbf{r}_t, \mathbf{z}_t, \tilde{\mathbf{h}}_t$, and $\mathbf{h}_t$ are reset gate, update gate, candidate hidden state, and updated hidden state respectively all of dimension $\mathbb{R}^{K^{(l)}}$ with $K^{(l)}$ as the number of units at layer l. $\mathbf{W}_r^{(l)} \in \mathbb{R}^{K^{(l)} \times K^{(l-1)}}$ and $\mathbf{U}_r^{(l)} \in \mathbb{R}^{K^{(l)} \times K^{(l)}}$ are the weight matrices for the input $\mathbf{x}^{(l)}(t)$ and previous hidden state $\mathbf{h}^{(l)}(t-1)$ at the reset gate. Similarly, $\mathbf{W}_z$, $\mathbf{U}_z$, $\mathbf{W}_h$, and $\mathbf{U}_h$ are corresponding weights for the update gate and candidate state. The σ and ϕ refer to the logistic sigmoid and hyperbolic tangent activation functions. The bias term is omitted for simplicity. Note that $\mathbf{h}^{(l)}(t)$ is fed to the next layer as an input, $\mathbf{x}^{(l)}(t)$.

For a single feedforward step, the RNN requires multiple sets of weights and performs operations in (1) for T timesteps. With deeper RNNs, the computational cost rises rapidly in terms of K and L. This paper presents an efficient method to reduce the computational and spatial complexity of the GRU network for the source separation problem while maintaining high performance results. We extend from the idea of Bitwise Neural Networks (BNN) [16] [17] and low-precision RNNs [18]. The model we propose is a Bitwise GRU (BGRU) network that reduces network complexity by re-defining the originally real-valued inputs and outputs, weights, and operations in a bitwise fashion. By limiting the network to bipolar binary values, the space complexity of the network can be significantly reduced. In addition, all real-valued operations during the feedforward procedure can be replaced with bitwise logic, which further reduces both spatial and time complexity [19, 20, 21, 22].

Transforming real-valued weights into bipolar binaries results in heavy quantization loss [23, 24]. To alleviate this effect, the weights are converted into binary values through a gentle training procedure. In this paper, we introduce an incremental training method for weights of the BGRU network that holds onto the quality of the source separation model. Experimental results for single-channel source separation tasks show that the BGRU model shows incremental and predictable loss depending on the amount of binarization and still performs better than a real-valued Fully-Connected Network (FCN).

This project was supported by Intel Corporation.

2. BITWISE GATED RECURRENT UNITS (BGRU)

2.1. Background: Bitwise Neural Networks

Binarization has been explored as a method of network compression. BinaryConnect [17], binarized neural networks [19], trained ternary quantization [25], and Bitwise Neural Networks (BNN) [26] have implemented a binarized or ternarized neural network in bipolar binaries (with zeros in the ternarized case) for network compression. They emphasize that replacing real-valued operations with bitwise versions greatly reduces the network's complexity. In particular, the BNN training process is assisted by initializing the binarized network with pretrained weights. The weights are compressed in a real-valued network with the hyperbolic tangent activation function in order to better approximate their binary versions. Further quantization is performed in the BNN, where the inputs are quantized using Quantization-and-Disperson, which uses Lloyd-Max's quantization to convert each frequency magnitude of the noisy input spectrum into 4 bits with bipolar binary features [27]. In the domain of source separation, BNN's have been applied by predicting Ideal Binary Masks (IBM) as target outputs [16].

While the BNN significantly reduces the space and time complexity of the network, the conversion from real-values to bipolar binaries inevitably produces quantization error. One method to reduce this penalty is the concept of sparsity [16]. Sparsity can be introduced to bitwise networks by converting the pretrained weights with smaller values to 0's. The threshold for determining the sparsity is calculated with a predefined boundary β. The relaxed quantization process for a weight element w is:

$$\bar{w} = \begin{cases} +1 & \text{if } w > \beta \\ -1 & \text{if } w < -\beta \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $\bar{w}$ represents the binarized variable. Another way to mitigate the quantization error is by multiplying a scaling factor μ to the bipolar-binarized weights, so that the quantized values approximate the original values more closely [25].

2.2. Feedforward in BGRU

2.2.1. Notation and setup

For the following sections of the paper, we specify discrete variables with a bar notation, i.e. $\bar{x}$. Depending on the context, this could be a binary variable with 0 and 1 (e.g. gates), a bipolar binary variable with $+1$ and -1 (e.g. binarized hidden units), or a ternary variable (e.g. sparse bipolar binary weights). The binary versions of logistic sigmoid and hyperbolic tangent activation functions are:

$$\bar{\sigma}(x) = \frac{sgn(x)+1}{2} \in \{0,1\},\ \bar{\phi}(x) = sgn(x) \in \{-1,+1\} \tag{3}$$

respectively where $sgn(x)$ is a sign function [19, 26].

2.2.2. The feedforward procedure

In the BGRU, the feedforward process is defined as follows:

$$\begin{aligned} \bar{\mathbf{r}}^{(l)}(t) &= \bar{\sigma}\Big(\bar{\mathbf{W}}_r^{(l)}\bar{\mathbf{x}}^{(l-1)}(t) + \bar{\mathbf{U}}_r^{(l)}\bar{\mathbf{h}}^{(l)}(t-1)\Big) \\ \bar{\mathbf{z}}^{(l)}(t) &= \bar{\sigma}\Big(\bar{\mathbf{W}}_z^{(l)}\bar{\mathbf{x}}^{(l-1)}(t) + \bar{\mathbf{U}}_z^{(l)}\bar{\mathbf{h}}^{(l)}(t-1)\Big) \\ \bar{\tilde{\mathbf{h}}}^{(l)}(t) &= \bar{\phi}\Big(\bar{\mathbf{W}}_h^{(l)}\bar{\mathbf{x}}^{(l-1)}(t) + \bar{\mathbf{U}}_h^{(l)}\big(\bar{\mathbf{r}}^{(l)}(t) \odot \bar{\mathbf{h}}^{(l)}(t-1)\big)\Big) \\ \bar{\mathbf{h}}^{(l)}(t) &= \bar{\mathbf{z}}^{(l)}(t) \odot \bar{\mathbf{h}}^{(l)}(t-1) + (1 - \bar{\mathbf{z}}^{(l)}(t)) \odot \bar{\tilde{\mathbf{h}}}^{(l)}(t) \end{aligned} \tag{4}$$

The product between two binarized values (e.g. between the (i,j)-th element of $\bar{\mathbf{W}}_r^{(l)}$ and the j-th element of $\bar{\mathbf{x}}^{(l-1)}(t)$) is equivalent to the XNOR operations, a cheaper binary operation than the corresponding floating-point operation. Also, the use of sign functions $\bar{\phi}$ and a hard step function $\bar{\sigma}$ in place of the hyperbolic tangent and sigmoid functions also expedite the process because they can be usually implemented by a pop counter.

2.2.3. Scaled sparsity and Bernoulli masks

We define two types of masks that are applied on various parts of the network. The scaled sparsity mask is a two-in-one solution to introduce both scaling parameters and sparsity into weights during the binarization process. To binarize the weight matrices the scaled sparsity mask $\mathbf{B}$ is created using a predefined sparsity parameter $0 < \rho < 1$. First, we find a per-layer cutoff value β and the scaling parameter μ that meet the following equations:

$$\begin{aligned} &\mathcal{S} = \{(i,j) : |W_{i,j}^{(l)}| > \beta\}, \quad |\mathcal{S}| = K^{(l-1)}K^{(l)}\rho \\ &\mu = \frac{1}{|\mathcal{S}|}\sum_{(i,j)\in\mathcal{S}} |W_{i,j}^{(l)}|, \end{aligned} \tag{5}$$

where $\mathcal{S}$ is the set of weight indices whose absolute values are larger than the cutoff value and $|\mathcal{S}|$ denotes the number of such weights. Therefore, for a given sparsity value ρ, we first sort the weights in their absolute values and then find the cutoff that results in $\mathcal{S}$ with the predefined size. Using β and μ, we set the mask elements as follows:

$$B_{i,j} = \begin{cases} \mu & \text{if } |W_{i,j}| > \beta \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The other type of mask is a random Bernoulli matrix $\mathbf{C}$ with a parameter $0 < \pi < 1$ as the amount of binarization. The value of π is initially chosen as a small value (e.g. 0.1 for 10% binarization) and gradually increased up to 1.0, which means the network is completely binarized. The created masks are applied on weights $\mathbf{W}$ to create the partly binarized matrix $\widehat{\mathbf{W}}$:

$$\widehat{\mathbf{W}} = \big(\bar{\phi}(\mathbf{W}) \odot \mathbf{B}\big) \odot \mathbf{C} + \phi(\mathbf{W}) \odot (1 - \mathbf{C}). \tag{7}$$

The purpose of $\mathbf{B}$ with μ values is to lessen the quantization error from the binarization. The $\bar{\phi}$ operator will transform all values into bipolar binary values, which would be too intensive of a tranformation because the distribution of the first round weights are all relatively close to 0. Thus, by multiplying the remaining nonzero bipolar values after applying sparsity with μ, the values are scaled down to the average value of the non-sparse portion, which is a better representative for the nonzero elements. Note that feedforward is still bitwise thanks to the symmetry of $\mathbf{B}$ and by skipping zeros.

The Bernoulli mask $\mathbf{C}$ enables a gradual transition from real-valued weights and operations to bitwise versions. This mask is applied on the bitwise and real-valued elements in a complementary way to control the proportion of binarization in the network. $\widehat{\mathbf{W}}$ in (7) is binarized only partly with the proportion set by π. Note that for the real-valued weights we are using a $\tanh$ compressed version $\phi(\mathbf{W})$ for the purpose of regularization (see Section 2.3.1 for more details).

$\mathbf{C}$ is used to control the binarization of the other network elements such as gates and hidden units, too. For the candidate hidden units $\tilde{\mathbf{h}}$, for example, the activations are performed as:

$$\widehat{\tilde{\mathbf{h}}} = \bar{\tilde{\mathbf{h}}} \odot \mathbf{C} + \tilde{\mathbf{h}} \odot (1 - \mathbf{C}), \tag{8}$$

The gates are also partially binarized in this way. The mask $\mathbf{C}$ is generated at each iteration for the weights as in (7) and then at each timestep for the activation functions of GRU cells as in (8). This ensures that the gradients of the bitwise terms are evenly distributed gradually for all weights at each levels of π. Without even distribution, certain elements of the graph that do not participate in the bitwise procedure begin focusing on compensating for the quantization loss from the other bitwise elements. This needs to be avoided since as π is increased to 1.0 these elements need to be quantized eventually.

2.3. Training BGRU Networks

The objective is to accept binarized mixture signal inputs and predict the corresponding IBMs. The inputs are binarized using the Quantization-and-Dispersion technique [26], and the target outputs are bipolar binary IBM predictions which are later converted to 0's and 1's for the source separation task. We follow the typical two-round training scheme from BNNs, too.

2.3.1. First round: Pretraining ϕ-compressed weights

The GRU network is first initialized with real-valued weights and then trained on quantized binary inputs. During training, the weights are wrapped with the hyperbolic tangent activation function, ϕ. This has the effect of bounding the range of weights between -1 and $+1$ as well as regularization. In the second round, the sign function, $\bar{\phi}$ is applied on the weights instead, hence the first round network can be perceived as its softer version. For example, the feedforward procedure in (1) for only the hidden candidate state at layer l and timestep t becomes:

$$\tilde{\mathbf{h}}^{(l)}(t) = \phi\Big(\phi\big(\mathbf{W}_h^{(l)}\big)\mathbf{x}^{(l-1)}(t) + \phi\big(\mathbf{U}_h^{(l)}\big)\big(\bar{\mathbf{r}}^{(l)}(t) \odot \bar{\mathbf{h}}^{(l)}(t-1)\big)\Big) \quad (9)$$

The ϕ-compressed weights are applied similarly for the reset and update gates.

Backpropagation: With the introduction of ϕ on the weight matrices, the derivative with respect to ϕ is added onto the backpropagation due to the chain rule. For example, the gradients for (9) are computed as:

$$\begin{aligned}
\boldsymbol{\delta}_{\tilde{\mathbf{h}}}(t) &= \boldsymbol{\delta}^{(l)}(t) \odot (1 - \mathbf{z}(t)) \odot \big(1 - \tilde{\mathbf{h}}^{(l)}(t)^2\big) \\
\nabla \mathbf{W}_h^{(l)} &= \Big(\sum_{t=0}^{T} \boldsymbol{\delta}_{\tilde{\mathbf{h}}}(t) \cdot \big(\mathbf{x}^{(l-1)}(t)\big)^\top\Big) \odot \Big(1 - \phi^2\big(\mathbf{W}_h^{(l)}\big)\Big) \quad (10) \\
\nabla \mathbf{U}_h^{(l)} &= \Big(\sum_{t=1}^{T} \boldsymbol{\delta}_{\tilde{\mathbf{h}}}(t) \cdot \big(\bar{\mathbf{r}}^{(l)}(t) \odot \bar{\mathbf{h}}^{(l)}(t-1)\big)\Big) \odot \Big(1 - \phi^2\big(\mathbf{U}_h^{(l)}\big)\Big)
\end{aligned}$$

where $\boldsymbol{\delta}^{(l)}(t)$ is the backpropagation error for the training sample at layer l and timestep t. The gradients are similarly defined for the weights in the gates.

2.3.2. Second round: BGRU

The BGRU network is initialized with the real-valued weights from the first round, which are pretrained to be optimal for the source separation task. The real-valued weights are saved for the backpropagation step and used to construct bitwise weights for the feedforward procedure using both the mean-scaled sparsity mask $\mathbf{B}$ and Bernoulli mask $\mathbf{C}$. The bitwise activation functions, $\bar{\sigma}$ and $\bar{\phi}$ are applied during the feedforward as well. Again as an example, with the introduction of the masks and bitwise functions, the feedforward step for the hidden candidate state becomes:

$$\begin{aligned}
\widehat{\mathbf{W}}_h^{(l)} &= (\bar{\phi}(\mathbf{W}_h^{(l)}) \odot \mathbf{B}) \odot \mathbf{C} + \phi(\mathbf{W}_h^{(l)}) \odot (1 - \mathbf{C}) \\
\widehat{\mathbf{U}}_h^{(l)} &= (\bar{\phi}(\mathbf{U}_h^{(l)}) \odot \mathbf{B}) \odot \mathbf{C} + \phi(\mathbf{U}_h^{(l)}) \odot (1 - \mathbf{C}) \\
\mathbf{V} &= \widehat{\mathbf{W}}_h^{(l)} \mathbf{x}^{(l-1)}(t) + \widehat{\mathbf{U}}_h^{(l)}\big(\bar{\mathbf{r}}^{(l)}(t) \odot \bar{\mathbf{h}}^{(l)}(t-1)\big) \\
\widehat{\tilde{\mathbf{h}}}^{(l)}(t) &= \bar{\phi}(\mathbf{V}) \odot \mathbf{C} + \phi(\mathbf{V}) \odot (1 - \mathbf{C})
\end{aligned} \quad (11)$$

where $\mathbf{V}$ is an intermediary term. The Bernoulli parameter π is incremented gradually to determine $\mathbf{C}$ until the network is completely binarized at $\pi = 1.0$.

Backpropagation: The derivatives of non-differentiable activation functions are overwritten with the derivatives of their relaxed counterparts such that $\bar{\phi}' = \phi'$ and $\bar{\sigma}' = \sigma'$. This simplifies the gradients for (11). The gradients are computed as (10) with an additional factor for the masks which are:

$$\begin{aligned}
\nabla \mathbf{W}_h^{(l)} &= \nabla \mathbf{W}_h^{(l)} \odot (\mathbf{B} \odot \mathbf{C} + (1 - \mathbf{C})) \\
\nabla \mathbf{U}_h^{(l)} &= \nabla \mathbf{U}_h^{(l)} \odot (\mathbf{B} \odot \mathbf{C} + (1 - \mathbf{C}))
\end{aligned} \quad (12)$$

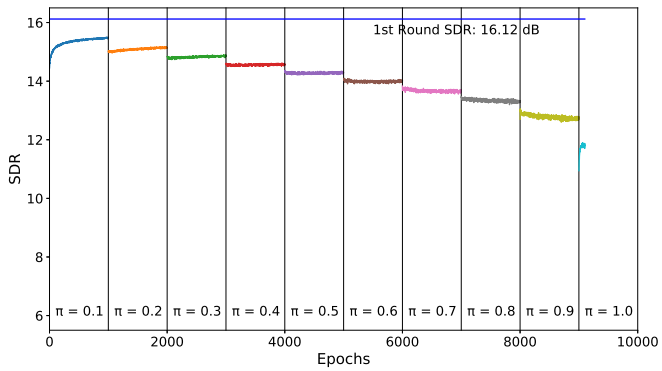
The gradients are computed similarly for the gates. The calculations in (12) show that the network is the same as the first round network except with the addition of masking factors. Only the real-valued weights are updated with the gradients during training.

3. EXPERIMENTS

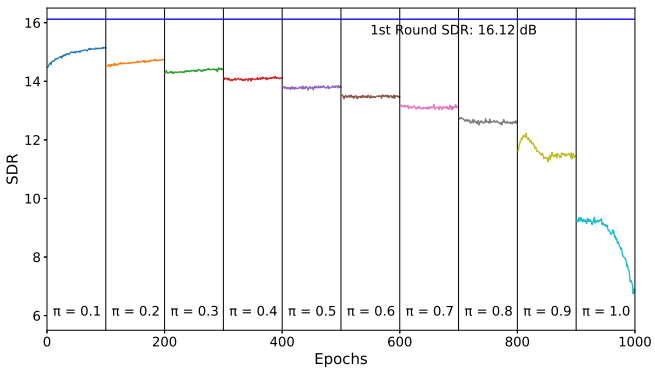
3.1. Experimental Setups

For the experiment, we randomly subsample 12 speakers for training and 4 speakers for testing from the TIMIT corpus. For both subsamples, we select half of the speakers as male and the other half as female. There are 10 short utterances per speaker recorded with a 16kHz sampling rate. Each utterances are mixed with 10 different non-stationary noise signals with 0 dB Signal-to-Noise Ratio (SDR), namely {birds, casino, cicadas, computer keyboard, eating chips, frogs, jungle, machine guns, motorcycles, ocean} [28]. In total, we have 227,580 training examples and 81,770 test examples from 1,200 and 400 mixed utterances, respectively. We apply a Short-Time Fourier Transform (STFT) with a Hann window of 1024 and hop size of 256. To quantize the spectra into bipolar binaries, we apply a 4-bit QaD procedure and convert them into $n \times (4 \times 513)$ dimension matrices. These vectors are used as inputs to the BGRU systems. The truncated BPTT length used was $T = 50$. We found $\rho = 0.8$ to perform well in our experiment. We used the Adam optimizer for both first and second rounds with the same beta parameters, $\beta_1 = 0.4$ and $\beta_2 = 0.9$. Minibatch size is set as 10 for 10 mixed utterances constructed from 1 clean signal mixed with the 10 noise signals. We train two types of networks that predict the IBMs with respect to the noisy quantized input:

- *Baseline with binary input*: The baseline network is constructed with a single GRU layer with $K = 1024$ units. The inputs to the network are 4×513 dimension 4-bit QaD vectors and predicted outputs are 513 dimension IBMs. We use the first round training algorithm to train the baseline network. For regularization, we apply dropout rate of 0.05 for the input layer and 0.2 for the GRU layers.
- *The proposed BGRU*: We initialize the weights with the pretrained weights and use the second round training algorithm to train the BGRU network. We increase the π parameter by 0.1 starting from 0.1 to 1.0. The learning rates are reduced for each increase in π.



(a) Results from 1000 epochs for $\pi < 1.0$ and 100 epochs for $\pi = 1.0$



(b) Results from 100 epochs for each π

Fig. 1. Second round testing results on incremental levels of π. Figures (a) and (b) show the effects of running different number of iterations.

Table 1. Speech denoising performance of the proposed BGRU-based source separation model compared to FCN, BNN, and GRU networks

Systems		Topology	SDR	STOI
FCN with original input		1024×2	10.17	0.7880
		2048×2	10.57	0.8060
FCN with binary input		1024×2	9.80	0.7790
		2048×2	10.11	0.7946
BNN		1024×2	9.35	0.7819
		2048×2	9.82	0.7861
GRU with binary input		1024×1	16.12	0.9459
BGRU	π=0.1	1024×1	15.50	0.9393
	π=0.2		15.17	0.9361
	π=0.3		14.90	0.9324
	π=0.4		14.58	0.9292
	π=0.5		14.32	0.9252
	π=0.6		14.02	0.9217
	π=0.7		13.66	0.9174
	π=0.8		13.30	0.9104
	π=0.9		12.70	0.9019
	π=1.0		11.76	0.8740

3.2. Discussion

Table 1 shows results for the BGRU along with other systems for comparison. The metrics displayed are Signal-to-Distortion Ratio (SDR) [29] and Short-Time Objective Intelligibility (STOI) [30]. At each increase in π, there is a distinct drop in SDR and STOI due to the loss in information as we increase the number of elements undergoing binarization. Since the initial weights transferred from the first round are optimal, we restrict the weights from updating too drastically by dampening the learning rate at each increase in π. We did not observe substantial difference from reducing the learning rate before $\pi = 0.8$, however the performance becomes sensitive as the rate of binarization nears 1. In Figure 1(a) it can be seen that from $\pi = 0.8$ the performance begins to decrease more than during previous π values.

The BGRU network is trained for an extended number of iterations so it propagates the corrections and adjusts to the quantization injected into the network. We trained 1000 epochs for each π values except at $\pi = 1.0$. Figure 1(a) shows that this many iterations is not always beneficial within the same session with a fixed π, because SDR improvement becomes stagnant and even starts to drop. However, in this way the network can prevent a greater drop in performance at the next increase in π. At $\pi = 1.0$, we only train for 100 epochs and perform early stopping because the network is less robust and degrades in performance after more than 100 epochs. Also, since the network has finished training for the source separation task at $\pi = 1.0$, further training is unnecessary. On the contrary, Figure 1(b) shows that training for less number of iterations, e.g. 100 epochs, produces a greater drop at each increment of π.

The drop in performance from a real-valued network to a bitwise version is quite comparable between a FCN with BNN and GRU with BGRU. The loss is much greater in the BGRU network (16.12 dB to 11.76 dB SDR) than in the case of BNN (10.11 dB to 9.82 dB SDR). Yet, the performance of a single-layer fully bitwise BGRU network with 1024 units (11.76 dB SDR and 0.8740 STOI) is still greater than that of a double-layer BNN with 2048 units (9.82 dB SDR and 0.7861 STOI), and also greater than that of a unquantized double-layer FCN with real-valued inputs and 2048 units (10.57 dB SDR and 0.8060 STOI). We discuss the space complexity of the BGRU network compared to a FCN and BNN. Considering that a GRU layer contains 3 sets of weights, the single layer BGRU network contains $3 \times (1024 \times 1)$ number of weights. This number is still less than a FCN or BNN of topology 2048×2. We introduced a real-valued scaling factor μ, but it reduces down to bipolar binaries once training is done, so it does not add additional costs.

In the future, we plan to extend the network structure to deeper ones. Also, more scheduled annealing of the π values is another option to investigate.

4. CONCLUSION

In this paper, we proposed an incremental binarization procedure to binarize a RNN with GRU cells. The training is done in two rounds, first in a weight compressed network and then in an incrementally bitwise version with the same topology. The pretrained weights of the first round are used to initialize the weights of the bitwise network. For the BGRU cells, we redefined the feedforward procedure with bitwise values and operations. Due to the sensitivity in training the BGRU network, the bitwise feedforward pass is performed gently using two types of masks that determine the level of sparsity and rate of binarization. With 4-bit QaD quantized input magnitude spectra and IBM targets, the BGRU at full binarization performs well for the speech denoising job with a minimal computational cost.

5. REFERENCES

[1] Y. Xu, J. Du, L.-R. Dai, and C.-H. Lee, "An experimental study on speech enhancement based on deep neural networks," *IEEE Signal processing letters*, vol. 21, no. 1, pp. 65–68, 2014.

[2] P. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis, "Joint optimization of masks and deep recurrent neural networks for monaural source separation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 12, pp. 2136–2147, 2015.

[3] A. A. Nugraha, A. Liutkus, and E. Vincent, "Multichannel audio source separation with deep neural networks.," *IEEE/ACM Trans. Audio, Speech & Language Processing*, vol. 24, no. 9, pp. 1652–1664, 2016.

[4] Y. Wang and D. Wang, "Towards scaling up classification-based speech separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 7, pp. 1381–1390, 2013.

[5] J. Le Roux, J. R. Hershey, and F. Weninger, "Deep NMF for speech separation," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 66–70.

[6] E. M. Grais, M. U. Sen, and H. Erdogan, "Deep neural networks for single channel source separation," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 3734–3738.

[7] H. Erdogan, J. R. Hershey, S. Watanabe, and J. Le Roux, "Phase-sensitive and recognition-boosted speech separation using deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 708–712.

[8] F. Weninger, H. Erdogan, S. Watanabe, E. Vincent, J. Le Roux, J. R. Hershey, and B. Schuller, "Speech enhancement with LSTM recurrent neural networks and its application to noise-robust ASR," in *International Conference on Latent Variable Analysis and Signal Separation*. Springer, 2015, pp. 91–99.

[9] F. Weninger, J. R. Hershey, J. Le Roux, and B. Schuller, "Discriminatively trained recurrent neural networks for single-channel speech separation," in *Proceedings 2nd IEEE Global Conference on Signal and Information Processing, GlobalSIP, Machine Learning Applications in Speech Processing Symposium, Atlanta, GA, USA*, 2014.

[10] Y. Isik, J. Le Roux, Z. Chen, S. Watanabe, and J. R. Hershey, "Single-channel multi-speaker separation using deep clustering," *arXiv preprint arXiv:1607.02173*, 2016.

[11] Z. Chen, S. Watanabe, H. Erdogan, and J. R. Hershey, "Speech enhancement and recognition using multi-task learning of long short-term memory recurrent neural networks," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[12] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[13] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural computation*, vol. 2, no. 4, pp. 490–501, 1990.

[14] I. Sutskever, *Training recurrent neural networks*, University of Toronto Toronto, Ontario, Canada, 2013.

[15] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[16] M. Kim and P. Smaragdis, "Bitwise neural networks for efficient single-channel source separation," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 701–705.

[17] M. Courbariaux, Y. Bengio, and J. P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[18] Joachim Ott, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio, "Recurrent neural networks with limited numerical precision," *arXiv preprint arXiv:1608.06902*, 2016.

[19] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[20] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[21] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna, "Analysis of high-performance floating-point arithmetic on fpgas," in *null*. IEEE, 2004, p. 149b.

[22] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. S. Hemmert, "Embedded floating-point units in fpgas," in *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*. ACM, 2006, pp. 12–20.

[23] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE, 2014, pp. 1–6.

[24] M. Courbariaux, Y. Bengio, and J. P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[25] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[26] M. Kim and P. Smaragdis, "Bitwise neural networks," in *International Conference on Machine Learning (ICML) Workshop on Resource-Efficient Machine Learning*, Jul 2015.

[27] S. Lloyd, "Least squares quantization in PCM," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[28] Z. Duan, G. J. Mysore, and P. Smaragdis, "Online PLCA for real-time semi-supervised source separation," in *International Conference on Latent Variable Analysis and Signal Separation*. Springer, 2012, pp. 34–41.

[29] E. Vincent, R. Gribonval, and C. Févotte, "Performance measurement in blind audio source separation," *IEEE transactions on audio, speech, and language processing*, vol. 14, no. 4, pp. 1462–1469, 2006.

[30] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen, "A short-time objective intelligibility measure for time-frequency weighted noisy speech," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE, 2010, pp. 4214–4217.