information

MDPI

Article

Shrink and Eliminate: A Study of Post-Training Quantization and Repeated Operations Elimination in RNN Models

Nesma M. Rezk [1,*], Tomas Nordström [2] and Zain Ul-Abdin [1]

[1] School of Information Technology, Halmstad Univeristy, 30118 Halmstad, Sweden; zain-ul-abdin@hh.se
[2] Department of Applied Physics and Electronics, Umeå University, 90187 Umeå, Sweden; tomas.nordstrom@umu.se
* Correspondence: nesma.rezk@hh.se

Abstract: Recurrent neural networks (RNNs) are neural networks (NN) designed for time-series applications. There is a growing interest in running RNNs to support these applications on edge devices. However, RNNs have large memory and computational demands that make them challenging to implement on edge devices. Quantization is used to shrink the size and the computational needs of such models by decreasing weights and activation precision. Further, the delta networks method increases the sparsity in activation vectors by relying on the temporal relationship between successive input sequences to eliminate repeated computations and memory accesses. In this paper, we study the effect of quantization on LSTM-, GRU-, LiGRU-, and SRU-based RNN models for speech recognition on the TIMIT dataset. We show how to apply post-training quantization on these models with a minimal increase in the error by skipping quantization of selected paths. In addition, we show that the quantization of activation vectors in RNNs to integer precision leads to considerable sparsity if the delta networks method is applied. Then, we propose a method for increasing the sparsity in the activation vectors while minimizing the error and maximizing the percentage of eliminated computations. The proposed quantization method managed to compress the four models more than 85%, with an error increase of 0.6, 0, 2.1, and 0.2 percentage points, respectively. By applying the delta networks method to the quantized models, more than 50% of the operations can be eliminated, in most cases with only a minor increase in the error. Comparing the four models to each other under the quantization and delta networks method, we found that compressed LSTM-based models are the most-optimum solutions at low-error-rates constraints. The compressed SRU-based models are the smallest in size, suitable when higher error rates are acceptable, and the compressed LiGRU-based models have the highest number of eliminated operations.

Keywords: recurrent neural network; quantization; delta networks; edge devices

check for updates

Citation: Rezk, N.M.; Nordström, T.; Ul-Abdin, Z. Shrink and Eliminate: A Study of Post-Training Quantization and Repeated Operations Elimination in RNN Models. *Information* **2022**, *13*, 176. https://doi.org/10.3390/info13040176

Academic Editors: Lorenzo Carnevale and Massimo Villari

Received: 28 February 2022
Accepted: 28 March 2022
Published: 31 March 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

A recurrent neural network (RNN) is a neural network (NN) that can handle the temporal relationship between inputs or outputs in time-series applications. RNNs are found in speech recognition, language translation, video-captioning, and many other applications. These applications are essential on edge devices used in cars, smart homes, and wearable devices. Like all other NNs, RNNs suffer from high computation and memory requirements, making realizing it on edge devices a challenging mission [1]. RNNs can be subjected to algorithmic optimization before deployment to decrease their size and increase the implementation efficiency on edge devices [1].

In this work, we focus on two kinds of model optimization. The first is quantization, where fewer bits are used for the weights and activation vectors. Quantization significantly enhances the performance on embedded platforms by compressing the model's size. A compressed model can be stored in the on-chip memory and saves time and energy during the frequent loading of data from off-chip memory. In addition, the decreased

precision of operands requires less complex multipliers and runs faster than full-precision operations. The second model optimization we apply in this work is the delta networks method. The idea of the delta networks method is based on the high similarity between input vectors in temporally related applications and the needlessness of repeating computations for highly similar data. Thus, input and activation vectors are compared with the vectors of previous time-steps. If the difference between any two corresponding elements is less than a predefined delta threshold θ, the computations for this element are eliminated.

The sparsity generated by applying the delta algorithm is sparsity in the vector, not the matrix. If the matrix-to-vector ($M \times V$) multiplications are carried out in a column-wise fashion, handling the sparsity becomes easier [2,3]. Each zero in the vector is multiplied by a column in the matrix. Thus, it is possible to eliminate loading a complete column from the weight matrix once a zero is detected in the delta vector. Then, this column is also eliminated from the $M \times V$ multiplications.

Applying model optimizations on RNNs can affect the accuracy of the model. Thus, retraining is usually applied to retain the loss of accuracy. On the other hand, it is possible to skip the training step by using more cautious methods during the optimization. For example, the selection of the clipping threshold during quantization has an influential role in decreasing the quantization error without the requirement of retraining. Post-training optimizations are also helpful in scenarios with no training data available, possibly for security reasons or due to the lack of training servers. We adopt a post-training approach while applying the model optimizations in this work.

In this article, we apply our experiments on four different recurrent neural network models based on LSTM, GRU, LiGRU, and SRU units for speech recognition on the TIMIT dataset. We studied the quantization of SRU-based models earlier [4]. We found that SRU-based models can be compressed to very small-sized models with a small increase to their baseline model's error. In addition, the removal of the previous time-step vectors from $M \times V$ computations made it possible to parallelize their execution over multiple time-steps. This property made SRU-based models favorable from a hardware-implementation perspective. In this work, we apply the post-training quantization on the four models. Then, we combine the quantization with the delta networks method and evaluate the combined effect on the error rate and percentage of eliminated computations. Finally, we compare the four studied models under compression to determine which model will perform better in different scenarios. The contributions of this work are summarized as follow:

- We propose "selected path skipping" during the quantization of recurrent units to decrease the error rates.
- We analyze and show a positive synergic effect of quantization on the delta networks method.
- We propose a method for delta threshold selection in a post-training scenario.
- We compare the four RNN models to find the smallest size and least number of operations at different error levels.

This article is organized as follows. Section 2 presents related research. Section 3 discusses RNN models and optimization methods briefly. Afterwards, Section 4 explains how we optimize the RNN models, and Section 5 evaluates the optimization of RNN models understudy. In Section 6, we compare the results from Section 5, and the concluding remarks are provided in Section 7.

2. Related Work

In this section, we discuss the work related to this article. First, we discuss the post-training quantization of neural networks. Then, we discuss the work done on RNN models using the delta algorithm.

Several work have been done on the post-training quantization of CNNs for image applications [5–9] and RNNs for text/language applications [8,10]. Banner et al. combined their proposed clipping method (ACIQ) with a bit-channel allocation policy and bias correction to minimize the accuracy loss [5]. They carried out experiments on six ImageNet models using 4/8-bit for weights and activations. Zhao et al. leveraged the Net2WiderNet

layers transformations to improve the quantization by outlier channel splitting (OCS) [8,11]. OCS reduces the magnitude of the outlier neurons by duplicating them and then halving the neurons' output values or their outgoing weights to preserve the functional correctness. By using OCS solely or by combining it with clipping methods, they quantized the models to 8-, 6-, and 4-bit while preserving high accuracy for ImageNet and WikiText-2 models.

ZeroQ used distilled data generated during training and fed it to the model to get a statistical distribution similar to that of the original model [9]. The distilled data distribution is used to compute the clipping range. Different precisions have been used for different layers, and an optimization algorithm was used to find the per-layer precision. ZeroQ was tested on ImageNet models as well, using 4- and 8-bit.

All previous post-training quantization methods do not require any training epochs after applying quantization. However, some data is required to calibrate or compute quantization clipping ranges. Nagel et al. proposed a data-free quantization method [6]. Their method successfully quantized ImageNet models to 8-bits but not less.

After the proposal of the delta networks method [12], some work has been done to apply the method to RNN models and design powerful accelerators that can run the algorithm. Gao et al. designed a one-layer GRU FPGA accelerator that uses 16-bit fixed-point operations [2]. They trained the model using the same delta threshold applied during inference and applied their method on the TIDIGIT dataset. They managed to reach up to 5.7× speedup with negligible accuracy loss.

EdgeDRNN is an accelerator that applied the delta network method on GRU-based models [13]. The accelerator uses an 8-bit integer for the weights and 16-bit fixed-point activations. A two-layer GRU-based model was trained while applying the delta network method on the TIGIT dataset. The word-error rate increased slightly from 0.77 to 1.3 on the TIDIGIT dataset to allow the arithmetic units to compute ten times faster.

Jo et al. proposed an accelerator that uses 8-bit integer MAC operations and applied the delta network method [3]. They reduced the operations by 55% on an LSTM model on the Librispeech dataset with only a 1% increase in error. Similarly, an AC codec based on the delta networks method for lossless compression has been proposed and implemented on a Zynq-7000 SoC board [14].

None of the discussed literature applied post-training quantization on GRU or LiGRU units. In addition, no one addressed the effect of applying the delta network method on integer-quantized RNNs. Nevertheless, the selection of the delta threshold in all of the previous work used to be heuristic, and no information could be predicted about how good or bad different thresholds were when applied to various models.

3. Background

3.1. Recurrent Neural Network Models

RNNs recognize the temporal relation between data sequences by adding recurrent layers to the NN model. The recurrent layer adds feedback from previous time steps and memory cells. In this paper, we use four types of recurrent layers, LSTM [15], GRU [16], LiGRU [17], and SRU [18]. Our primary focus is to decrease the computational complexity of the $M \times V$ multiplications. The LSTM has four equations with $M \times V$ multiplications, the input and the hidden-state vector (previous time-step output) are multiplied with weight matrices, added to bias, and applied to a non-linear function such as the forget-gate equation:

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f), \tag{1}$$

where x_t is the input vector, h_{t-1} is the hidden-state output vector, W_fx is the input weight matrix, W_fh is the hidden-state weight matrix, b_f is the bias vector, and σ is the *sigmoid* function. In the computation of the forget, input, and output gates the non-linear function is σ, while in the candidate-state equation, the non-linear function is *tanh*:

$$\widetilde{C}_t = \tanh(W_{Cx}x_t + W_{Ch}h_{t-1} + b_C), \tag{2}$$

where x_t is the input vector, h_{t-1} is the hidden-state output vector, W_c is the weight matrix, and b_c is the bias vector. The GRU unit has three $M \times V$ equations: two gate equations (reset and update) using a σ non-linear function and a candidate output vector equation using a tanh non-linear function. LiGRU is a light version of GRU that has only two $M \times V$ multiplications equations. One of them is the update gate using a σ non-linear function, and the other is a candidate output vector equation using a RELU non-linear function. SRU implemented a different approach by removing the previous time steps from all $M \times V$ multiplications and using them only in element-wise computations. All the recurrent layers used in this work are bidirectional. The input is fed into the layer from past to future and future to past, which helps the network better understand the context.

In this work, we use the models provided by the Pytorch–Kaldi [19] library as illustrated in Figure 1. The Pytorch–Kaldi project develops hybrid speech recognition systems using state-of-the-art DNN/RNN, where Pytorch is used for the NN acoustic model. Feature extraction, label computation, and decoding are done by the Kaldi toolkit [20]. We used the logarithmic Mel-filter bank coefficients (FBANK) for feature extractions in our experiments. The acoustic model training labels come from a procedure of forced alignment between the speech features and the context-dependent phone state sequence. Afterwards, the features vector is fed into the Pytroch NN as input, and the output vector is generated in the form of posterior probabilities over phone states. Finally, the Kaldi decoder computes the final word-error-rate (WER). We trained the model using the TIMIT dataset for 24 epochs as configured in the Pytorch–Kaldi default experiments. TIMIT is a dataset composed of recordings of 630 different speakers using six different American English dialects, where each speaker is reading up to 10 sentences [21].

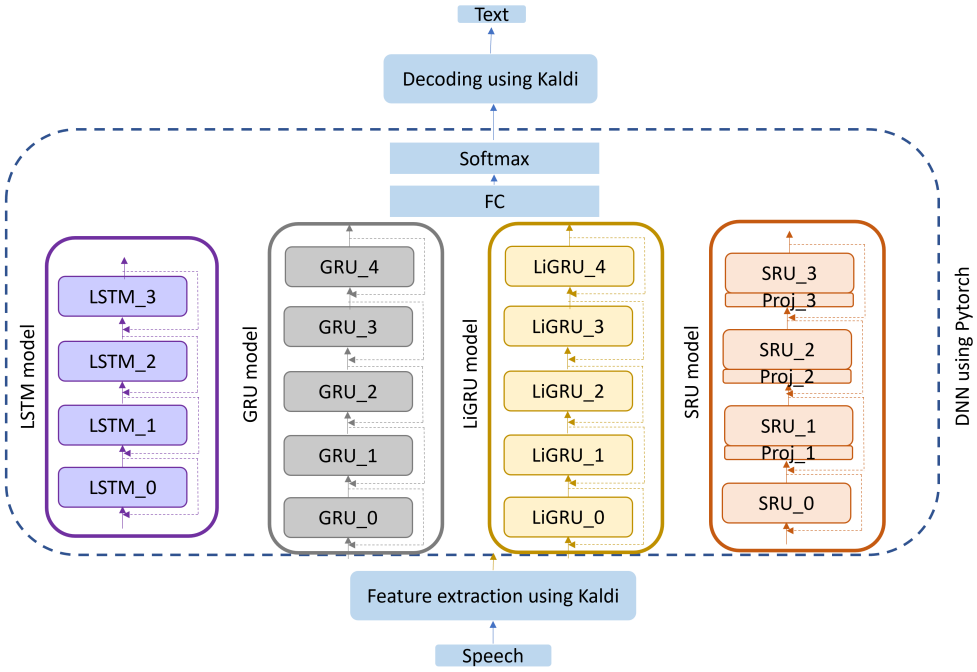

Figure 1. The ASR models using KALDI for feature extraction and decoding and RNN for speech recognition. The RNN can be a LSTM-, GRU-, LiGRU-, or SRU-based model.

3.2. Post-Training Quantization

Quantization compressed the NN model by reducing the precision of weights and activations. NN models can be quantized to 16-bit fixed-point—which usually does not affect the model's accuracy—or to integer precisions [22]. Integer quantization makes the deployment of NN models on embedded platforms more feasible [23,24]. The precision can be reduced to anything between 8-bit and 1-bit. However, Integer quantization can

highly degrade model accuracy, and training is required to correct for the accuracy loss in such cases.

Alternatively, certain post-training processing methods using wiser quantization have been proposed to eliminate the need for training after quantization. It has been observed that the outlier values in the full-precision vectors unnecessarily consume the allowed integer precision. Thus, clipping the data range and outlier saturation is extensively used for post-training quantization [7,8]. One other solution, called outlier channel splitting (OCS), reduces the outlier effect by duplicating the channels with outlier values. Then, the output values or their outgoing weights are halved to preserve the correctness of computations [8]. Unfortunately, channel duplication leads to an increase in the model size. In this work, we apply clipping during integer linear quantization using the minimum mean square error (MMSE) clipping-threshold-selection method [25].

3.3. Delta Networks Method

The Delta networks method changes the $M \times V$ computation without affecting the result [12] by capturing the redundancy in input/hidden vectors and eliminating repeated computations. So, the equation was initially:

$$MV_t = W_x x_t + W_h h_{t-1}, \tag{3}$$

where MV_t is the matrix-to-vector multiplication result in the current time-step, W_x is the input weight matrix, x_t is the input vector, W_h is the recurrent weight matrix, and h_{t-1} is the hidden-state vector. The equation is changed to:

$$MV_t = W_x \Delta_x + W_h \Delta_h + MV_{t-1}, \tag{4}$$

where $\Delta_x = x_t - x_{t-1}$, $\Delta_h = h_{t-1} - h_{t-2}$, and MV_{t-1} is the matrix-to-vector multiplication result in the previous time-step.

The delta vectors will have zeros for each two similar values in successive vectors. Later, when the delta vectors are used in $M \times V$ computations, computations corresponding to the zeros will be Eliminated. As a further development, approximation was added to the method to increase the quantity of zeros in the delta vectors. When comparing successive vectors, values do not have to be exactly the same to put a zero in the delta vector. Instead, a delta threshold Θ is defined, and if the difference between two values is less than Θ , they are considered similar, and zero is placed in the delta threshold [12]. However, if we only compare two time-step vectors, we may suffer from error accumulation over time steps. For instance, the input vector x_t can increase every time-step with a value close to Θ, and the method will keep eliminating the vector computations. Thus, the delta method authors introduced memory records (states) to store the last values that caused the over-threshold changes. The states equations are as follow:

$$\begin{aligned} \hat{x}_{i,t-1} &= \begin{cases} x_{i,t-1} & \text{if } |x_{i,t-1} - \hat{x}_{i,t-2}| > \Theta, \\ \hat{x}_{i,t-2} & \text{otherwise,} \end{cases} \\ \hat{h}_{i,t-1} &= \begin{cases} h_{i,t-1} & \text{if } |h_{i,t} - \hat{h}_{i,t-2}| > \Theta, \\ \hat{h}_{i,t-2} & \text{otherwise,} \end{cases} \end{aligned} \tag{5}$$

where the states $\hat{x}_{i,t-1}$ and $\hat{h}_{i,t-1}$ store the last change to the element i in the input and hidden state vectors, respectively. The new delta vectors equations are:

$$\begin{aligned} \Delta x_{i,t} &= \begin{cases} x_{i,t} - \hat{x}_{i,t-1} & \text{if } |x_{i,t} - \hat{x}_{i,t-1}| > \Theta, \\ 0 & \text{otherwise,} \end{cases} \\ \Delta h_{i,t} &= \begin{cases} h_{i,t} - \hat{h}_{i,t-1} & \text{if } |h_{i,t} - \hat{h}_{i,t-1}| > \Theta, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{6}$$

To compute the delta vectors $\Delta x_{i,t}$ and $\Delta h_{i,t}$, the current input and hidden-state vectors are compared to the state vectors, and if the difference between each pair of elements i is less than Θ, the i element is set to zero in the delta vector.

4. Method

This section discusses the post-training quantization of different recurrent layers. Then, it shows how the delta networks method benefits from integer quantization. Finally, it presents the method of delta threshold selection for different RNN models.

4.1. Quantization of Recurrent Neural Networks Models

The recurrent layers are mainly composed of $M \times V$ multiplications between weight matrices and input or hidden state vectors, element-wise arithmetic operations, and non-linear operations. Since the bottleneck for computations in the recurrent layers is the $M \times V$ multiplications, we do not quantize any other part in the layer. Initially, we quantize the weight matrices to integer precision and keep all other weights in a 16-bit fixed-point format. During inference, all vectors are in a 16-bit fixed-point format. Before the $M \times V$ operations, the activation vector is quantized to the required integer precision. After $M \times V$ is carried out, the activation vector is requantized back to a 16-bit fixed point as seen in Figure 2. We used linear quantization to quantize the weights to integer precisions, and we used the minimum mean square error (MMSE) method to determine the clipping threshold [25] during quantization. In our code, we used the implementation of MMSE clipping provided by other work on Github [8]. We had to modify the code to support varying precisions between model layers and to support recurrent neural network layers. The range of the quantized values are (−128:127), (−8:7), and (−2:1) for 8-bit, 4-bit, and 2-bit, respectively. The same quantization method is used during integer quantization of activation. A portion of validation data is used during the clipping threshold computation since it is not feasible to be computed during inference on edge devices. It requires the computation of the range of the activation vector.

As we mentioned, all other weights are kept in a 16-bit fixed-point format. To quantize to fixed-point, we compute the number of bits that are enough to store the integer part, and the remaining bits are used to store the approximated fraction part. To requantize integer vectors into fixed-point values, integer values are divided by a scale value computed earlier, using the validation data to be in the same range as if quantization had not been applied.

During the quantization of LSTM, GRU, and LiGRU, we have experimented with turning off quantization in some gates to see if any of the gates are more sensitive to quantization. We found out that in each of the recurrent models there is a gate for which we can skip integer quantization in the recurrent computation of the gate to have a much lower quantization error. In LSTM, the skipped part is the candidate state computation. In GRU and LiGRU, the skipped part is the candidate output vector computation. In SRU, this skipping is not applied, as the SRU has no $M \times V$ applied to the hidden state vector. In Figure 2, we show an example of $M \times V$ computations in a gate during integer quantization, and we show how we skip the quantization in the recurrent part of the gate computation.

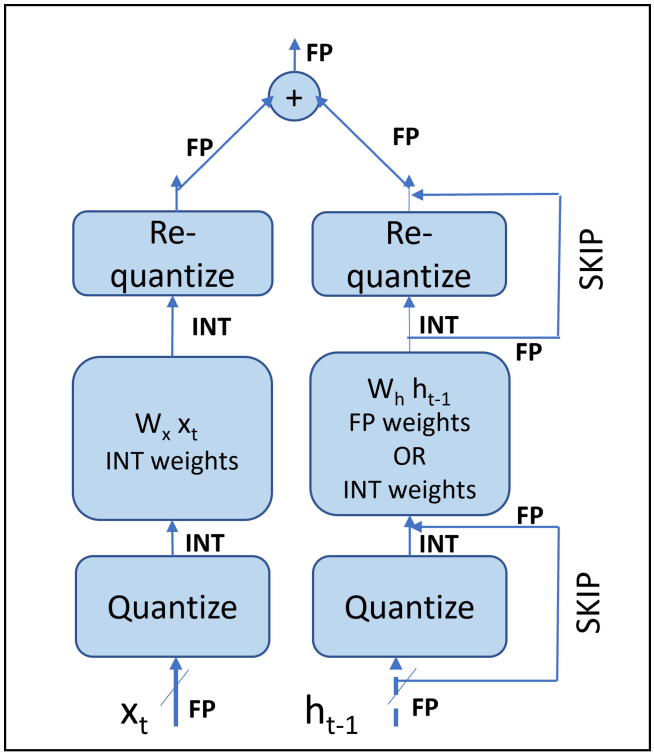

Figure 2. Integer matrix-to-vector ($M \times V$) multiplication in a recurrent unit gate. Both of the input and the hidden state vectors are quantized to integer precision. Then, matrix-to-vector multiplication is applied between integer vectors and integer weights. Then, the vectors are re-quantized to a 16-bit fixed point. If the gate is configured to skip the recurrent path quantization. The hidden state vector is kept in a 16-bit fixed-point by skipping quantization; then it is multiplied with fixed point weights and requantization is skipped.

4.2. Applying the Delta Networks Method on Quantized Recurrent Models

We have studied the effect of activation vector integer quantization on successive input and hidden state vectors in different types of recurrent models. We computed the delta vector described in Equation (5). Then, we computed the histogram of the delta vectors. We found that delta vectors have a high percentage of zeros among their values. Thus, if the delta networks method is applied during inference, many computations can be eliminated without any effect on accuracy. We believe that integer quantization, while decreasing the resolution of fixed-point values, quantized many values to be the same. Thus, integer quantization acted as a double weapon. To visualize this effect, we selected two quantization configurations with 8-bit and 4-bit as activation precision. In Figure 3, for these two quantization configurations, in the x-axis, we put the possible numerical values that can be found in the delta vector. In the y-axis, we plot the frequency of appearance of this value in the vector added to the frequencies of appearance of all values less than this value. So, if we look at the figure we see the percentage of computations that can be eliminated if a specific value is selected as a delta threshold. As we see in Figure 3, different recurrent models show different behaviors. The LiGRU-based model has a high percentage of eliminated operations (EO) at zero delta threshold even for 8-bit activations. LSTM and GRU each have around 50% of EO percentage at 4-bit activations while having a low EO percentage at 8-bit activations. However, the 8-bit plot shows that more operations can be eliminated if the delta threshold is increased over zero.

Thus, the next step would be to increase the delta threshold in a post-training scenario to maximize the percentage of eliminated operations (EO) with a minimal effect on accuracy. As we see in Figure 3, different delta thresholds have different effects on RNN models' percentage of EO. Consequently, we expect various delta thresholds to have different effects on the accuracy levels of different RNN models. Thus, we will use a simple approach

to explore the sensitivity of RNN models to delta thresholds at different activations' precisions. For each model, we want to decide on a threshold for delta for the input vectors, hidden state vectors, and FC activation vector in two cases: once when the vector precision is an 8-bit integer and once when it is a 4-bit integer. Thus, we form some combinations of possible delta thresholds and apply them during quantization then run inference using a portion of the validation data. We selected half of the validation data that would resemble the same distribution of speaker gender and dialects in the testing dataset. Then, we plotted the resulting error rate versus the resulting EO percentage and drew a Pareto curve for the dominating points. Then, we selected the point that would maximize the increase in the EO while minimizing the error increase if found.

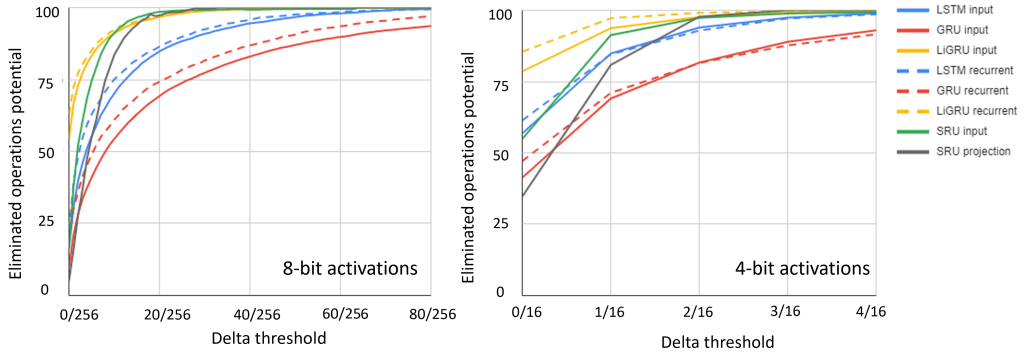

Figure 3. The eliminated operations potential plotted vs the delta threshold. In a delta vector, we computed the eliminated operations potential as the summed frequency of the delta threshold and all smaller delta thresholds. So, if a value is selected as a delta threshold, we can predict how many operations will be eliminated.

In Figure 4, we plot the Pareto curve for the LSTM model for 8-bit activations. We use a quantized model that has all the weights and activations in 8-bit. At 8-bit integer, the possible thresholds are (0, 8, 16, 32, 48, ...). We evaluate combinations of thresholds until the error levels stop being acceptable. In Figure 4, we show how we select the delta-thresholds to further increase the percentage of EO in the LSTM model. We select a threshold of 8 for the input and hidden state vectors and for the FC activation vector. We follow the same method for all types of models under-study. The LiGRU model and the SRU model show a higher increase in the EO percentage with lower thresholds in Figure 3. Thus, we add 4 as a possible threshold in their case. In the case of 4-bit activations, the possible delta thresholds are (0, 1, 2, 3, ...). We skip the first layer from the study by keeping the delta threshold 0, and we do not quantize it to 4- bit (i.e., we kept it at 8-bit) to prevent its higher sensitivity to compression to affect the results.

As we mentioned earlier, SRU models do not have recurrent vectors involved in the $M \times V$ multiplication. Thus, we keep the hidden state vectors in a 16-bit fixed-point format. Also, as we see in Figure 1, the SRU model has projection layers in-between the SRU layers that are not present in the other models we studied. Thus, the three thresholds in the SRU case are the input vector, the projection layer activation vector, and the FC activation vector. Later, in Section 5, we will examine the effect of applying the chosen thresholds on different configurations of quantization of the models under study.

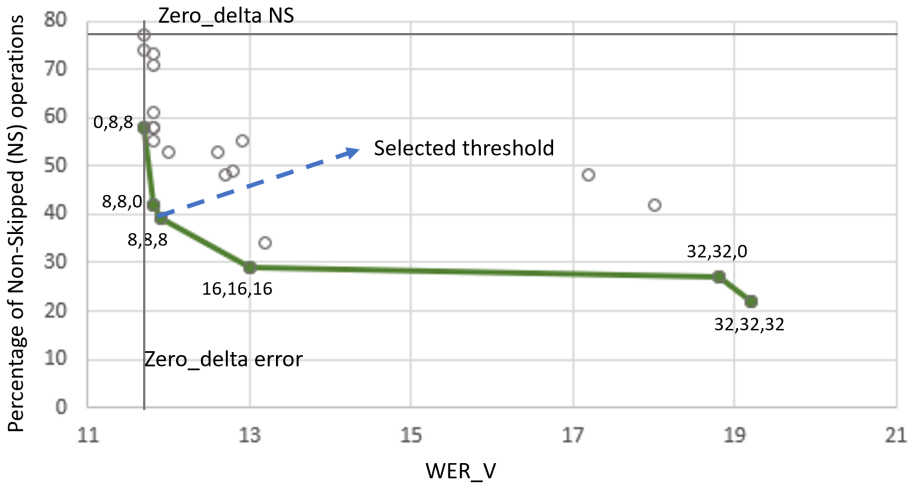

Figure 4. The selection of the delta thresholds for the LSTM-based models with 8-bit integer activation vectors. A threshold of 8 is selected for the input vector, hidden vector, and FC activation vector.

5. Evaluation

This section evaluates the effect of quantization with operations elimination on different RNN models. First, we show the impact of selected recurrent path skipping from integer quantization in LSTM-, GRU-, and LiGRU-based models. Then, we evaluate the effect of post-training quantization with different precision configurations. Finally, we assess the effect of applying the delta networks method on the quantized models using the delta threshold of zero and higher values.

5.1. Impact of Skipping Quantization in Selected Recurrent Paths

In this section, we show how to decrease the error of quantized models by skipping selected paths during quantization. As mentioned in Section 4, the paths to skip are the recurrent path in the candidate output computation equation in GRU and LiGRU and the candidate state computation in the LSTM. In Table 1, we quantize the LSTM-, GRU-, and LiGRU-based models to 8-bit and 4-bit integers. We show the resulting error rate and size in three cases. In the first case, all the paths in all the gates are quantized to integer precision. In the second case, we skip the early stated selected recurrent path. In the third case, we do not skip the selected path in the last two layers. We think that the last two layers may not suffer from quantization in the same way as the early layers. Thus, we might increase the compression ratio in the third case without increasing the error as much.

As observed in Table 1, the error decreased by about 2% in the LSTM-based model. In the GRU- and LiGRU-based models, the decrease is even more, ranging between 3.8% and 6%. The path skipping increased the models' sizes by 1 or 2 MB. Not skipping the selected path in the last two layers decreased the effect of the skipping on the memory increase with a slight increase in the error rate. We apply the selected path skipping in all the layers during quantization in the later experiments. We want to decrease the error rate as much as possible and keep our choices simple for hardware implementation.

Table 1. The effect of selected path skipping during integer quantization of LSTM-, GRU-, and LiGRU-based models. The effect is evaluated at 8-bit and 4-bit quantization. The error and size are evaluated at three cases: no skip—no path skipping is applied; skip—path skipping is applied to all model layers; skip part—path skip is not applied to last two recurrent layers in the model. WER is the testing word-error-rate of the models.

Type	***Sol.***	**No Skip**		**Skip**		**Skip Part**	
		WER	**Size**	***WER***	**Size**	***WER***	**Size**
LSTM	8-bit	16%	13.6	14.2%	14.8	14.4%	14.2
	4-bit	17.2%	6.8	15.1%	8.5	15.1%	7.7
GRU	8-bit	19.3%	13.3	15.5%	14.8	15.6%	14.2
	4-bit	21.9%	6.7	16%	8.8	16.3%	8
LiGRU	8-bit	21%	9.6	14.9%	10.5	14.9%	10
	4-bit	21.6%	4.8	15.5%	6.2	15.5%	5.7

5.2. Post-Training Quantization of RNN Models

In this section, we apply post-training quantization on the four models visualized in Figure 1 with selected path skipping as explained in Section 4. We first quantize the models' weights and activations to 16 bit-fixed and 8-bit, 4-bit, and 2-bit integers. Then we select eight mixed-precision configurations named $M1$ to $M8$. In the mixed-precisions configuration, we mix between 8-bit, 4-bit, and 2-bit weights and 8-bit and 4-bit activations. We select the precisions of the first layer input computations separately as we know it is the most sensitive layer [8]. The rest of the recurrent layer $M \times V$ multiplication operations have the same precision.

For each quantization configuration, we show the resulting testing error rate and the size of the model in Table 2. As observed in the table, all models maintain low error rates at 8-bit quantization. The error rate increases reasonably at 4-bit quantization, But, at 2-bit quantization all the models fail. In the mixed-precision solutions, it is possible to quantize the models' weights to 2-bit and have the activation with higher precision to achieve a reasonable error increase. Furthermore, if the first layer input computation is kept in 8-bit integer precision, the lowest error rates are achieved (M1, M2, M3, M6, and M7). The LSTM-based and the SRU-based models keep the error increase between 0.5 p.p. (percentage points) and 1.7 p.p. (LSTM-based) and 2.5 p.p. (SRU-based) compared to their baseline errors. At the same time, the GRU model has a negligible increase in the error. In contrast, the increase of the baseline error of the LiGRU-based model is found higher.

Table 2. The result of applying post-training quantization on the LSTM-, GRU-, LiGRU-, and SRU-based models. The first row is for the baseline model size and error rate. The second four rows are for non-mixed quantization. The last group of rows are for selected mixed quantizations with integer precisions. $L0x$ is the precision of first-layer input computations, while LXH is the precision of the rest of the layers' input and recurrent (projection is SRU case) computations. FC is the precision of the FC layer. W/A denotes weight and activation precision. For each quantization configuration, we show the WER, size.

Sol.	*L0X* W/A	*LXH* W/A	*FC* W/A	LSTM		GRU		LiGRU		SRU	
				WER	Size	*WER*	Size	*WER*	Size	*WER*	Size
Base	32/32	32/32	32/32	14.1%	54.4	15.4%	53	14.5%	38.2	17.2%	21.2
F-pt.	16/16	16/16	16/16	14.1%	27.2	15.4%	26.6	14.5%	19.1	17.2%	10.6
8-bit	8/8	8/8	8/8	14.2%	14.8	15.5%	14.8	14.9%	10.5	16.9%	5.3
4-bit	4/4	4/4	4/4	15.1%	8.5	16%	8.8	15.5%	6.2	17.9%	2.7
M1	8/8	2/8	8/8	14.7%	7	15.4%	7.4	16.6%	5.6	17.9%	2.9
M2	8/8	2/8	4/4	15.1%	6	15.5%	6.4	17.1%	4.6	18.4%	1.9
M3	8/8	2/8	2/8	15.8%	5.5	15.7%	5.9	17.4	4.1	18.5%	1.4
M4	2/8	2/8	2/8	17.4%	5.4	17.2%	5.8	23.3%	4.1	21.4%	1.3
M5	8/8	4/4	4/4	14.6%	8.6	15.5%	8.9	15.4%	6.2	17.4%	2.7
M6	8/8	2/4	4/4	14.8%	6	15.5%	6.4	16.9%	4.6	19.3%	1.9
M7	8/8	2/4	2/4	15.8%	5.7	15.7%	6	17.4%	4.2	19.7%	1.5
M8	4/4	2/4	2/4	16.4%	5.7	16.2%	6	18.5%	4.2	20.6%	1.4

5.3. Redundant Operation Elimination

This section evaluates the effect of applying the delta networks method on quantized RNN models for the four models under study. First, we keep the delta threshold zero and evaluate the eliminated operations (EO) percentage for all the quantization configurations used in the previous experiment. Then, we apply the method explained in Section 4 to select the optimum delta thresholds for different models. Finally, we reevaluate the EO under all quantization configurations using the chosen delta thresholds.

We present the experiment results in Table 3. When we apply the delta thresholds of values greater than zero, we put the resulting error and EO between brackets. In some cases, we cannot find a delta threshold other than zero that can preserve low error. In this case, we do not repeat the evaluation. The results show that the EO percentage is quite low at 8-bit activation precision for the LSTM-, GRU-, and SRU-based models. It was necessary to increase the delta threshold to reach a considerable EO percentage. However, the LiGRU has a high EO percentage for 8-bit activations even at zero delta threshold, and increasing the delta threshold does not give much increase in the EO percentage.

For the 4-bit integer activation solutions, all models have a high EO percentage. The SRU and the LiGRU-based models did not permit the increase in the delta thresholds. The LSTM- and the GRU-based models at higher delta thresholds gained more than a 10% increase in EO.

Table 3. The result of applying post-training quantization with delta networks method on the LSTM-, GRU-, LiGRU-, and SRU-based models. $L0x$ is the precision of first-layer input computations, while LXH is the precision of the rest of the layers' input and hidden state (projection is case of SRU) computations. FC is the precision of the FC layer. W/A denotes weight and activation precision. For each quantization configuration, we show the WER and EO (percentage of eliminated operations). We applied delta with thresholds higher than zero and put resultant error and EO is between brackets.

Sol.	*L0X* W/A	*LXH* W/A	*FC* W/A	LSTM		GRU		LiGRU		SRU	
				WER	EO	*WER*	EO	*WER*	EO	*WER*	EO
8-bit	8/8	8/8	8/8	14.2% (14.4%)	17% (56%)	15.5% (15.5%)	18% (56%)	14.9% (15%)	48% (55%)	16.9% (16.8%)	20% (31%)
4-bit	4/4	4/4	4/4	15.1% (14.9%)	51% (66%)	16% (16.1%)	46% (64%)	15.5%	64%	17.9%	55%
M1	8/8	2/8	8/8	14.7% (14.7%)	22% (55%)	15.4% (15.5%)	17% (56%)	16.6% (16.8%)	49% (56%)	17.9% (18.4%)	20% (31%)
M2	8/8	2/8	4/4	15.1% (15.1%)	25% (57%)	15.5% (15.6%)	18% (57%)	17.1% (17.2%)	53% (56%)	18.4% (18.8%)	34% (45%)
M3	8/8	2/8	2/8	15.8% (16%)	20% (56%)	15.7% (15.8%)	17% (56%)	17.4% (17.4%)	49% (56%)	18.5% (18.8%)	20% (31%)
M4	2/8	2/8	2/8	17.4% (17.6%)	21% (56%)	17.2% (17.3%)	19% (57%)	23.3% (23.9%)	51% (57%)	21.4% (21.5%)	20% (33%)
M5	8/8	4/4	4/4	14.6% (14.7%)	52% (66%)	15.5% (15.8%)	47% (64%)	15.4%	65%	17.4%	56%
M6	8/8	2/4	4/4	14.8% (15.2%)	52% (66%)	15.5% (15.8%)	47% (64%)	16.9%	63%	19.3%	55%
M7	8/8	2/4	2/4	15.8% (16.1%)	51% (66%)	15.7% (16.1%)	46% (64%)	17.4%	63%	19.7%	55%
M8	4/4	2/4	2/4	16.4% (16.6%)	51% (65%)	16.2% (16.6%)	46% (63%)	18.5%	63%	20.6%	53%

6. Discussion

In this section, we analyze the results in Tables 2 and 3. First, we provide a relative comparison, where each model is compared to its baseline to see how much the model can be compressed. Then, we compare the size and the number of operations different models can provide at different error levels to hold an absolute comparison.

6.1. Compressability Analysis

In this section, we study the ability of the models to be compressed while maintaining a minor effect on the error level. We measure compressibility in two metrics: the compression ratio and the percentage of eliminated operations. Those two percentages are computed against each model's baseline size and total number of operations. Thus, in Figures 5 and 6, we put in the x-axis the increase in error in percentage points. The increase is computed by subtracting the baseline model error from the plotted point error. In Figure 5, we study the compression ratio, and in Figure 6 we study the percentage of eliminated operations.

In Figure 5, we observe that the GRU model has the highest compression ratio at a low increase in the error. At higher error levels, the SRU models have the highest compression ratio. The LSTM and LiGRU models start with a smaller compression ratio that increases with error. However, the LiGRU model has the lowest compression ratios among the models. In Figure 6, we observe that the SRU-models have the highest EO percentage at zero increase in the error. While increasing the error, the EO increases with the GRU model, then the LSTM model, then, at higher error rates, the LiGRU shows high EO percentages. The highlighted EO results from applying the delta method to quantized activations at zero thresholds or higher thresholds.

It is observed that the SRU models have limited EO percentages. We have observed during the selection of delta thresholds in the other recurrent models that the recurrent part in the other models is less sensitive to higher delta thresholds than the input part. The recurrent part is not subjected to the delta method in SRU models. We think this has contributed to making the SRU models more sensitive to higher delta thresholds. In addition, the presence of projection between the SRU layers might have led to a different reaction to higher delta thresholds.

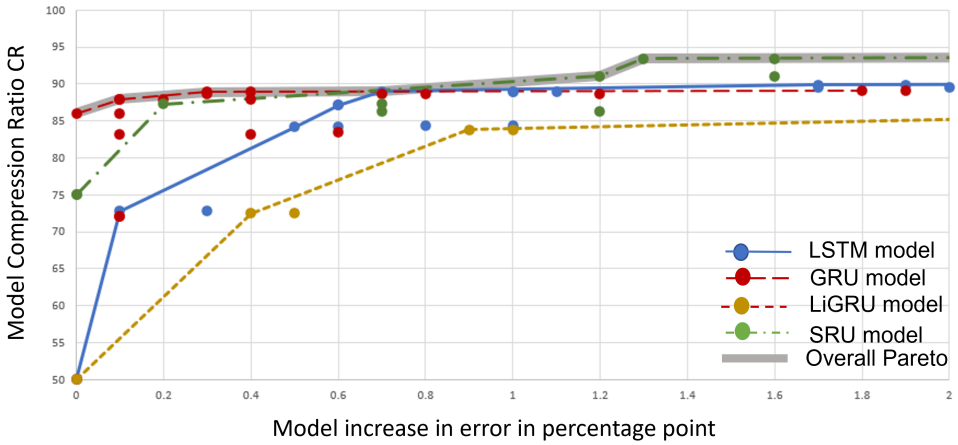

Figure 5. Compressability plot. For each model, the compression ratios for different quantization configurations are plotted against the increase in error compared to each model's baseline error. The best solutions are connected in a line. The gray shadow highlights the best points among all models.

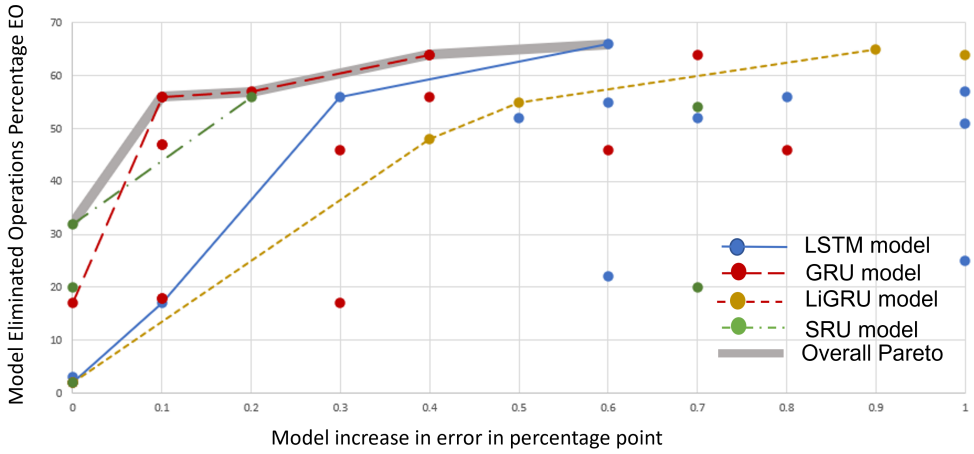

Figure 6. Eliminated operations plot. For each model, the eliminated operations (EO) percentages are plotted against the increase in error compared to each model's baseline error. The best solutions are connected in a line. The gray shadow highlights the best points among all models.

6.2. Absolute Comparison

In this section, we compare the compressed models to each other to find the best model at different error levels in terms of size (Figure 7) and number of operations (Figure 8).

In Figure 7, we observe that at low error rates the LSTM model provides the most-optimal solutions, while at higher error levels the SRU models provide the smallest-sized solutions. In Figure, 8, the LSTM model provides optimal solutions till 14.7% error rate. Afterward, the LiGRU model provides low counts of operations. The SRU model managed to decrease operations count more at a higher error rate. However, it should be noted that the SRU-model has a higher ability to be parallelized over time-steps than the three other models. Some studies have shown that, depending on the degree of parallelism, it can achieve speedup values that vary between 5× and 13× [26].

To conclude, LSTM is the highest accuracy model, and it shows good compressibility levels to both quantization and delta methods. Therefore, at very low error thresholds it is the best option. However, since LSTM models are usually larger than their alternatives, at higher error rates other recurrent models can provide better compressed solutions in terms of size and number of operations at the same error levels. The GRU unit is smaller than the LSTM unit. However, the GRU model we used had to be deeper than the LSTM model

to decrease its error. Even-though the GRU baseline model is less accurate than the LSTM baseline model, the GRU model showed very good compressibility. It can be compressed and subjected to delta thresholds with a minor increase in the error rate. However, starting as a baseline model with big size and a relatively high error rate (compared to LSTM) limited its ability to provide good solutions compared to other alternatives.

LiGRU is a lighter version of the GRU unit. Thus, its baseline model is relatively small in size. However, it has limited compressibility. Applying quantization on LiGRU resulted in a higher error increase than other recurrent models. Thus, the LiGRU model is not the best choice under high compression ratios. Still, it has an increased number of eliminated operations at the zero-delta threshold for integer activations. The SRU model's baseline has high error rates. However, it did provide incredibly small-sized solutions. Further, the possibility of parallelizing execution over multiple time-steps makes the SRU model suitable for highly efficient deployment if some error increase is acceptable.

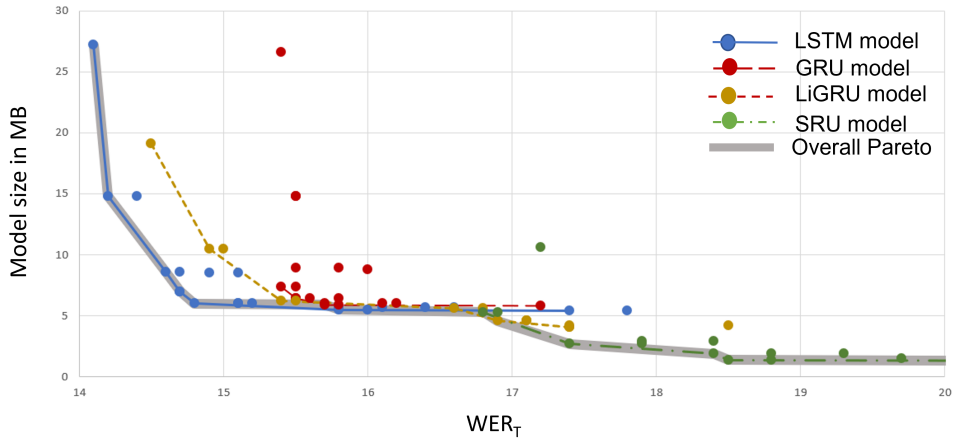

Figure 7. The size of compressed models at different error levels. For each model, we connect the best points using a line. Then we highlight the best points of all models with a gray shadow.

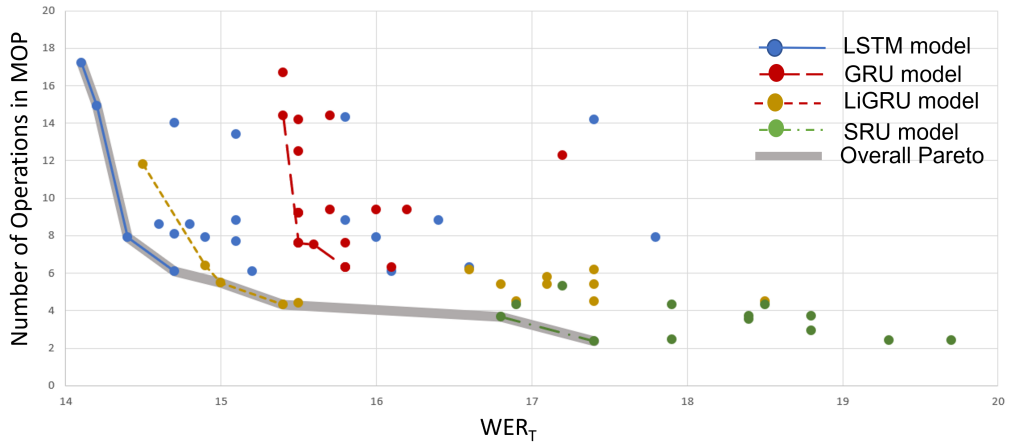

Figure 8. The number of operations in compressed models at different error levels. For each model, we connect the best points using a line. Then, we highlight the best points of all models with a gray shadow.

7. Conclusions

The craving for running large recurrent neural network (RNN) models on resource-limited edge devices makes it necessary to decrease such models' computational and memory needs. Quantization is a compression method that *shrinks* the size of the neural network model by reducing the precision of its weights. Applying quantization to activation vectors helps decrease the computational complexity of such models. Another side

effect of activation quantization to integer precisions is the increased percentage of similar values between successive activation vectors. Thus, if consecutive vectors are compared, computations corresponding to similar values can be *eliminated* without affecting accuracy. The method of eliminating computations for matching values in activation vectors is called the delta networks method. This work combines quantization and delta network methods in a post-training approach, where no retraining is required after compression. The study includes four RNN models for speech recognition using the TIMIT dataset. We skip quantization in some paths in the models that are more sensitive to quantization. Furthermore, we use a method to decide on the delta thresholds for different models to eliminate more computations with a minor effect on the error rates.

By skipping a selected path during post-training quantization, we managed to compress LSTM-, GRU-, and LiGRU-based models to 85% of their baseline size with an increase in the error of 0.6, 0, and 2.1 percentage points, respectively, compared to baseline error. By applying the delta networks method on quantized RNN models, more than 50% of the operations could be eliminated, in most cases with no or slight increase in the quantized models' error levels.

We compared the four RNN models—LSTM, GRU, LiGRU, and SRU—under the post-training quantization and delta networks method. LSTM has the lowest baseline error, and it can be highly compressed with a minor increase in the error. Thus, LSTM provides the highest compressed solutions at low error rates constraint. The GRU-based model can be compressed to almost 90% of its baseline size with negligible effect on the error. However, it could not compete with other RNN models under the same error constraints, as it started from a baseline model with an error rate higher than the LSTM-based model and a size larger than the LiGRU- and SRU-based models. On the other hand, LiGRU and SRU are light recurrent models. The LiGRU model has the highest percentage of eliminated operations at the zero-delta threshold. However, LiGRU is more sensitive to quantization and has a higher error rate when subjected to low precision quantization. SRU models, despite having a baseline model with a relatively high error rate, can be compressed to extremely low-sized models less than 2 MB.

Author Contributions: Conceptualization, N.M.R., T.N. and Z.U.-A.; methodology, N.M.R. and T.N.; formal analysis, N.M.R.; investigation, N.M.R., T.N. and Z.U.-A.; writing—original draft preparation, N.M.R.; writing—review and editing, N.M.R., T.N. and Z.U.-A.; visualization, N.M.R. and T.N.; project administration and funding acquisition, T.N., and Z.U.-A.; implementation, N.M.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research is part of the CERES research program funded by the ELLIIT strategic research initiative funded by the Swedish government.

Acknowledgments: The authors would also like to thank Eren Erdal Aksoy, Halmstad University, Sweden, for valuable discussion and comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rezk, N.M.; Purnaprajna, M.; Nordström, T.; Ul-Abdin, Z. Recurrent Neural Networks: An Embedded Computing Perspective. *IEEE Access* **2020**, *8*, 57967–57996. [CrossRef]
2. Gao, C.; Neil, D.; Ceolini, E.; Liu, S.C.; Delbruck, T. *DeltaRNN: A Power-Efficient Recurrent Neural Network Accelerator*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 21–30. [CrossRef]
3. Jo, J.; Kung, J.; Lee, S.; Lee, Y. Similarity-based LSTM architecture for energy-efficient edge-level speech recognition. In Proceedings of the 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), Lausanne, Switzerland, 29–31 July 2019; pp. 1–6. [CrossRef]
4. Rezk, N.M.; Nordström, T.; Stathis, D.; Ul-Abdin, Z.; Aksoy, E.E.; Hemani, A. MOHAQ: Multi-objective Hardware-Aware Quantization of Recurrent Neural Networks. *arXiv* **2021**, arXiv: 2108.01192
5. Banner, R.; Nahshan, Y.; Soudry, D. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems 32*; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 7950–7958.

6. Nagel, M.; Amjad, R.A.; Van Baalen, M.; Louizos, C.; Blankevoort, T. Up or down? Adaptive rounding for post-training quantization. In Proceedings of the International Conference on Machine Learning, Virtual, 13–18 July 2020; pp. 7197–7206.
7. Nahshan, Y.; Chmiel, B.; Baskin, C.; Zheltonozhskii, E.; Banner, R.; Bronstein, A.M.; Mendelson, A. Loss aware post-training quantization. *Mach. Learn.* **2021**, *110*, 3245–3262. [CrossRef]
8. Zhao, R.; Hu, Y.; Dotzel, J.; De Sa, C.; Zhang, Z. Improving neural network quantization without retraining using outlier channel splitting. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019; pp. 7543–7552.
9. Cai, Y.; Yao, Z.; Dong, Z.; Gholami, A.; Mahoney, M.W.; Keutzer, K. Zeroq: A novel zero shot quantization framework. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 13169–13178.
10. Aji, A.F.; Heafield, K. Neural Machine Translation with 4-Bit Precision and Beyond. *arXiv* **2019**, arXiv:1909.06091.
11. Chen, T.; Goodfellow, I.; Shlens, J. Net2Net: Accelerating Learning via Knowledge Transfer. In Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016.
12. Neil, D.; Lee, J.H.; Delbruck, T.; Liu, S.C. Delta networks for optimized recurrent network computation. In Proceedings of the International Conference on Machine Learning, Sydney, NSW, Australia, 6–11 August 2017; pp. 2584–2593.
13. Gao, C.; Rios-Navarro, A.; Chen, X.; Delbruck, T.; Liu, S.C. EdgeDRNN: Enabling low-latency recurrent neural network edge inference. In Proceedings of the 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Genova, Italy, 31 August–2 September 2020; pp. 41–45. [CrossRef]
14. Shan, B.; Fang, Y. DRAC: A delta recurrent neural network-based arithmetic coding algorithm for edge computing. *Complex Intell. Syst.* **2021**, 1–7. [CrossRef]
15. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
16. Cho, K.; van Merrienboer, B.; Bahdanau, D.; Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. In Proceedings of the SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014; pp. 103–111.
17. Ravanelli, M.; Brakel, P.; Omologo, M.; Bengio, Y. Light Gated Recurrent Units for Speech Recognition. *IEEE Trans. Emerg. Top. Comput. Intell.* **2018**, *2*, 92–102. [CrossRef]
18. Lei, T.; Zhang, Y.; Wang, S.; Dai, H.; Artzi, Y. Simple Recurrent Units for Highly Parallelizable Recurrence. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 31 October–4 November 2018; pp. 4470–4481.
19. Ravanelli, M.; Parcollet, T.; Bengio, Y. The pytorch-kaldi speech recognition toolkit. In Proceedings of the ICASSP 2019–2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019; pp. 6465–6469.
20. Povey, D.; Ghoshal, A.; Boulianne, G.; Burget, L.; Glembek, O.; Goel, N.; Hannemann, M.; Motlicek, P.; Qian, Y.; Schwarz, P.; et al. The Kaldi Speech Recognition Toolkit. In Proceedings of the IEEE 2011 Workshop on Automatic Speech Recognition and Understanding, Waikoloa, HI, USA, 11–15 December 2011.
21. Garofolo, J.; Lamel, L.F.; Fisher, W.M.; Fiscus, J.G.; Pallett, D.S. DARPA TIMIT Acoustic-Phonetic Continous Speech Corpus CD-ROM. NIST Speech Disc 1-1.1. 1993. Available online: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir4930.pdf (accessed on 20 February 2022).
22. Wang, S.; Li, Z.; Ding, C.; Yuan, B.; Qiu, Q.; Wang, Y.; Liang, Y. C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; pp. 11–20.
23. Sharma, H.; Park, J.; Suda, N.; Lai, L.; Chau, B.; Chandra, V.; Esmaeilzadeh, H. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, 1–6 June 2018; pp. 764–775.
24. Lee, J.; Kim, C.; Kang, S.; Shin, D.; Kim, S.; Yoo, H.J. UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision. *IEEE J. Solid-State Circuits* **2019**, *54*, 173–185. [CrossRef]
25. Sung, W.; Shin, S.; Hwang, K. Resiliency of Deep Neural Networks under Quantization. *arXiv* **2015**, arXiv:1511.06488.
26. Sung, W.; Park, J. Single Stream Parallelization of Recurrent Neural Networks for Low Power and Fast Inference. *arXiv* **2018**, arXiv:1803.11389.