A High Energy-efficiency FPGA-Based LSTM Accelerator Architecture Design by Structured Pruning and Normalized Linear Quantization

Yong Zheng[1,2], Haigang Yang[1,2,*], Zhihong Huang[1,2], Tianli Li[1,2], Yiping Jia[1,2]
[1]*Institute of Electronics, Chinese Academy of Sciences*
[2]*University of Chinese Academy of Sciences*
Beijing, China
*yanghg@mail.ie.ac.cn

***Abstract*—LSTM (Long Short-Term Memory) is an artificial recurrent neural network (RNN) architecture and has been successfully applied to the areas where sequences of data need to be dealt with such as Natural Language Processing (NLP), speech recognition, etc. In this work, we explore an avenue to minimization of the LSTM inference part design based on FPGA for high performance and energy-efficiency. First, the model is pruned to create structured sparse features for the hardware-friendly purpose by using permuted block diagonal mask matrices. To further compress the model, we quantize the weights and activations following a normalized linear quantization approach. As a result, computational activities of the network are significantly deducted with a negligible loss on accuracy. Then a hardware architecture design has been devised to fully exploit the benefits of regular sparse structure. Having been implemented on Arria 10 (10AX115U4F45I3SG) FPGA running at 150 MHz, our accelerator demonstrates a peak performance of 2.22 TOPS at a power dissipation of 1.679 Watts. In comparison to the other FPGA-based LSTM accelerator designs previously reported, our approach achieves a 1.17-2.16x speedup in processing.**

***Index Terms*—LSTM, FPGA, Pruning, Quantization**

I. Introduction

LSTM is the most popular variants of RNN, which have been commonly applied in the areas of natural language processing and speech recognition. Despite its superior capabilities in dealing with the dynamic and temporal behavior of data sequences, its architecture usually suffers from difficulties in the hardware design due to the presence of overwhelmingly numerous parameters and hence the incurrence of much high computational complexity. In this work, we explore an avenue to minimization of the LSTM inference part design based on FPGA for high performance and energy-efficiency.

As we known, the power dissipated in accessing the off-chip memory is far more than the on-chip memory and to the worst, those floating-point operations are extremely power hungry. Therefore, by equivalent compression of the network model, the power dissipation shall be effectively reduced for the whole system. This paper presents a hardware-friendly pruning approach that uses the permuted block diagonal mask matrix to generate a regular sparse structure. Differing from the previous approaches, this processing method eliminates completely the huge indexing overhead in tracking non-zero values, which is beneficial to the hardware implementation. Quantifying the weights and activations to fixed-point numbers can further compress the model. However, traditional linear quantization usually suffers from non-convergence problems literally caused by the saturation operations involved. To overcome such drawbacks, this work proposed a normalized linear quantization, mapping the operands within the range [-1, 1] evenly. Therefore, the complex and energy-consuming floating-point operation can then be avoided yet with a negligible loss on accuracy.

In addition to the above optimization at algorithmic level, we also investigate into an energy-efficient approach to design of FPGA based accelerator for LSTM. This features some careful scheduling of the data flow in order to maximize the parallelism in execution, disable the clock for those idle modules, and took calculations of the activation function via the piecewise linear approximation.

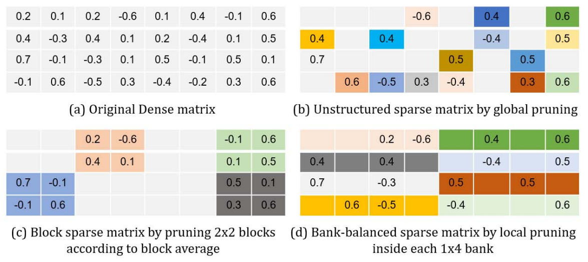
II. Background

A. Long Short-Term Memory

The input of LSTM is a sequence of data, that is: $x = (x_1, x_2, \ldots, x_n)$. For an LSTM unit, the cell states (h, c) and the internal gate $outputs(z)$ are calculated by the following formulas:

$$z_t = tanh(w_x x_t + w_h h_{t-1} + b_*) \tag{1}$$
$$z_t^i = \sigma(w_{ix} x_t + w_{ih} h_{t-1} + b_i) \tag{2}$$
$$z_t^o = \sigma(w_{ox} x_t + w_{oh} h_{t-1} + b_o) \tag{3}$$
$$z_t^f = \sigma(w_{fx} x_t + w_{fh} h_{t-1} + b_f) \tag{4}$$
$$c_t = z_t^f \bigodot c_{t-1} + z_t^i \bigodot z_t \tag{5}$$
$$h_t = z_t^o \bigodot tanh(c_t) \tag{6}$$

The term w indicates a weight matrix (e.g. w_{ix} means a weight matrix associated with the multiplication of input data x dedicated to input gate i). Referring to (1)$\sim$(4), The terms b_*, b_i, b_o, and b_f are categorically defined as the bias vector and the function is taken as a logistic sigmoid. The operator $\bigodot$ denotes the element-wise multiplication.



0.2	0.1	0.2	-0.6	0.1	0.4	-0.1	0.6
0.4	-0.3	0.4	0.1	0.2	-0.4	0.1	0.5
0.7	-0.1	-0.3	0.1	0.5	-0.1	0.5	0.1
-0.1	0.6	-0.5	0.3	-0.4	-0.2	0.3	0.6

(a) Original Dense matrix

			-0.6		0.4		0.6
0.4		0.4			-0.4		0.5
0.7				0.5		0.5	
	0.6	-0.5	0.3	-0.4		0.3	0.6

(b) Unstructured sparse matrix by global pruning

		0.2	-0.6			-0.1	0.6
		0.4	0.1			0.1	0.5
0.7	-0.1					0.5	0.1
-0.1	0.6					0.3	0.6

(c) Block sparse matrix by pruning 2x2 blocks according to block average

		0.2	-0.6		0.4		0.6
0.4		0.4			-0.4		0.5
0.7		-0.3		0.5		0.5	
	0.6	-0.5		-0.4			0.6

(d) Bank-balanced sparse matrix by local pruning inside each 1x4 bank

Fig. 1. The existing pruning methods illustrated by examples.

B. Pruning

Pruning techniques have proven to be an effective method of compressing the models without causing too much performance degradation. Unfortunately, the existing weight pruning approaches all seem have some drawbacks. [1] perform weight pruning by following the smaller-norm-less-important criterion, which believes that weights with smaller norms can be pruned safely due to their less importance. [2] search for the trivial weights heuristically. As illustrated in Figure 1b, these two approaches generally prone to generate irregular sparse structures, which would impose inefficiency on hardware implementation. [3] proposed the coarse-grained pruning methods in terms of the weights in blocks, enabling generation of a regular sparse structure, but at expense of the model accuracy (see Figure 1c). Referred to Figure 1d, [4] divides each weight matrix row into a plurality of those equally sized banks (marked with different colors), and separately prunes each bank using a fine-grained pruning approach to obtain the same pattern of sparsity, thereby generating a regular sparse structure. However, the address corresponding to a non-zero weight is unpredictable and requires additional memory for storage.

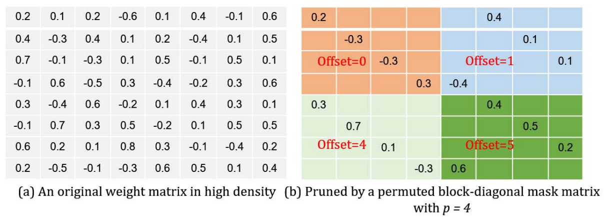
C. Quantization

Quantization is also an effective technique for network compression. [5] quantizes all the weights into the binary form, which has only two values, namely -1 and 1. In order to enhance the digital representation of the network for better accuracy, [6] quantizes weights into the ternary form, still inefficient when dealing with some complicated tasks. [7] proposed an approach of quantifying the weights and the activations to any-bit values. However, due to the saturation function applied to the updated weights and the range confinement of the activations within the range $[-1+\Delta, 1-\Delta]$, considerable deviation errors would incur vanishing gradients and hence leave the network hard to converge. As such, this type of quantification is not performing well on certain tasks.

III. Model Compression

A. Pruned by Permuted Block Diagonal Mask Matrices

[8] proposed a hardware-friendly pruning approach that overcomes the shortcomings of irregularity and large indexing overhead. It uses the permuted block-diagonal mask matrices shown in (7) to generate a regular sparse structure.



0.2	0.1	0.2	-0.6	0.1	0.4	-0.1	0.6
0.4	-0.3	0.4	0.1	0.2	-0.4	0.1	0.5
0.7	-0.1	-0.3	0.1	0.5	-0.1	0.5	0.1
-0.1	0.6	-0.5	0.3	-0.4	-0.2	0.3	0.6
0.3	-0.4	0.6	-0.2	0.1	0.4	0.3	0.1
-0.1	0.7	0.3	0.5	-0.2	0.1	0.5	0.5
0.6	0.2	0.1	0.8	0.3	-0.1	-0.4	0.2
0.2	-0.5	-0.1	-0.3	0.6	0.5	0.1	0.4

(a) An original weight matrix in high density

0.2					0.4		
	-0.3					0.1	
Offset=0		-0.3		Offset=1			0.1
			0.3	-0.4			
0.3					0.4		
	0.7					0.5	
Offset=4		0.1		Offset=5			0.2
			-0.3	0.6			

(b) Pruned by a permuted block-diagonal mask matrix with *p = 4*

Fig. 2. Pruned by permuted block mask diagonal matrix.

Algorithm 1 structured pruning & NLQ

```
Input: initial $\boldsymbol{w}_0, \boldsymbol{h}_0, \boldsymbol{c}_0, \boldsymbol{x} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t\}$
Output: initial $\boldsymbol{w}_N$
for i in {0,...,N-1} do
    $\boldsymbol{w}_i^p \leftarrow Prune\ \boldsymbol{w}_i$
    $\boldsymbol{w}_i^q \leftarrow Quantize\ \boldsymbol{w}_i^q$
    for j in {0,...,t-1} do
        $\{\boldsymbol{x}_j^q, \boldsymbol{h}_j^q, \boldsymbol{c}_j^q\} \leftarrow Quantize\{\boldsymbol{x}_j, \boldsymbol{h}_j, \boldsymbol{c}_j\}$
        $Compute\{\boldsymbol{z}_j, \boldsymbol{z}_j^i, \boldsymbol{z}_j^o, \boldsymbol{z}_j^f\}$
        $Quantize\{\boldsymbol{z}_j, \boldsymbol{z}_j^i, \boldsymbol{z}_j^o, \boldsymbol{z}_j^f\}$
        $Compute\{\boldsymbol{h}_{j+1}, \boldsymbol{c}_{j+1}\}$
    end for
end for
```

$$mask_{ij} = \begin{cases} 1\ if((offset + c)\ mod\ p\ =\ d) \\ 0\ otherwise \end{cases} \tag{7}$$

Where p is a rank of the block diagonal matrix, and $offset = \lfloor i/p \rfloor \times p + \lfloor j/p \rfloor$, with $c = i\ mod\ p$, $d = j\ mod\ p$. Then, the pruned weight w_{ij}^p can be obtained below:

$$w_{ij}^p = w_{ij} \times mask_{i,j} \tag{8}$$

Figure 2 further illustrates the way of the said weight Pruning.

B. Normalized Linear Quantization

As mentioned earlier, the traditional linear quantization(TLQ) approaches usually experience the non-convergence problem caused by the saturation capping. To circumvent, we propose a normalized linear quantization(NLQ) method.

For a weight after being updated, we normalize and map it evenly within the range [-1,1]. That is:

$$w_{ij}^n = \frac{w_{ij}}{max|w_{:,j}|} \tag{9}$$

We then linearly quantize the normalized weight according to the following functions:

$$w_{ij}^q = \frac{round(2^{k-1} \times w_{ij}^n)}{2^{k-1}} \tag{10}$$

$$w_{ij}^s = max(min(1 - \frac{1}{2^{k-1}}, w_{ij}^q), -1 + \frac{1}{2^{k-1}}) \tag{11}$$

In (9)$\sim$(11), w_{ij}^n denotes the normalized weight, w_{ij}^q the quantized weight, and w_{ij}^s the updated weight by saturating operation. Note that k is the bit-width given to a weight.

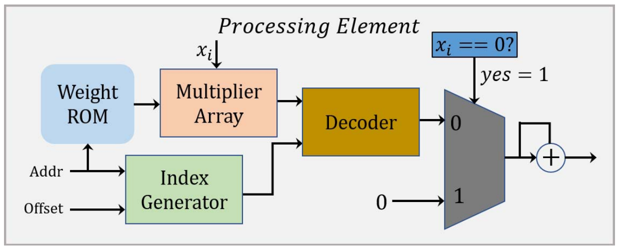


Fig. 3. The architecture of processing element.

Differing from the normalization of weights, the activations are first normalized at the input to a layer and then quantized linearly. Thus, we have:

$$a_i^n = \frac{a_i}{max|\boldsymbol{a}|} \tag{12}$$

Where a_i^n is a normalized activation. Here, i is ranging over for all the input nodes with respect to a layer.

The entire process for the model compressing and training is described in Algorithm 1.

IV. Design of Hardware Architecture

A. Processing Element

Figure 3 shows a processing element architecture having the zero-skipping capability in place. As seen there, it includes a multiplier array that can deliver the multiplication between x_i and non-zero $w_{:,i}$. In order to maximize the data access bandwidth, those non-zero weights from the same column of the matrix are stored exactly into one row of the ROM, which facilitates the read-out just in one clock cycle.

Since the row indexes for non-zero weights at different columns are different, the outputs from the multiplier array need to be decoded before being further accumulated, for the aims to ensure correct calculation regarding the sum of the product for two columns. The decoding operation refers to recovering the zero weight around the non-zero weight according to the row index of non-zero weight. By analyzing the characteristics of the permuted block diagonal mask matrix, we can find that the row index for a non-zero weight can be calculated by the following formula:

$$row_{index} = (col_{index})\ mod\ p - (offset)\ mod\ p \tag{13}$$

B. Overall Architecture

A block diagram of the overall architecture for the proposed accelerator is shown in Figure 4. There are two kinds of matrix-vector multiplications in (1)∼(4), i.e. $w_{*x}x_t$ and $w_{*h}h_{t-1}$. As the dimensions for w_{*x} and w_{*h} are usually not the same, the run times required to complete those two respective multiplications are in fact different. By exploiting this property for energy saving over the waiting period, our architecture dedicates two separate modules h_x_w and $data\_x\_w$ to respectively accomplish $w_{*x}x_t$ and $w_{*h}h_{t-1}$. At the waiting time, the clock to that module in idle can be disabled to cut down the dynamic power dissipation to a certain degree. A data FIFO is then used to synchronize the outputs from those two modules, for the sake of eliminating the memory footprint otherwise taken by the intermediate results.

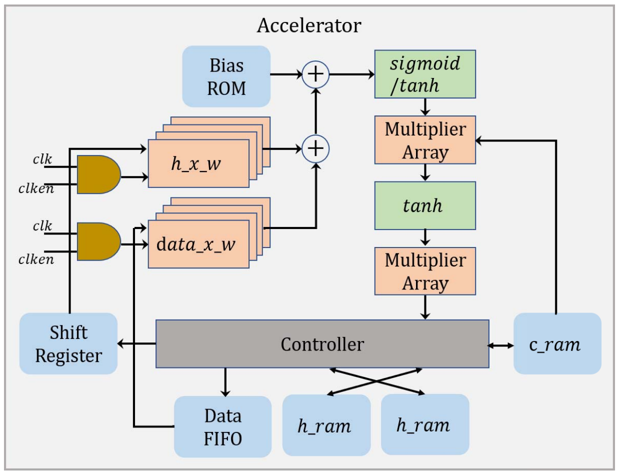


Fig. 4. The overall architecture of the accelerator.

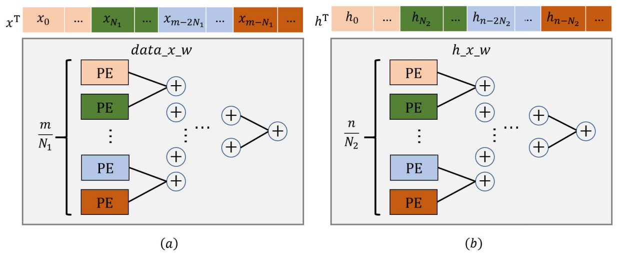


Fig. 5. The internal structure of module h_x_w and $data\_x\_w$.

The designs for realizing modules h_x_w and $data\_x\_w$ are given in Figure 5. There, the numbers of the data processed by each PE are N_1 and N_2 respectively, while $\frac{m}{N_1}$ or $\frac{n}{N_2}$ PEs operate in parallel.

V. Experimental Results

A. Comparison with Traditional Linear Quantization Approach

We first evaluate the model accuracy on PTB dataset based on the structured pruning and the normalized linear quantization (NLQ), by comparing to the traditional linear quantization (TLQ). The rank of the block-diagonal mask matrix is set to 4. As shown in Table I, the network perplexity achieved by our design (Prune&NLQ), specifically when the bit-width of weight and activation (bitsW, bitsA) is set to 4 and 8 respectively, is very close to that acquired by the full-precision model. In a contrast, when the model is compressed by Prune&TLQ, the perplexity becomes much worse off by almost a factor of 3, against the full-precision result.

B. Resource Utilization and Comparison with State-of-the-art LSTM Accelerators

The following design parameters were specified in our implementation of LSTM accelerator: (1) the rank of block-

TABLE I
THE PERPLEXITY OF MODEL THAT USING TWO COMPRESSION APPROACHES WITH DIFFERENT BIT-WIDTH

$(bitsW, bitsA)$	Prune&NLQ	Prune&TLQ
(2, 8)	84.030	142.739
(4, 4)	58.930	113.929
(4, 6)	50.362	115.521
(4, 8)	46.323	116.433
(8, 8)	45.742	120.141
Full precision	45.861	

TABLE II
COMPARE WITH OTHER FPGA-BASED ACCELERATORS

	BBS [4]	FINN-L [10]	C-LSTM [11]
Platform	Arria 10	XCZU7EV	Virtex-7
Bit-width[1]	16-16-16	4-4-4	16-16-16
Frequency(MHZ)	200	266	200
Throughput[2](GOPS)	1520.5	857.6	218.5 [4]
Power[3](W)	19.1	-	22.0
Efficiency(GOPS/W)	15.9	-	15.9
	DeltaRNN [12]	ESE [9]	Ours
Platform	XC7Z100	XCKU060	Arria 10
Bit-width[1]	16-16-16	12-12-12	4-8-8
Frequency (MHZ)	125	200	150
Throughput[2](GOPS)	1198	1575	1850
Power[3](W)	7.3	41.0	-
Efficiency(GOPS/W)	164.1	61.5	-

[1]Consisting with the representation given in FINN-L, the three numbers are the bit-widths of weights, input activations and output activations respectively.

[2]Throughput has been normalized with respect to 125 MHZ.

[3]The listed power is measured (not estimated).

diagonal matrix is 4; (2) the bit-width of weight is 4; (3) the bit width of activation is 8. Also, the clock gating only scheme is applied. Further, the size of hidden state is set as 200. Then, the number of PEs in module h_x_w should be 25, and each PE performs 25 multiplication operations simultaneously.

The design attains a peak performance of 2.22 TOPS at a power dissipation of 1.679 Watts and a resource utilization of 44% in ALMs, 1% in BRAM, 0% in DSP. The power was estimated by Quartus PowerPlay. Further, we compared our accelerator with other five state-of-the-art LSTM accelerators, i.e. BBS [4], FINN-L [10], C-LSTM [11], DeltaRNN [12], and ESE [9], all realized on FPGA. The results have been summarized in Table II. To make a fair comparison, we choose the data of FINN-L with the bit-width setting being 4-4-4 (see note 1 of Table II for explanation). Thanks to our compression approach that overcomes some inefficiency drawbacks existing in the prior-art works, the proposed accelerator design achieves the highest throughput, about 1.17-2.16x speedup when compared to those five FPGA-based accelerators.

VI. CONCLUSION

This work proposed an effective compression approach that uses structured pruning and normalized linear quantization to compress the LSTM model. To great benefits, it solves the problem caused by structural irregularity and large indexing overhead, and also addresses the non-convergence issue arising from limiting saturation operation required by the traditional linear quantization method. Our accelerator design demonstrates a peak performance of 2.22 TOPS at a power dissipation of 1.679 Watts, achieving a high energy efficiency of 1.32TOPS/W. In comparison to the other FPGA-based LSTM accelerators previously reported, our approach demonstrates a 1.17-2.16x speedup in processing performance.

ACKNOWLEDGMENT

This work was supported jointly by the National Natural Science Foundation of China under Grant 61876172, 61704173, and the Major Program of Beijing Science and Technology under Grant Z171100000117019.

REFERENCES

[1] S. Han, H. Mao and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”, Fiber, vol. 56, no. 4 pp. 3–7, 2015.

[2] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio and Jan Kautz, “Importance Estimation for Neural Network Pruning”, In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 11264–11272, 2019.

[3] H. Mao, S. Han, J. Pool, et al., “Exploring the Regularity of Sparse Structure in Convolutional Neural Networks”, arXiv preprint arXiv:1705.08922, 2017.

[4] S. Cao, C. Zhang, Z.Yao, , et al., “Efficient and Effective Sparse LSTM on FPGA with Bank-BalancedSparsity”, in Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA). ACM, 2019.

[5] K. Ando, K. Ueyoshi, Y. Oba, et al., “Dither NN: An Accurate Neural Network with Dithering for Low Bit-Precision Hardware”, 2018 International Conference on Field-Programmable Technology (FPT), pp. 6–13, 2018.

[6] F. Li, B. Zhang and B. Liu. “Ternary weight networks”. arXiv preprint arXiv:1605.04711, 2016.

[7] S. Wu, G. Li, F. Chen and L. Shi, “Training and Inference with Integers in Deep Neural Networks”, In Conference of ICLR, 2018.

[8] C. Deng, Keshab K. Parhi, S. Liao, et al., “PERMDNN: Ef?cient Compressed DNN Architecture with Permuted Diagonal Matrices”. In international symposium on microarchitecture, 2018.

[9] S. Han, J. Kang, H. Mao, et al., “ESE: Ef?cient Speech Recognition Engine with Sparse LSTM on FPGA”, In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA). ACM, pp. 75–84, 2017.

[10] V. Rybalkin, A. Pappalardo, M. M. Ghaffar, et al., “FINN-L: Library Extensions and Design Trade-Off Analysis for Variable Precision LSTM Networks on FPGAs”, 2018 28th International Conference on Field Programmable Logic and Applications (FPL), pp. 89–897, 2018.

[11] S. Wang, Z. Li, C. Ding, et al., “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(FPGA).ACM, pp. 11–20, 2018.

[12] C. Gao, D. Neil, Enea Ceolini, et al., “DeltaRNN: A Power-efficient Recurrent Neural Network Accelerator”. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays(FPGA).ACM, pp. 21–30, 2018.