OCEAN: an On-Chip Incremental-Learning Enhanced Processor with Gated Recurrent Neural Network Accelerators

Chixiao Chen[1,3], Hongwei Ding[1], Huwan Peng[1], Haozhe Zhu[1], Rui Ma[1], Peiyong Zhang[2], Xiaolang Yan[2], Yu Wang[1], Mingyu Wang[1], Hao Min[1] and Richard C.-J Shi[1,3]
[1]Brain Chip Research Center, Fudan University, Shanghai, PR China, 201203
[2]Institute of VLSI Design, Zhejiang University, Hangzhou, PR China, 310027
[3]Department of Electrical Engineering, University of Washington, Seattle, WA, 98195, USA
Email: cxchen@fudan.edu.cn

***Abstract*—A deep learning processor with 8 gated recurrent neural network (RNN) accelerators is proposed in this paper. It features on-chip incremental learning by numerical and local gradient computation enhancement. Extra precision of training is obtained without extending the bit-width. Tri-mode weight access (DMA/FIFO/RAM) improves the throughput during incremental learning. The number multipliers and activation function engines are reduced by hardware sharing. The processor is fabricated in 65nm CMOS, consumes 155mW at 400MHz /1.2V or 6.6mW at 20MHz /0.8V. It achieves a peak throughput of 54.2Kfps and energy efficiency of 0.24μJ/classification on MNIST. Demonstrations are accomplished on sequential AI tasks such as text-based motion detection and real-time wake-up word speech recognition.**

I. Introduction

Recently, deep learning experienced an explosive artificial intelligence (AI) success that, in many applications, paves the way of a dedicate integrated circuits implementation. Most of the previous works are directed to convolutional neural networks (CNNs) [1]-[3], widely successful for image processing, but few to recurrent neural networks (RNNs) [4].

RNN features its dynamic temporal behavior, also known as *memory*, by a feedback path from neurons' outputs to inputs. Fig. 1 illustrates a typical RNN-based sequential classification machine. Different from CNN, RNN's weights are re-used by unfolding neurons in time domain. The non-linear activation function used in RNN are *sigmoid* and *tanh*, far more complicated than *Relu* used in CNN. On the other hand, RNN's pooling layer and fully connected (FC) layer are similar to CNN.

Modern RNN adopts algorithms of long short term memory (LSTM, [6]) and gated recurrent units (GRU, [7]) to avoid weight explosion or vanishing. Though most RNN's computation, including inference and training, are distributed on power hungry GPUs/CPUs, some cases require low power implementation on RNNs. One example is the wake-up word detection on mobile assistants.

In this paper, we proposed an efficient RNN implementation on a RISC platform to support sequential classification on temporal signals. Besides, gated recurrent neural network computation, the design features on-chip incremental learning with few overhead on inference hardware. The rest of paper is organized as follow. Section II introduces the on-chip incremental learning implementation, while section III demonstrates a deep-learning processor prototype. Section IV emphasizes the accelerator of the processor and section V illustrates the measurement of demonstration. Section VI concludes the paper.

The paper was supported by Shanghai Science and Technology Innovative Program under the Grant of 16JC1420300.

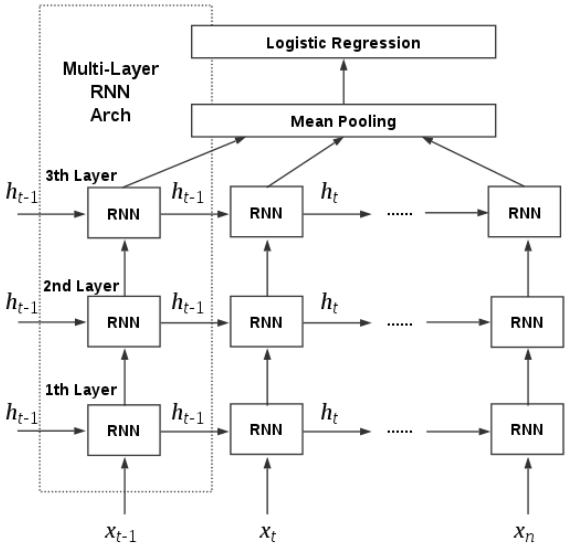
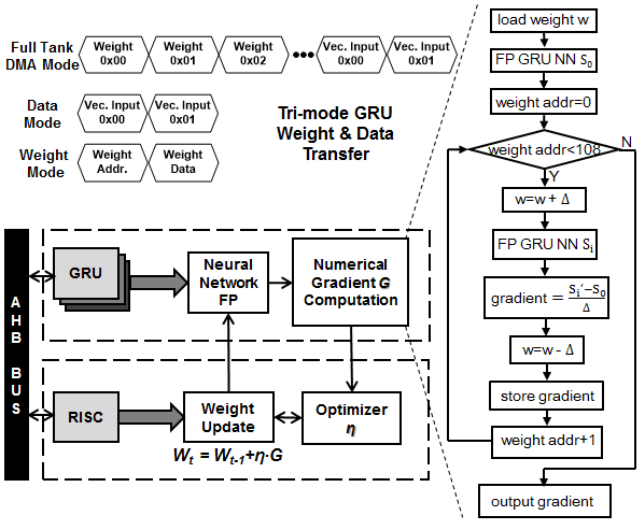


Fig. 1. Speech recognition and wake-up classification adopting recurrent neural network.

II. On-Chip Incremental Learning

Incremental learning is an adaptive learning process that optimizes the network for certain features after the pre-trained classifier. It is a promising feature for future intellectual IoT devices with human-machine interface. A mobile speech assistant, for example, manages to adapt its accuracy with certain users' accents by the incremental learning.

Incremental learning treats the pre-trained neural network as a weak classifier, and re-trains certain weights to enhance the accuracy[5]. According to back propagation through time



(a)

Algorithm 2 GRU Incremental Training Algorithm

Require: Weight, Data
Ensure: Gradient and new weight

```
for s = 0; s < number of samples; s++ do
    *pConfig= Full tank weight memory access mode
    for i =0; i < number of weight / 2 ; i++ do
        *pWeight= Weight[i]; // Weight DMA
    end for
// Original Inference
    *pConfig = Data FIFO mode // Inference
    for i = 0; i < length of sequence; i++ do
        for j =0; j < input dimension / 2; j++ do
            *pData_Wgt = Data[s,i,j];
        end for
    end for
    Result = *pResult;

    for k = 0; k < number of weights; k++ do
        *pConfig = Weight-RAM mode;
        *pData_Wgt = Weight[k] + Δ;
        *pConfig = Data FIFO mode;
        for i = 0; i < length of sequence; i++ do
            for j = 0; j < input dimension / 2; j++ do
                *pData_Wgt = Data[s,i,j];
            end for
        end for
        *pConfig = Write RAM mode;
        *pData_Wgt = Weight[k]; // Resume Weight
// Numerical Gradient
        Grad_Wgt[k]=(*pResult-Result)/Δ;
    end for
// Weight Update
    for k = 0; k < number of weights; k++ do
        Weight[k] = Weight[k]+Grad_Wgt[k]>> 4;
    end for
end for
```

(b)

Fig. 2. On-chip incremental learning (a) calculation flow and (b) Peseudo Code

(BPTT) [6] methods, RNNs' learning updates weights by stochastic gradient descent,

$$\mathbf{w}^{(j+1)} = \mathbf{w}^{j} - \eta \cdot \underbrace{\frac{\partial \text{ Error}}{\partial \mathbf{h}_t}}_{\text{LogReg}} \cdot \underbrace{\frac{\partial \mathbf{h}_t}{\partial \mathbf{w}}}_{\text{Hidden}}, \tag{1}$$

where $\mathbf{w}$ stands for the weight vectors of the hidden layer, $\mathbf{h}_t$ is the hidden output, η is step size defined by various algorithms. Traditional analytic gradient computation requires complicated recursive multiplication and wide dynamic range of digital presentation, infeasible for fixed point systems. However, if it is assumed that learned knowledge is obvious, a numerical gradient computation might be applied to the incremental learning by,

$$\frac{\partial \text{ Error}}{\partial \ w_{i,j}} \approx \sum_{i=1}^{m} \frac{\sum_{k=1}^{n}[P_i^{(k)}(x_i|w_0 + \Delta w) - P_i^{(k)}(x_i|w_0)]\mathbb{L}_i^{(k)}}{m \cdot \Delta w} \tag{2}$$

$$, \text{when } w_{i,j} = w_0$$

where Δw is a tiny incremental step, m is the batch size and $P^{(k)}$ is the *softmax* function result of the k-th class of n class in total. One advantage of the numerical derivative is that it utilize the same feed-forward (inference) computational blocks without extra hardware overhead.

Provided that the inference is implemented by certain on-chip accelerators in a deep learning processor (DLP), the incremental learning could be programmed by a flow illustrated in Fig.2(a). The DLP invokes accelerators to compute gradients numerically. To efficiently accommodate both numerical gradient computation and feed-forward inference, weights access requires both direct-map-access (DMA) and address-index access, i.e. RAM. Incremental learning is realized by downloading pre-trained weights via DMA, and performing gradient calculation via RAM read/write operation.

One example of computing the weight update based on numerical gradient is shown in the algorithm table in Fig. 2(b). All the operations on the accelerator base on the memory access operation with pointers, prefixed by "*p". After initialization and weight loading, the first and original inference brings a reference output layer result, $P_i^{(k)}(x_i|\omega_o)$. Then, each weight is biased by Δ and re-compute the result again, the $P_i^{(k)}(x_i|\omega_o + \Delta\omega)$ in (2). The gradient is obtained by the ratio between result difference and Δ. To implement the division efficiently, Δ are chosen as powers of 2, thus only shift operations involved. A complete iterative would give a vector of gradient. Combining the gradient with a fixed step size (0.0625, or shift by four in the example), a new weight array is updated. Due to the limited bit-width, the gradient tends to be zero. Empirical study shows that random dither injection could improve the accuracy and avoid local minimum. In Fig. 2(b), the batch number of training is set to 1 for simplicity.

III. Processor Architecture Supporting On-Chip Incremental Learning

A prototype deep learning processor supporting the on-chip learning is presented, whose top-level architecture is shown in Fig. 3. It has a 32-bit RISC processor and eight INT-16 precision RNN accelerators, interconnected by an AMBA-AHB bus. The RISC processor deals with most data transfer, whereas the accelerators process most computation. The accelerator would be described in the next section. Interfaces such as GPIO, SPI and JTAG are controlled with an AMBA-APB bus. A compact memory hierarchy is employed, including level-one cache, on-chip high-speed SRAM, and off-chip volume flash memory. The processor is fully programmable in C using a custom developed compiler.

The DLP processor supports on-chip learning using numerical gradient descent. It invokes RNN accelerators to compute

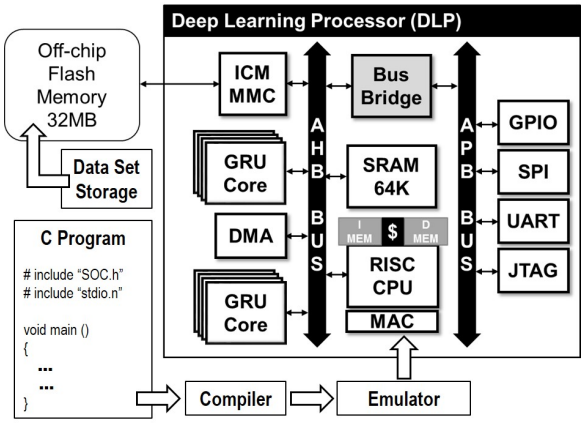


Fig. 3. Deep learning processor's architecture.

gradients numerically, as mentioned above. To efficiently accommodate both numerical gradient computation and feed-forward inference, triple modes of data/weight transfer are employed. A full-tank-DMA mode is to load all the weights initially, a data-FIFO mode is to transfer sequential samples and perform inference, a weight-RAM mode is to update certain weights. The tri-mode design saves the unnecessary data transfer cost. Incremental learning is realized by firstly downloading pre-trained weights via full-tank-DMA, then doing inference with real data via data-FIFO, and finally performing SGD training via weight-RAM.

IV. Gated Recurrent Unit Accelerator

The RNN accelerator is optimized for GRU algorithms, employing a five-stage pipeline architecture, including two matrix-vector product (MVP) stages, two nonlinear activation stages, and one gating-and-storing stage. The accelerator block diagram is illustrated in Fig. 4.

A. RNN Dependency Invoked Hardware Sharing

One significant difference between CNN and modern RNN models is the internal data dependency among RNN's gating coefficients and outputs. Therefore, the critical path of RNN is inevitably longer than CNN. However, this dependency brings a side effect – saving the multiplier-and-accumulate (MAC) number by hardware sharing without throughput penalty.

A straightforward implementation of a GRU RNN cell needs 6 banks of MVPs, if all MACs are unfolded. Data dependency analysis shows that r_t and $\tilde{h_t}$ consist of the critical path, while z_t is a parallel branch. Thus, the hardware of r and z could shared, reducing to 4 MVP banks. The sharing is applied with non-linear activation function (sigmoid) as well. The hardware sharing and dedicated clock gating are controlled are a finite state machine.

B. Precision Extension for On-chip Learning

Requiring higher precision, the on-chip learning method adopts two methods to enhance the training without extending the actual bit-width.

First, sparse weight overflow is compensated by updating bias parameter. Note that RNN's weights are Gaussian distributed and few exceeds the $3-\sigma$ range. Thus the digital presentation is set to the normal range, while exceptions are detected and compensated. These rare weight overflow is fixed by adding the exceeding integer part of the input into the bias (offset) registers when overflow is detected. The fractional residual still invokes the multiplication. The compensation consumes shift/add hardware only.

The second method is to gate extra activation calculation. Typically, activation utilized a 3-segment piece wise linear implementation during inference. However six segments are turned on during on-chip learning to enhance convergence.

V. Measurement Results and Demonstration

The processor is fabricated in a 65nm CMOS 1P9M technology, integrating 4.91M equivalent gates and 64Kb of SRAM within a $2.9 \times 3.5mm^2$ die (GRU core: $900 \times 60\mu m^2$). The chip die photo and measurement results are shown in Fig. 5. It achieves a peak clock rate of 400MHz at 1.2V, consuming 155.8mW of which 9.36mW is dissipated by a GRU core. Scaling to 20MHz and 0.8V, the power consumption is 6.6mW. Table I shows that our GRU-RNN DLP, although not optimized for CNNs, has achieved the competitive performance as those dedicated to CNNs, on MNIST a CNN-oriented benchmark.

Numbers of sequential learning tasks, both inference and numerical training, have been performed. Benchmarks includes MNIST (hand-written digits classification), EEG-based seizure detection, and nature language processing (IMDB and Elec semantic classification). The effectiveness of on-chip incremental learning is demonstrated in Fig.6. A real-time wake-up-word detection demonstration is also completed with

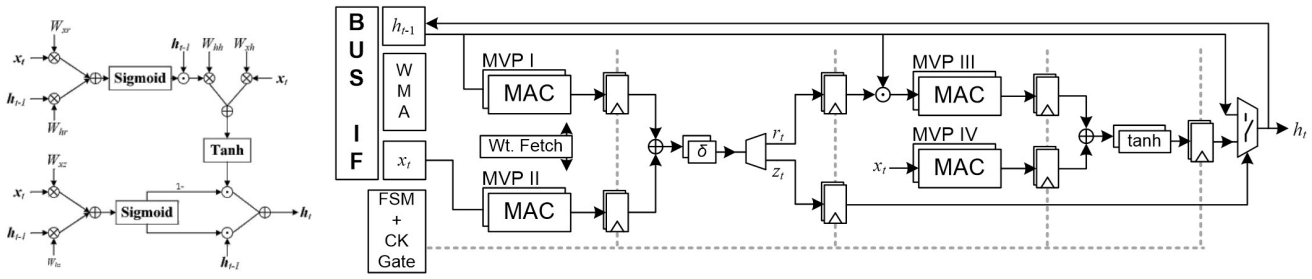


Fig. 4. GRU computational flow and accelerator Implementation.

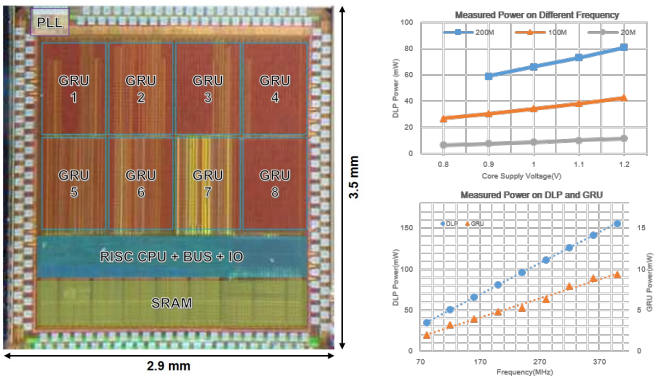


Fig. 5. Deep learning processor die micrograph and power consumption measurement.

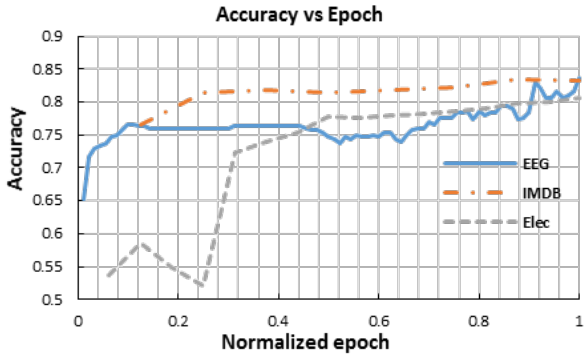


Fig. 6. On-chip incremental learning accuracy convergence.

TABLE I
PERFORMANCE SUMMARY AND COMPARISON

	2016 ISSCC Sim[2]	2016 VLSI Moons[3]	This Work
Technology	65nm	40nm	65nm
ML Algorithm	CNN	CNN	GRU-RNN
Area (mm^2)	4 x 4	1.2 x 2	2.9 x 3.5
On-Chip CPU	No	No	RISC
Clock	125 MHz	204 MHz	400MHz
On-Chip Mem	36 KB	144 KB	64 KB
Power	45mW	76mW	155.8mW
Gate #	3.2M	1.6M	4.91M
Throughput GOPS	64	52.8	311.6
Energy Eff. TOPS/W	1.42	1.6	2.0
MNIST Classf. Throughput	2.78 fps	13.4Kfps	54.237Kfps
Energy per Classificiation	16 μJ	5.7 μJ	0.24 μJ *

* The figure excludes RISC for a fair comparison, the one of the entire system is 4.9μJ.

the proposed chip and a ZYNQ-7000 FPGA-ARM SoC for audio pre-processing. The demo's block diagram is illustrated in 7. The FPGA plays the role of spectrum analysis and segmentation.

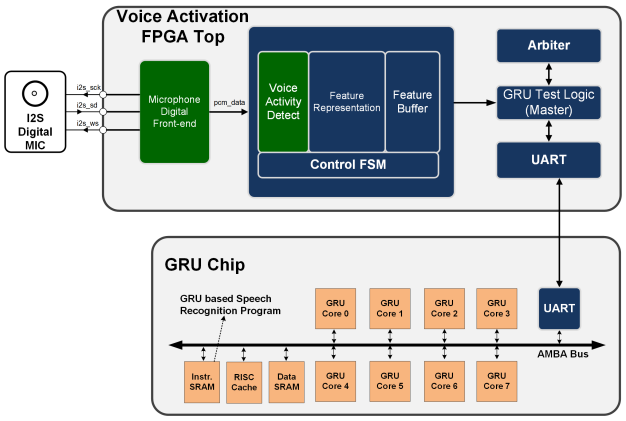


Fig. 7. Wake-up word speed recognition with the proposed chip.

VI. Conclusion

This paper presents a deep learning processor capable of RNN inference and on-chip incremental learning. The processor has a RSIC core and 8 RNN accelerators. On-chip incremental learning is achieved by numerical gradient descent programing with the inference accelerator. Hardware sharing saves the RNN's multipliers' number without penalty of throughput, and weight range extension accommodates the extra precision during on-chip learning. The DLP prototype achieves a clock rate of 400MHz, dissipating 155.8mW and could qualifies as an AI engine for real-time wake-up word recognition under 20MHz clock rate and 0.8V supply.

Acknowledgments

The authors would like to thank Menhang Zhang, Meng Duan, Liang Wang, and Lu Liu for helpful discussions and case testing and C-SKY Inc. for licensing a CK-803 RISC core.

References

[1] Y. Chen, et al., "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks", in *ISSCC Dig. Tech.*, pp 262263, 2016.

[2] J. Sim, el al, "1.42TOPS/W Deep Convolutional Neural Network Recognition Processor for Intelligent IoE Systems", in *ISSCC Dig. Tech.*, pp. 264-266, 2016.

[3] B. Moons, et al, "A 0.3-2.6 TOPS/W Precision-Scalable Processor for Real-Time Large-Scale ConvNets", in *Symp. VLSI.*, pp. 178-179, 2016.

[4] D. Shin, J. Lee, J. Lee and H-J. Yoo, "DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers,* Feb. 2017, pp. 240-241.

[5] R. Polikar, L. Upda, S. S. Upda and V. Honavar, "Learn++: an incremental learning algorithm for supervised neural networks," in *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 31, no. 4, pp. 497-508, Nov 2001.

[6] S. Hochreiter and J. Schmidhuber, "Long Short Term Memory", in *Neural Computation*, No. 8, vol. 9, pp. 1735-1780, 1997.

[7] Cho, et. al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", in *ICML*, 2015.

[8] K.-S. Tai, R. Socher, C. D. Manning, "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks", in *ACL 2015*, arXiv:1503.00075.