AN OPTIMIZED RECURRENT UNIT FOR ULTRA-LOW-POWER KEYWORD SPOTTING

A PREPRINT

Justice Amoh
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755
justice.amoh.jr.th@dartmouth.edu

Kofi M. Odame
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755
kofi.m.odame@dartmouth.edu

February 12, 2019

ABSTRACT

There is growing interest in being able to run neural networks on sensors, wearables and internet-of-things (IoT) devices. However, the computational demands of neural networks make them difficult to deploy on resource-constrained edge devices.

To meet this need, our work introduces a new recurrent unit architecture that is specifically adapted for on-device low power acoustic event detection (AED). The proposed architecture is based on the gated recurrent unit ('GRU' – introduced by Cho et al. [9]) but features optimizations that make it implementable on ultra-low power micro-controllers such as the Arm Cortex M0+.

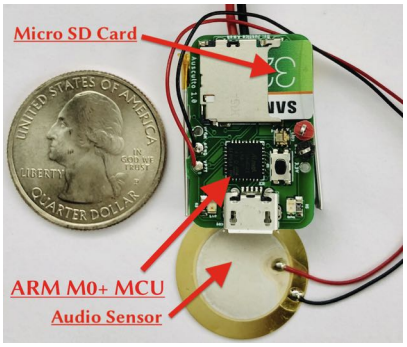
Our new architecture, the Embedded Gated Recurrent Unit (eGRU) is demonstrated to be highly efficient and suitable for short-duration AED and keyword spotting tasks. A single eGRU cell is 60× faster and 10× smaller than a GRU cell. Despite its optimizations, eGRU compares well with GRU across tasks of varying complexities.

The practicality of eGRU is investigated in a wearable acoustic event detection application. An eGRU model is implemented and tested on the Arm Cortex M0-based Atmel ATSAMD21E18 processor. The Arm M0+ implementation of the eGRU model compares favorably with a full precision GRU that is running on a workstation. The embedded eGRU model achieves a classification accuracy 95.3%, which is only 2% less than the full precision GRU.

1 Introduction

Deep neural networks are a powerful way to extract information from noisy, raw sensor data that is collected in an unconstrained real-world environment. This approach has been used successfully in computer vision [21], computer audition [19] and physiological monitoring [32]. Since deep neural networks are generally memory- and computationally-intensive, they are typically implemented on high-end cloud compute servers. However, transmitting raw data from the sensor to the cloud has negative implications for battery life [27], real-time responsiveness [14] and data privacy [37]. It also demands a reliable communications connection to the cloud which is not always possible. These challenges can be avoided by implementing the deep neural networks directly onto sensors.

Two approaches to realizing neural networks on sensors, or "edge" devices [39] are (1) customized hardware processors like the Nvidia Drive Px2 and the A11 Bionic Chip [1, 2], and (2) light-weight software libraries like uTensor and CMSIS-NN [38, 23]. Unfortunately, these approaches are inadequate for wearable applications like the recently-introduced cough-detection device (see Fig. 1) [6, 5, 4, 7]. To promote long-term use, wearable devices have stringent size requirements that translate to a small battery (often the single largest component) and a correspondingly small power budget, on the order of 10 mW. Custom neural network processors like the Jetson Tx2 would drain such a battery in less than a minute. An ultra-low-power micro controller unit (MCU) [40, 35, 34, 26] could stretch the limited power budget further, but it only provides a limited amount of memory and computational resources; even light-weight



(a) Our wearable cough detection device.

Device	Size	Power	Computations	RAM
Drive Px2	~9.6×9.2"	30,000 mW	20 TFLOPs	16 GB
Jetson Tx2	~3.5×2.5"	7,500 mW	2 TFLOPs	8 GB
iPhone X	5.6×2.8"	1,000 mW	1 GFLOPs	3 GB
Freedom K64F	3.2×2.1"	500 mW	1 MFLOPs	256 kB
Ausculto (ours)	**1.1×0.7"**	**40 mW**	**0.1 MFLOPs**	**32 kB**

(b) Table of computational resources available on various portable systems.

Figure 1: 1a Our wearable device for detecting coughs and adventitious sounds is about the size of a thumb drive. It features the Atmel ATSAMD21E18 [12], an ultra-low power Arm Cortex M0+ micro-controller. Embedding a neural network to this device is difficult because of its size and power constraints. 1b Compared to our device, portable systems that run neural networks like Nvidia's Drive and the iPhone X are larger and have enormous power requirements. Even devices like Freedom K64F recommended for uTensor light-weight library still need $10\times$ more power than available on our ultra-low power system.

libraries have some minimum hardware requirements that cannot be met by the sparse resources of an ultra-low-power MCU.

To address these challenges, we developed a novel recurrent neural network that is specifically adapted for implementation on the Arm M0+ [24] class of ultra-low-power MCUs. In particular, we introduce a new architecture for the recurrent unit which is optimized for keyword spotting tasks like wearable cough detection. This paper presents details on the architecture, training scheme and hardware implementation of the proposed recurrent cell. We also present experimental results that show our architecture requires $12\times$ less memory and $60\times$ less computational time than comparable conventional recurrent units.

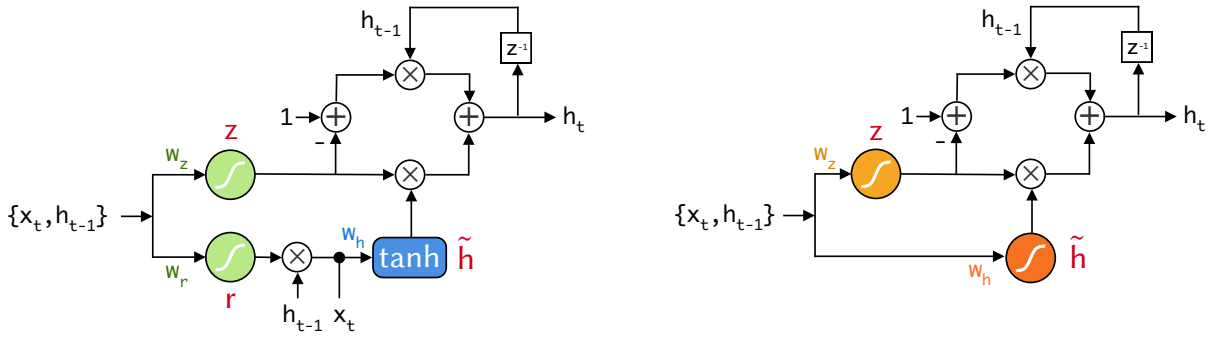
2 Related Work

Neural network optimizations similar to those proposed in this work have previously been considered in isolation by other researchers. For instance, recent efforts to optimize the GRU architecture include the removal of one of its gates. Zhou et al. [43] achieved this single gate architecture by coupling the update and reset gates into a single forget gate. The resulting 'minimal' gated unit (MGU) directly effects both cell update and reset using the same gate. Ravanelli et al. [31] discarded the reset gate altogether, highlighting that it is potentially redundant in speech recognition where input signals evolve slowly. Our work investigates this further in AED tasks where events are sudden and infrequent, in sharp contrast to speech recognition. Furthermore, our proposed cell architecture replaces the sigmoid and hyperbolic tangent activation functions with more efficient softsign variants.

Another common area of optimization is weight quantization. In previous works [18, 42], 8-bit weight quantization was shown to drastically reduce network size without significant loss in accuracy. However, these approaches focused solely on convolutional and fully-connected networks. They also required the storage of a codebook for decoding weights at run-time. With regards to recurrent networks, [28] demonstrated that 2-bit weight quantization is possible in GRU, with only a slight loss in accuracy. However, the combined effect of such a low precision quantization and other optimizations like a single gate architecture remained unknown.

Using solely integer arithmetic in neural networks has also been studied in prior works [18, 17, 8]. Courbariaux et al. [13] found that in a fully connected network, a dynamic point numeric format yields much better results than 20-bit fixed point. Another work demonstrates that 32-bit fixed point formats are effective in convolutional networks [8]. Once again, all these efforts consider fully-connected and convolutional architectures. Since recurrent architectures are fundamentally different and much more difficult to train than other architectures [29], it is worthwhile to investigate fixed point arithmetic in recurrent neural networks.

To date, no one has ever simultaneously applied all of these modifications to a GRU and validated that they yield a usable network that is implementable on a low power, low resource MCU like the Arm Cortex M0+. This work bridges that gap.



(a) GRU cell.

(b) eGRU cell.

Figure 2: Circuit diagrams illustrating GRU (2a) and eGRU (2b) cell architectures. Compared to GRU, eGRU omits the reset gate r and does not require the weights, w_r. Additionally, it replaces sigmoid and $\tanh$ activation functions with softsign variants.

3 Embedded Gated Recurrent Unit

Previous work showed that a network of gated recurrent units (GRU) [9] outperforms classical approaches like Hidden Markov Models on an acoustic event detection task when evaluated in a noisy, non-ideal environment [6]. But the GRU network described in [6] is too memory- and computationally-expensive to be implemented on a low-resource MCU like the Arm M0+.

To meet the resource constraints of a low power MCU like the M0+, we propose a new recurrent architecture: the embedded Gated Recurrent Unit (eGRU). It is based on the traditional GRU, but with four major modifications: (1) a single gate mechanism, (2) faster activation functions, (3) 3-bit exponential weight quantization and (4) fixed point arithmetic in all network operations. These modifications lead to massive reductions in memory and computations in the recurrent cell, making it feasible to run eGRU networks on our target device. Below, we discuss the features of the eGRU in detail.

3.1 Single Gate Mechanism

A practical idea for optimizing recurrent units is the removal of gates. Modern recurrent cell architectures like the Long Short-Term Memory (LSTM) unit [20] and GRU are characterized by gating mechanisms that regulate information flow in and out of the cell's memory. For instance, the LSTM cell has 3 gates: two for controlling the cell's input and output, and a third for forgetting or resetting the cell's internal state. Since gates are implemented using weights and activations, omitting a gate reduces the required memory and computations in a recurrent cell. Accordingly, GRU was introduced as an optimized form of LSTM by reducing the number of gates from 3 to 2: the update and reset gates.

In the same vein, GRU can also be optimized further by discarding yet another gate. Zhou et al. [43] accomplished this very 'minimal' gated unit by using a single gate for both resetting and updating the cell's internal state. Ravanelli et al. [31] extended that work further by highlighting a redundancy between the two gates. They deduced that in applications like speech recognition where signals change slowly, reset gates are unnecessary and can be omitted altogether. However, in applications where events of interest are abrupt and isolated (eg. detecting cough sounds), the assumption by Ravanelli et al. [31] that state resets are irrelevant does not hold. In fact, we found that without state resets, recurrent units in our application are unable to recover from large impulse signals. Thus, for keyword spotting applications, we can eliminate the reset gate and rely on only the update gate.

3.2 Activation Functions

The original GRU cell utilized two kinds of activations; sigmoid functions (σ) in the update and reset gate equations, and the hyperbolic tangent function ($\tanh$) in the state update equation. However, since the M0+ lacks a dedicated floating point unit and DSP instruction set, executing either sigmoid or $\tanh$ functions are quite slow (see Table 1b). For a more efficient recurrent unit, it was suggested in [31] that the $\tanh$ be replaced with a rectified linear unit (ReLU). Unfortunately, when combined with heavily quantized weights, recurrent cells with ReLU activations are too lossy to

GRU	**eGRU** (this work)
$z_t = \sigma(W_z \odot [h_{t-1}, x_t])$	$z_t = (\varsigma(W_z \odot [h_{t-1}, x_t]) + 1)/2$
$r_t = \sigma(W_r \odot [h_{t-1}, x_t])$	
$\tilde{h_t} = \tanh(W_h \odot [r_t * h_{t-1}, x_t])$	$\tilde{h_t} = \varsigma(W_h \odot [h_{t-1}, x_t])$
$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t}$	$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t}$

(a) State equations of traditional GRU and proposed eGRU cells.

Function	Definition	Latency
TanH	$f(x) = \tanh(x)$	328 us
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	230 us
ReLU	$f(x) = \max(0, x)$	11 us
Softsign	$f(x) = \frac{x}{1+\|x\|}$	24 us

(b) Cost of activation functions on Arm M0+.

Table 1: 1a Equations for GRU and eGRU highlights architectural differences between the two cells. In a single cell, gate weights W_r, W_z are vectors of size $N + 1$, where N is the dimension of the input x. eGRU omits the reset gate (r_t) and uses softsign (ς) activations instead of the $\tanh$ and sigmoid (σ) functions. 1b Latency of standard activation functions executed on the M0+ MCU. Sigmoid and $\tanh$ functions are $10 - 30\times$ slower than ReLU and softsign functions. However, softsign is ideal since ReLU does not work well with extremely quantized weights.

Weight	**Binary**	**Decimal**
+1.00	000	0
+0.50	001	1
+0.25	010	2
0	111	7
-0.25	110	6
-0.50	101	5
-1.0	100	4

Table 2: Encoding of quantized weights. Encoding scheme is chosen for fast computation of weight transformations using only bitwise operations in Algorithm 1. No additional look-up table is required at run-time.

ALGORITHM 1: Transformation by Quantized Weight

Input: x, value to be scaled and, w, quantized weight.
Output: Result of weight transformation, y.
$y \leftarrow 0$;
if $w = 7$ **then**
  return y;
else
  $y \leftarrow x >>$ last 2 bits of w;
  if *bit* 3 *of* $w = 1$ **then** $y \leftarrow -y$;
end
return y

learn well. And since quantization is an essential part of our proposed eGRU architecture, ReLU activations are not an option.

A desirable alternative to the activation functions above is the softsign function. In [15], softsign was shown to perform comparably with $\tanh$ and sigmoid in feed-forward networks. Although not as cheap (computationally) as ReLU, softsign is much cheaper than sigmoid or $\tanh$. A floating point implementation of the softsign function is more than 10x faster than either sigmoid or $\tanh$ functions on the M0+ (Table 1b). Furthermore, the simplicity of the softsign function permits an even faster fixed point implementation. For these reasons, we adopt the softsign and a shifted version of it as activation functions for our proposed eGRU recurrent architecture.

3.3 Weight Quantization

Efforts to reduce neural network memory footprint often involve weight quantization. As networks are purely defined by learned parameters or weights, a reduction in memory for each weight through quantization results in tremendous shrinkage in the overall network size. Several studies have shown that neural networks are still effective even after weights are quantized to only 8-bits, leading to $20 - 49\times$ memory reduction [18, 42]. Specifically in GRU architectures, ternarization (2-bit weights) has been proposed as feasible for recurrent neural networks at a small cost in performance [28].

However, we discovered that 2-bit quantization does not work well for our application. When combined with the single gate and activation function optimizations, ternarized weights in eGRU lead to poor performance. On the other hand, 3-bit quantization with septenary weights (7 levels) proved to be quite effective. Thus, 3-bit quantization was adopted for eGRU.

Besides the reduction in bits, our quantization scheme also ensures that quantized levels are negative integer exponents of two, similar to an approach in [28]. This exponential quantization enables the replacement of weight multiplications with bit shifting, which in turn drastically reduces the computation time of an eGRU cell. The septenary weights used are: $[0, \pm 0.25, \pm 0.5, \pm 1]$ and they are encoded using the mapping in Table 2 such that no external look-up tables are required at run-time. To multiply an input by a quantized weight, a simple, fast procedure featuring only bitwise operations is employed (Algorithm 1).

Operation	int32 (ns)	float32 (ns)
$\pm$	252	4,853
$<<, >>$	273	-
$\times$	253	6,578
$\div$	1,923	12,942
softsign	4,740	24,910

Table 3: Comparison of time cost of integer vs floating point arithmetic on M0+. For basic maths, integer operations are beyond $10\times$ faster than floating point ones. Also, our fixed point approximation of the softsign is $5\times$ faster than the floating point implementation.

```
int16 softsign(int32 x) {
    int32 a = x << 10;
    int32 b = (abs(x) + 32768) << 5;
    int16 y = a / b;
    return y;
}
```

Listing 1: C++ routine for softsign function in Q15. Only integer operations are used. With this routine, softsign function is correctly defined for domain of (64.0,-64.0].

3.4 Fixed Point Arithmetic

The final area of optimization in eGRU is the numeric format used for all math operations in the neural network. On better equipped processing units, single or double precision floating point formats are typically used. Unfortunately, since M0+ lacks an FPU, floating point operations are very costly (see Table 3). Hence, using solely integer operations in eGRU is desirable.

Considering the M0+ has a 32-bit architecture, optimal execution of operations is achieved when operands are contained within 32-bits. Hence, we adopt the Q15 16-bit fixed point format for all arithmetic within eGRU. In Q15 format, 16-bit signed integers from 32,767 to 32,768 are used to represent decimals in the range of [-1,1) at intervals of 2^{-15}. Necessary precautions ought to be taken during basic operations to prevent overflow and ensure closure under Q15. For instance, to divide two Q15 numbers, it is necessary to left-shift the dividend by 16 bits (yielding a 32 bit value) before undertaking the division to avoid losing precision. Hence, all eGRU operations (including activation functions) need to be translated to Q15 versions.

Weight multiplication is the most frequent operation in a neural network. Fortunately, by virtue of our exponential quantization, affine transformations for all layers in the network can be implemented by right-shift operations, which remains the same in Q15 format. The summation of all transformed inputs can exceed 16 bits and is thus accumulated in a 32 bit register. However, since the activation function is bounded by [-1,1), the output of the recurrent node remains a 16-bit Q15 number which can then be fed into yet another node. From simulations, we discovered that all inputs to an eGRU network will flow through the entire model in Q15 format and result in an output that is precise to at least 2 decimal places compared to those from an equivalent full precision (floating point) network.

An interesting modification worth mention pertains to the Q15 implementation of the softsign activation function. Inputs to activations, in the scheme described above, is a 32-bit accumulation of all transformed inputs. Since such inputs are already in 32-bit, undertaking a Q15 division would be impractical as it would require left-shifting, resulting in an overflow. One way to circumvent this is to clip the accumulated value to a certain domain to prevent overflow. We found that clipping to the domain (64,-64] resulted in a fast and accurate integer softsign approximation (see Table 3). A simple C++ definition of our softsign function is provided in Listing 1.

3.5 Summary of Optimizations

A summary of the proposed optimizations and their corresponding reductions in computational resource requirements is provided in Table 4. Table 4a presents a breakdown of the resource requirements for an eGRU cell is compared with that for a traditional GRU cell. The overall number of operations are fewer for eGRU (12) than for GRU (17). Furthermore, eGRU trades off most multiplications for cheaper and faster bit shift operations. Also, note that all eGRU operations are integer operations (INOPs) compared to the floating point operations (FLOPs) in GRU.

Memory and latency reduction scale factors for each optimization is also listed in Table 4a. Quantization mostly accounts for the miniaturization, reducing the cell size by a factor of 10. On the other hand, using solely Q15 integer operations rather than floating point operations drastically speeds up cell execution and yields a $20\times$ reduction in latency. Analytically, softsign activations also account for significant reduction in latency ($10\times$), even without accounting for the additional gains in speed afforded by the faster integer softsign approximation in Listing 1.

3.6 Training Setup

Since the sole interest is inference (not learning) on target devices, training of eGRU cells can be undertaken on high-end computers with enormous resources, using cutting-edge deep learning libraries like PyTorch [30] and Tensorflow [3]. However, due to the modifications proposed above, additional steps are still necessary to effectively train eGRU cells

Requirement	eGRU	GRU
# of Parameters	6	9
Additions	6	8
Multiplications	2	9
Bit Shifts	4	-

(a) Comparison of cell requirements.

Optimization	Memory	Latency	Modification
Single Gate Unit	1.5×	1.5×	3 gates ⇒ 2 gates
Quantization	10×	-	32 bits ⇒ 3 bits
Softsign Activations	-	10×	σ & $\tanh$ ⇒ softsign
Q15 Arithmetic	-	20×	FLOPs ⇒ INOPs

(b) Reduction scale factors for individual optimizations.

Table 4: Summary of computational requirements and savings of an eGRU cell compared to GRU. 4a Comparison of number of parameters and operations required for each cell type. eGRU's fewer parameters decreases required memory storage and its fewer operations reduces execution latency. 4b Tabulation of memory and latency reductions afforded by proposed optimizations. Quantization achieves the most memory reduction by a factor of 20, whereas Q15 integer implementation attains the most speed up at a 10× reduction in latency.

in a neural network. Specifically, the effects of quantization and fixed point arithmetic in eGRU need to be simulated during the training process as well.

One common approach to quantize neural network weights is by means of post-training step. The network is first trained with full-precision weights (32- or 64-bit floating point numbers), then learned weights are quantized to the desired precision. This approach requires no adjustments to the training process since quantization is applied afterwards. While this method is suitable for reducing model sizes, it is not robust especially with heavy quantization like ours. That is because during training, the network is oblivious to the post-training quantization and therefore cannot learn to accommodate it.

A more desirable approach is to implement the quantization process as part of the network's computational graph so it can be incorporated into the training process. Yet, since quantization is discontinuous, it cannot be incorporated in backpropagation training without adjustments. To address this, we adopt an approach similar to that used by Hubara et al. [22]. During forward pass of backpropagation, weights are first cached and passed through a quantization function before they are used in the dot product. Then during backward pass, gradients are computed with respect to the stored full-precision weights.

An illustration for this is provided in Fig. 3 and the quantization function used is given by:

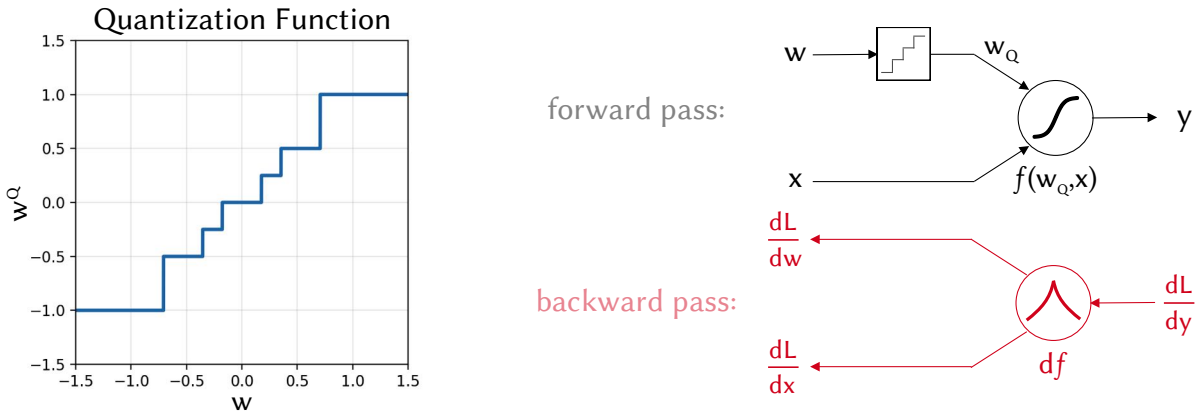
$$W_Q = \begin{cases} +1, & W \geq +1 \\ 0, & |W| \leq 0.25 \\ -1, & W \leq -1 \\ \text{sgn}(W) \cdot 2^{\lfloor (\log_2 |W|) \rceil} & \text{otherwise} \end{cases}$$

where W is the full-precision weight, and W_Q is the quantized weight. In a sense, quantization is realized just like weight clipping (where weights are restricted to a defined range), and can thus be similarly considered a regularization [25]. Note though, that once the network is trained, full-precision weights are discarded and only quantized weights are used in inferencing.

Besides quantization, simulating eGRU's fixed point arithmetic also requires training adjustments. The Q15 fixed point format used can only represent decimal numbers in the range (1,-1]. Therefore, during training, all network operations have to be constrained to fit within these Q15 bounds, in order to avoid overflow on the target device. In our work, we achieve this by clipping all dot products and activations to the (1,-1] range. The weight clipping here, as alluded to above, serves as yet another regularizer for eGRU networks.

4 Experimentation

To evaluate the proposed features of eGRU, four experiments are undertaken. The first experiment investigates the benefits of our architectural choices, mainly the single gate mechanism and the activation functions. The second experiment analyzes the isolated impact of the various optimizations on the performance of a recurrent network. The third experiment benchmarks the computational speed of an eGRU cell on an M0+ in contrast to the GRU and RNN. Finally, the last experiment evaluates the performance of a fully optimized eGRU neural network embedded onto the target low power device for acoustic event detection tasks. The tasks, network architecture and experimental setups are discussed below in detail.



(a) Exponential quantization function.

(b) Quantization in backpropagation.

Figure 3: Modifications to include quantization in training process. 3a Quantization function for yielding septenary weights that are negative integer exponents of 2. 3b During forward pass, quantization is applied to weights before they are used in any transformation. In backward pass, gradients are computed with respect to stored full precision weights.

4.1 Task Description

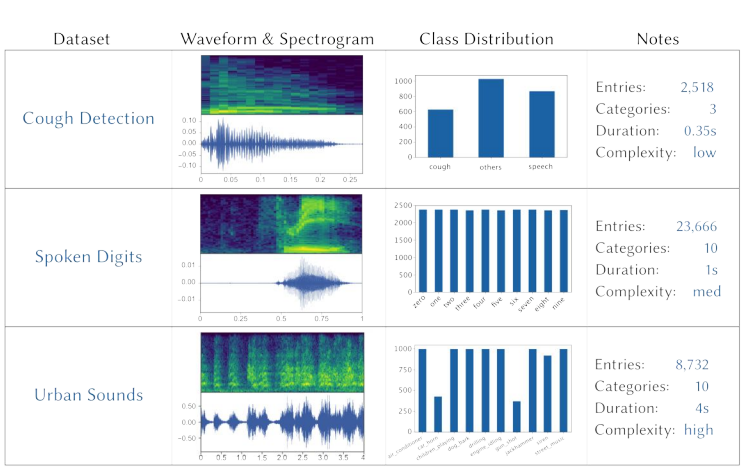
To evaluate the various recurrent cells, three Acoustic Event Detection (AED) tasks of varying complexity are considered. They are as follows:

- **Cough Detection:** The first and least complex of the three task is the classification of short acoustic events as cough, speech or other non-cough sounds. Other non-cough sounds consists of voiced non-speech sounds, ambient sounds and noise (white and pink). This task is actually the application of interest for our wearable device. Since there are only three possible classes, the problem is much easier than the other tasks. Furthermore, because cough sounds have a relatively short duration (350 ms on average), this problem would not require that much long-term memory compared to the others. The dataset used here is from a prior work [6], and consists of 2,518 audio sounds. The dataset is not exactly balanced and contains more speech and other sounds than actual cough events.

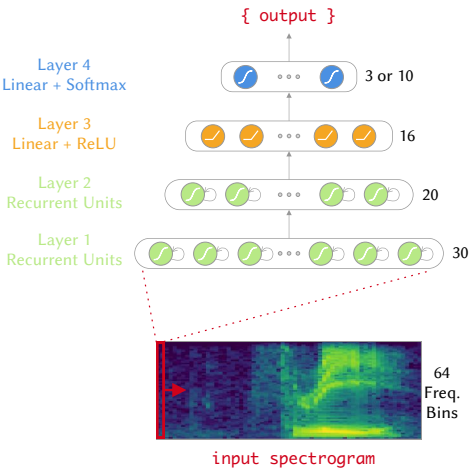
- **Spoken Digits:** The next task involves distinguishing between ten spoken digits (numbers 0 to 9). The audio data is obtained from Google's Speech Commands Dataset [41] which comprises of audio recordings covering 20 simple commands. In this work, we use a subset of that dataset, consisting of 23,666 one-second audio clips. There are ten digits and the dataset is well balanced, with roughly 2,400 examples per digit. On one hand, this task can be considered more complex than the cough detection problem because there more classes and audio samples are roughly 3× longer. On the other hand, this dataset is 10× larger and thus should be easier to learn.

- **Urban Sounds:** The third and most challenging of the tasks considered involves identifying different urban sounds. The UrbanSound8k dataset is used [33] and it includes 8,732 sounds recorded from 10 urban environments. Thus, there are ten different classes in this problem as well, and some examples are: *children_playing*, *dog_barking*, *car_horn*, *siren* and *street_music*. Certain classes sound very similar and it can be difficult for even a human to accurately identify some entries. Moreover, entries have a maximum duration of 4 seconds, which is much longer than the other two tasks and would therefore require larger long-term memory capability.

A summary of the three tasks is provided in Fig. 4a. Audio from all datasets are pre-processed in a similar manner. First, recordings are downsampled from their original sampling rates to 8 KHz, then further processed into spectrograms using the Short-Time Fourier Transform (STFT). For the STFT, a window length of 128 samples is used, with no zero-padding or overlap. These analysis configuration are motivated by what can feasibly be computed on the M0+ MCU in a real-time application. For each entry in the corresponding dataset, the spectral analysis yields a 64×24 spectral chunk (64 frequency bins and 24 time steps) for the Cough task, 64×64 for Spoken Digits and 64×250 for Urban Sounds.



(a) Dataset and tasks for training neural networks.



(b) Recurrent neural network architecture.

Figure 4: 4a Summary of the three audio event detection (AED) datasets and tasks investigated in experiments. Cough detection is easiest because samples are relatively short and it is a 3-class problem. Urban sounds are 10× longer and consist of 10 different classes. 4b An illustration of the neural network architecture used for all tasks. Recurrent units in the first two layers are RNN, eGRU or GRU cells. The input is a sequence of FFT vectors.

4.2 Network Architecture

For all three tasks, the same recurrent neural network architecture is used. Though better performing architectures can be used for each task, fixing the architecture makes for a fair comparison across tasks and experiments. Also, it is also common practice to use similar network architectures when introducing new recurrent units to enable direct comparison of architectural benefits [10, 11]. In our case, we are primarily interested in the differences in performance for a drop-in replacement of all GRU cells with eGRU. Thus, it is preferred to keep the same network architecture for both eGRU and GRU cells. A four layer network is used, featuring two recurrent layers with 30 and 20 corresponding recurrent units (eGRU, GRU, or RNN cells). This is preceded by a 16 node linear layer with ReLU activations. The fourth and final layer is the classification layer with task specific number of output nodes and preceded by a softmax function. An illustration of this neural network architecture is provided in Fig. 4b.

The neural networks are implemented and trained using PyTorch on a computer with 64GB RAM, a 3.8 GHz Core i5 CPU and an Nvidia GTX 1080 Ti GPU. The ADAM optimizer is used, with mini-batch sizes of 128. Training is undertaken for a maximum of 200 epochs with checkpointing at each epoch to retain only the best performing model state. In checkpointing, if the network's loss on a validation set is the minimum seen so far, the network's state is saved as the current best model. Model checkpointing therefore enables training for much longer epochs without the risk of overfitting and ensures the best model state is used eventually for evaluation.

4.3 Experiment 1

In the first experiment, we seek to answer the following question: how does the proposed eGRU architecture compare with traditional recurrent architectures on acoustic event detection tasks? Of particular interest here is the performance of the single gate mechanism together with the softsign activation functions because those are distinctive features of the eGRU. Quantization and integer arithmetic, on the other hand, can be applied to any recurrent cell and are therefore not considered in this experiment.

To evaluate eGRU's architecture, RNN, GRU and eGRU networks are trained on the three datasets using the network architecture in Fig. 4b. While eGRU is a single-gate version of the GRU, the vanilla RNN is in a sense an even simpler, "zero-gate" GRU. We include the RNN in this experiment to explore whether gated models like the GRU or eGRU are even necessary for our application. A 10-fold cross-validation scheme is adopted for a thorough evaluation. On each task, classification accuracies on test sets are computed for the different cell types and aggregated across all folds. In all, it takes about 5 days to train all models across the ten folds.

4.4 Experiment 2

Next, the second experiment explores how the proposed optimizations, in isolation, impact performance of a GRU network. To that end, four variants of the network architecture in 4b are implemented featuring: (1) full precision original GRU cells, (2) single gated recurrent cells (3) 3-bit quantized weights in GRU cells and linear layers and (4) softsign activations in GRU cells. These networks are to illustrate the performance costs of individual optimizations. However, integer arithmetic is not included in this analysis because it mainly affects run-time speed on the embedded system and has little effect on network performance in terms of accuracy.

All four networks are trained on the above mentioned AED datasets. Also, a 70% : 30% training and test set split is used rather than the 10-fold cross-validation scheme used in the first experiment. From the 70% subset of the dataset used for training, 25% is further set aside as the validation set to determine when training converges.

4.5 Experiment 3

The third experiment aims to measure the latency of the each recurrent cell type on the Arm M0+ MCU. Measuring latency will reveal and quantify the gains in speed and memory that eGRU cell has over the GRU cell. Consequently, single eGRU and GRU cells are implemented and benchmarked on the target M0+ device. The eGRU cell here features all optimizations proposed including quantization and fixed point arithmetic.

For each recurrent cell type, a sequence of 1000 one dimensional inputs ranging from -1.0 to +1.0 are iteratively fed to the cell. The time taken for each cell to process the sequence on the M0+ is measured and reported as the latency benchmark. Note that while the GRU cell and its activations are implemented in 32-bit floating point format, eGRU is implemented solely in 16-bit integer data types. The latency benchmarks provide a concrete measure of how much faster and smaller eGRU is compared to GRU on an M0+ MCU.

4.6 Experiment 4

In the fourth and final experiment, the goal is to evaluate the performance of eGRU models trained on the three datasets and deployed to the M0+ for inference. For each dataset, a fully optimized eGRU model with the same architecture as in Fig. 4a is trained. Since deployment is the aim, eGRU cells used here feature all optimizations: single gate architecture, 3-bit quantization, softsign activations and a Q15 implementation. The same 70% : 30% cross-validation scheme used in experiment 2 is also adopted here.

Once the network is trained, quantized network weights are included in the firmware of the target M0+ device. At test time, test audio samples are fed to the M0+ device via a USB serial communication for processing and classification. The entire audio classification by the eGRU model takes place on the device, which then returns a prediction afterwards. Predictions are aggregated and analyzed to obtain performance metrics. Also, the memory, computations and speed of the embedded model are analyzed. For comparison, an equivalent GRU network is implemented on a workstation computer and evaluated using the same train and test sets.

Cell	CoughDetect	SpokenDigits	UrbanSounds
RNN	88.0 $\pm$ 2.1	34.2 $\pm$ 5.7	51.0 $\pm$ 2.4
eGRU	**92.1** $\pm$ **1.7**	**92.3** $\pm$ **0.5**	**74.3** $\pm$ **2.1**
GRU	**92.5** $\pm$ **1.9**	**93.1** $\pm$ **0.6**	**76.2** $\pm$ **2.7**

Table 5: Experiment 1 results reporting average classification accuracies (%) for the recurrent cells across 10-folds in three AED tasks. The eGRU architecture performs comparably with GRU across all tasks.

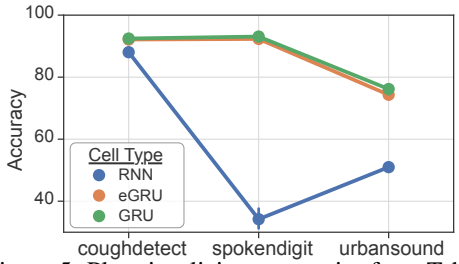


Figure 5: Plot visualizing accuracies from Table 5. Tasks have varying complexities. Specialized cells perform better than RNN.

5 Results

5.1 Experiment 1

Table 5 and Fig. 5 report the classification accuracies across 10-folds for the various recurrent cells in the first experiment. As expected, performance in general drops as the sequence length and task complexity increases across tasks. Cough detection, with a short sequence length of 24, is shown to be the easiest task where even RNN does well. On the other hand, in urban sound identification where sequences are $10\times$ longer, even the gold standard GRU suffers a notable decrease in performance.

In all tasks, the baseline RNN cell significantly under-performs both GRU and eGRU cells, especially in the more complex tasks. This trend is observed in the accuracy scores in Table 5 and in the per-dataset box-plots and validation curves in Fig. 6-8. Counter to expectations, RNN actually performed worse on spoken digits classification rather than the longer, seemingly more complex urban sounds identification. A possible explanation for this observation is that although audio entries from the spoken digit dataset are one second long, the actual event contained is often much shorter than that as evident in Fig. 4a. Since recordings begin and end with long periods of silence or background sounds, traditional RNN cells lacking advanced memory capacity will struggle to identify the buried event. The validation curves in Fig. 7 attests to this, showing the RNN cells' inability to learn much in spoken digits task. Such behavior is not observed in the more advanced GRU and eGRU cells. In all, the under-performance of RNN cells confirms that gated recurrent architectures like eGRU and GRU are indeed beneficial for the AED tasks in question.

The eGRU model performs consistently well across all tasks, closely following GRU's performance. Even in the difficult urban sounds task, eGRU only suffers a 2% loss in accuracy relative to GRU. The same trend is observed in the training curves in Fig. 6-8, where eGRU follows closely behind GRU. That eGRU performs decently even on longer sequences is interesting because one would expect the lack of a reset gate to greatly inhibit the cell's long-term memory. However, it turns out that with full precision weights, eGRU can indeed accomodate longer sequences.

From the observations mentioned above, the proposed eGRU architecture, mainly the single gating mechanism, appears to be suitable for AED tasks. eGRU is certainly preferable to RNN and a good alternative to GRU in keyword-spotting applications.

5.2 Experiment 2

The ablation study results are presented in Fig. 9. In Fig. 9a, classification accuracies for isolated optimizations are compared to that of a baseline GRU without any optimization. In all three AED tasks, 3-bit quantization impacts network performance the most, resulting in 1-20% reduction in accuracy. The effects of the single gate and softsign activations are not as pronounced except for the harder urban sounds task. In the longer urban sounds task, softsign activations evidently hurt performance (by $\sim$6%) whereas single gate architecture appears to slightly help.

The validation curves on the spoken digits tasks provided in 9b also reveal how the individual optimizations influence the network training. From the curves, all optimizations are seen to make training more difficult, taking longer time to converge than the original GRU. Once again, 3-bit weight quantization has the most adverse effect on training, taking much longer to converge and exhibiting noisy performance due to the extremely low precision weights. Softsign activations also makes it harder to train, requiring 20 more epochs to catch up to the original GRU in accuracy. The single gate optimization impacts training the least.

5.3 Experiment 3

The results for the third experiments are provided in Table 6 and it informs on the computational costs of the different recurrent cells. Evidently, eGRU's use of solely integer operations result in much faster execution on the M0+ compared to GRU. On the M0+, an eGRU cell takes only 13 ns to execute, which is more than $60\times$ faster than GRU. Besides speed, the extreme 3-bit quantization of eGRU results in a $12\times$ reduction in memory compared to GRU (with 32-bit floats). A single eGRU cell is only 3 bytes large.

5.4 Experiment 4

Finally, in Fig. 10, the performance of fully optimized eGRU models trained for the different tasks and embedded on the target M0+ device is presented. Note that while eGRU models here feature all optimizations proposed including quantization and fixed point operations, GRU models have full precision weights and employ floating point operations. Also, because a different validation scheme is used, the results for the GRU models here are a bit different than those in the first experiment.

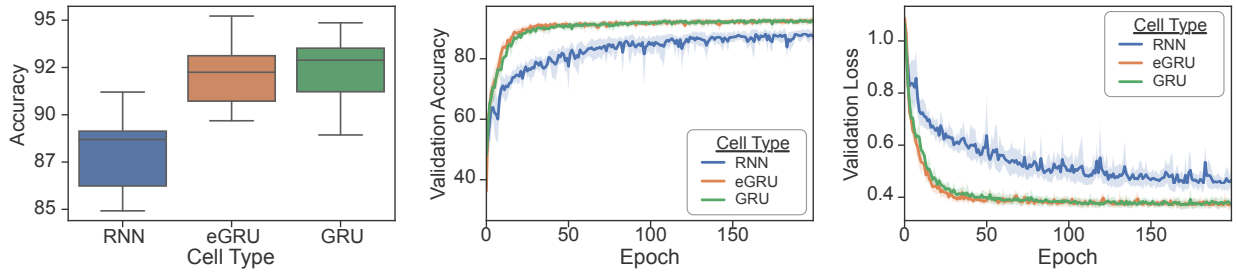


Figure 6: Cough detection task performance and training curves.

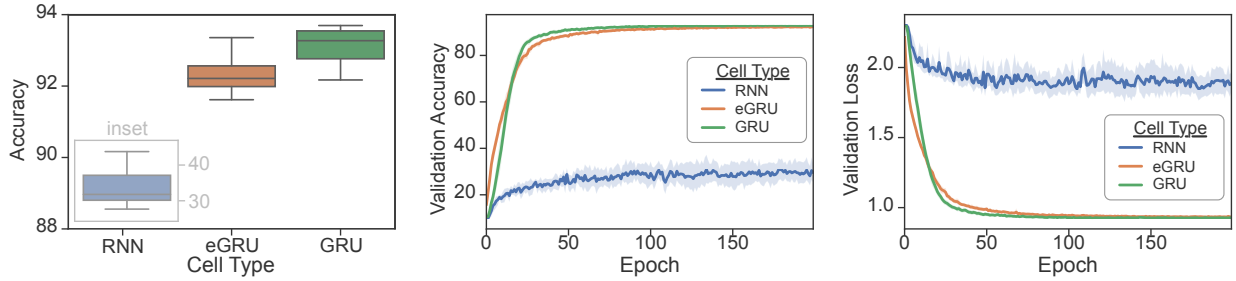


Figure 7: Spoken digits task performance and training curves.

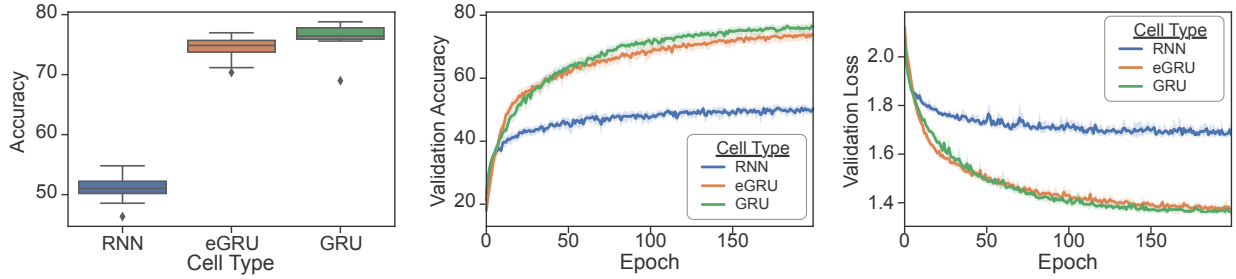


Figure 8: Urban sounds task performance and training curves.

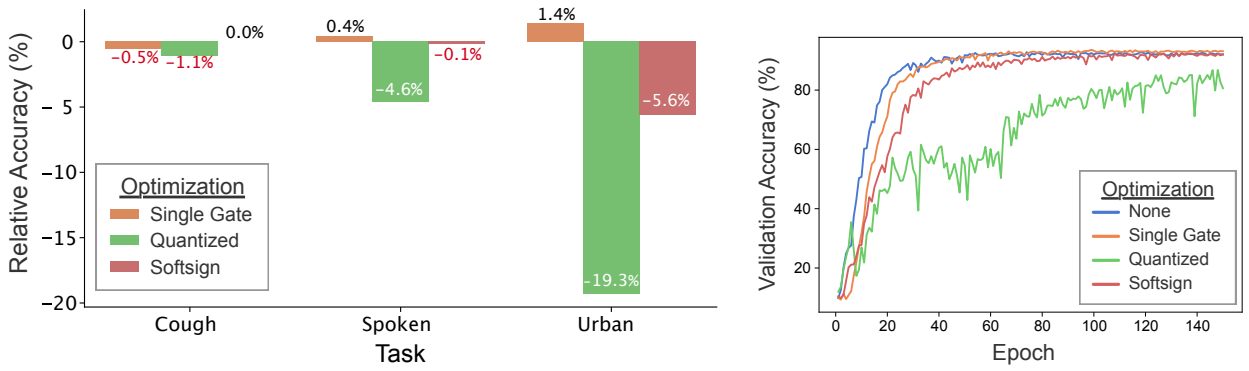
Cell	Operations	Memory	Latency
eGRU	12 INOPs	**3 bytes**	**13 ns**
GRU	17 FLOPs	36 bytes	794 ns

Table 6: Experiment 3 results showing benchmark of M0+ latency for eGRU versus GRU cells. An eGRU cell requires only integer operations (INOPs) instead of the more expensive floating point operations (FLOPs) in traditional GRU. On the M0+, eGRU is **60**× **faster** and **12**× **smaller** than GRU.

On the embedded system, the eGRU model attains an accuracy of 95.5% on the cough detection task, which is only 2% less than the accuracy of a corresponding GRU model on a fully equipped desktop computer. This is impressive considering the eGRU model is only of 3kB in size, 10× smaller than its GRU equivalent. From latency benchmarks, the embedded eGRU model is measured to take only 9.3 ms to analyze and classify a 350 ms (64 x 24 spectrogram) cough event.

On the spoken digits identification, the eGRU model running on the M0+ still compares well with the GRU. It attains an accuracy of 87.8% compared to 91.8% of GRU. That said, the more complex spoken digits task with longer sequences widens the performance difference between eGRU and GRU at 4%.

Similar to the first experiment, both recurrent cell types perform the worst on the urban sounds dataset. The deployed eGRU model achieves a 61.3% accuracy, compared to 72.1% of GRU. Here, the eGRU model under-performs GRU by a significant 11%. However, that is not very surprising considering the input sequences in this task are 4-10× longer than in the other two tasks.



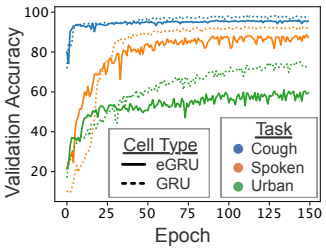
(a) Bar chart comparing performance of individual optimizations.

(b) Validation curves for spoken digit task.

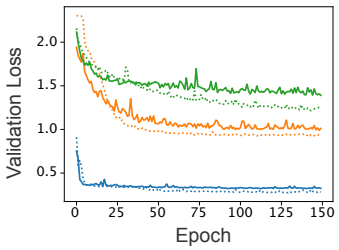
Figure 9: Experiment 2 results analyzing the isolated impacts of proposed eGRU optimizations on performance. 9a Relative classification accuracies for optimizations with respect to original GRU, across three AED tasks. 3-bit quantization affects accuracy the most, resulting in up to 19% reduction in accuracy. 9b Corresponding training curves for different optimizations in spoken digit task. All proposed optimizations make networks harder to training, requiring more epochs to converge compared to original GRU.

	eGRU	**GRU**
CoughDetect	95.3%	97.3%
SpokenDigits	87.8%	91.8%
UrbanSounds	61.3%	72.1%
Model Size	3 kB	34 kB
Device	Arm M0+	Intel Core i5
Speed	48 MHz	3,800 MHz

(a) eGRU deployed on Arm M0+ versus GRU running on Intel workstation.



(b) Validation accuracy curves for eGRU versus GRU models.



(c) Validation loss curves for eGRU versus GRU models.

Figure 10: Experiment 4 results showing performance of eGRU model embedded unto the Atmel ATSAMD21 [12], an ultra low-power Arm M0+ processor. 10a The eGRU model on the M0+ compares well with its full precision GRU counterpart on a computer, across the different task. Yet, the embedded eGRU model is highly efficient, taking only 10% the size of GRU and requiring inexpensive integer operations exclusively. 10b Validation accuracy curves for eGRU and GRU models across all task. eGRU takes longer epochs to train than GRU. 10c Corresponding validation loss curves. eGRU models approach similar loss in short-duration tasks.

6 Discussion

The results from the first experiment (Fig. 5 and Fig. 8) confirm that the proposed architectural modifications in eGRU are indeed effective for short acoustic event detection or keyword spotting tasks. In particular, single gate architecture with softsign activations is seen to be effective for short sequences, and even for longer ones. From a purely architectural stand-point, eGRU is seen to be certainly better than traditional RNN and an efficient candidate for replacing GRU cells.

With quantization and fixed point arithmetic, eGRU is demonstrated to be efficient enough to fit and run successfully on a resource constrained MCU like the M0+. From the second and third experiments, eGRU is shown to be 10× smaller and 60× faster than GRU. Yet, in short sequence tasks, eGRU only suffers 2-4% reduction in accuracy relative to GRU. Indeed, on more complex AED tasks with much longer sequences such as the urban sounds identification task, fully optimized eGRU under-performs GRU even more. The combination of extreme quantization as well as the single gate architecture appears to heavily impair eGRU's ability to handle long sequences. Thus, eGRU is most suitable for short-duration acoustic event detection applications such as keyword spotting.

One way to make eGRU effective even for long sequences is to use a less aggressive quantization scheme. As noted in experiment 1 results, full precision eGRU performs well on long sequences even with the single gate architecture. Thus, increasing weight precision from 3 to 4 or even 8 bits will considerably improve accuracy on long events. Another idea for tackling for lengthy sequences is to use eGRU in a sequence-to-sequence network architecture [36] with a sequential

loss function such as the connectionist temporal classifier (CTC) [16]. In the sequence-to-sequence with CTC setup, rather than reading an entire input sequence and outputting a single prediction as done above, recurrent networks yield sequential outputs that are aligned with the input. Sequence-to-sequence network architectures are more suitable for long sequences and is typically used in speech recognition tasks.

However, the significance of eGRU is crucial in ultra-low power wearable applications where resources are limited. For instance, for our wearable cough detector application, the 3 kB eGRU model can easily fit within the 32 kB RAM available on the M0+, with a lot of space left over for other processes. Also, the 9 ms latency can be easily managed on the MCU since cough events are about 350 ms on average. In contrast, the equivalent GRU model requires 34 kB memory and up to 500 ms latency, both of which will not be feasible on the M0+. In such applications, eGRU is the only practical choice.

7 Conclusion

In summary, our work explores a means to realize recurrent neural network inferencing on ultra-low power wearable devices. We discussed the constraints of ultra-low-power devices in terms of size, power, memory and computation specifications. We then presented a new recurrent unit architecture, the embedded Gated Recurrent Unit (eGRU), specifically adapted for ultra-low power micro-controller implementation. In particular, eGRU featured a single gate mechanism, faster activation functions, 3-bit weight quantization and utilized exclusively fixed point arithmetic. From our experiments, we demonstrated that while eGRU can be feasibly implemented on target ultra-low power devices, it still performs comparably with GRU on short-duration acoustic event detection applications.

Two main areas for further investigation remain. First, since final eGRU network only utilized a fraction (10%) of the available RAM on the M0+, better, bigger and even more complex network architectures can be explored. It will be interesting and beneficial to incorporate recent neural network innovations such as batch normalization, attention mechanisms, and skip/residual connections in the models.

A more pressing next step is to prepare the entire system to run effectively in a real life scenario, untethered and robust to completely unexpected events. Currently, since the datasets used were all created under controlled environments, the model does not perform quite as well on events that are very different from those in the datasets. For the system to be completely untethered, the neural network model has to be combined with other real-time processing steps such as signal acquisition, voice activity detection and spectral decomposition, all running on the ultra-low power micro-controller.

References

[1] Autonomous car development platform from nvidia drive px2. URL `https://www.nvidia.com/en-au/self-driving-cars/drive-px/`. Accessed: 2018-09-17.

[2] iphone 8 and iphone 8 plus: A new generation of iphone, September 2017. URL `https://www.apple.com/newsroom/2017/09/iphone-8-and-iphone-8-plus-a-new-generation-of-iphone/`. Accessed: 2018-09-17.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[4] Justice Amoh and Kofi Odame. Technologies for developing ambulatory cough monitoring devices. *Critical Reviews™ in Biomedical Engineering*, 41(6), 2013.

[5] Justice Amoh and Kofi Odame. Deepcough: A deep convolutional neural network in a wearable cough detection system. In *Biomedical Circuits and Systems Conference (BioCAS), 2015 IEEE*, pages 1–4. IEEE, 2015.

[6] Justice Amoh and Kofi Odame. Deep neural networks for identifying cough sounds. *IEEE transactions on biomedical circuits and systems*, 10(5):1003–1011, 2016.

[7] Justice Amoh, Teresa Y Ou, Malika Khurana, Vibhu Yadav, and Kofi M Odame. Mobile system for self-monitoring of asthma. In *Engineering in Medicine and Biology Conference (EMBC), 2014 IEEE*.

[8] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. DaDianNao: A Machine-Learning Supercomputer. *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, 2015-Janua(January):609–622, 2015. ISSN 10724451. doi: 10.1109/MICRO.2014.58.

[9] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine

Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014. doi: 10.3115/v1/D14-1179. URL http://arxiv.org/abs/1406.1078.

[10] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[11] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*, 2016.

[12] Atmel Corporation. Atsamd21e datasheet, 2015. URL https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/Atmel-42181-SAM-D21_Datasheet.pdf.

[13] Matthieu Courbariaux, Yoshua Bengio, and Jean-pierre David. Training deep neural networks with low precision multiplications. (Section 5):1–10, 2014. ISSN 10495258. doi: arXiv:1412.7024. URL http://arxiv.org/abs/1412.7024.

[14] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. Ieee, 2010.

[15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[16] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[17] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep Learning with Limited Numerical Precision. 37, 2015. ISSN 19410093. doi: 10.1109/72.80206. URL http://arxiv.org/abs/1502.02551.

[18] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. pages 1–14, 2015. ISSN 01406736. doi: abs/1510.00149/1510.00149. URL http://arxiv.org/abs/1510.00149.

[19] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1–32, 1997.

[21] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[22] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. pages 1–29, 2016. ISSN 1664-1078. URL http://arxiv.org/abs/1609.07061.

[23] Liangzhen Lai, Naveen Suda, and Vikas Chandra. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. pages 1–10, 2018. URL http://arxiv.org/abs/1801.06601.

[24] Arm Ltd. Cortex-m0 – arm developer. URL https://developer.arm.com/products/processors/cortex-m/cortex-m0-plus. Accessed: 2010-09-30.

[25] Paul Merolla, Rathinakumar Appuswamy, John Arthur, Steve K Esser, and Dharmendra Modha. Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv preprint arXiv:1606.01981*, 2016.

[26] *20-Pin Flash, 8-Bit Microcontrollers with XLP Technology*. Microchip Technology, 10 2015. Rev. 1.0.

[27] Antti P Miettinen and Jukka K Nurminen. Energy efficiency of mobile clients in cloud computing. *HotCloud*, 10: 4–4, 2010.

[28] Joachim Ott, Zhouhan Lin, Ying Zhang, Shih-Chii Liu, and Yoshua Bengio. Recurrent Neural Networks With Limited Numerical Precision. pages 1–11, 2016. URL http://arxiv.org/abs/1611.07065.

[29] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[31] Mirco Ravanelli, Philemon Brake, Maurizio Omologo, and Yoshua Bengio. Improving speech recognition by revising gated recurrent units. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, volume 2017-Augus, pages 1308–1312, 2017. doi: 10.21437/Interspeech.2017-775.

[32] Daniele Ravı, Charence Wong, Fani Deligianni, Melissa Berthelot, Javier Andreu-Perez, Benny Lo, and Guang-Zhong Yang. Deep learning for health informatics. *IEEE journal of biomedical and health informatics*, 21(1): 4–21, 2017.

[33] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1041–1044. ACM, 2014.

[34] *Ultra Low Power 128K, LCD MCU Family*. Silicon Labs, 7 2013. Rev. 1.0.

[35] *Ultra-low-power value line Arm Cortex-M0+ MCU*. STMicroelectronics, 9 2018. Rev. 2.0.

[36] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[37] Hassan Takabi, James BD Joshi, and Gail-Joon Ahn. Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy*, (6):24–31, 2010.

[38] Neil Tan. uTensor: AI Inference library based on mbed and TensorFlow. \url{https://github.com/uTensor/uTensor}, 2017.

[39] Surat Teerapittayanon, Bradley McDanel, and HT Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *International Conference on Distributed Computing Systems (ICDCS), 2017 IEEE*, pages 328–339. IEEE, 2017.

[40] *Ultra-Low-Power Mixed-Signal Microcontrollers*. Texas Instruments, 20 2018. Rev. N.

[41] Pete Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. 2018. URL `http://arxiv.org/abs/1804.03209`.

[42] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized Convolutional Neural Networks for Mobile Devices. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4820–4828, 2016. ISSN 10636919. doi: 10.1109/CVPR.2016.521. URL `http://ieeexplore.ieee.org/document/7780890/`.

[43] Guo Bing Zhou, Jianxin Wu, Chen Lin Zhang, and Zhi Hua Zhou. Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, 13(3):226–234, 2016. ISSN 17518520. doi: 10.1007/s11633-016-1006-2.