A COMPACT AND CONFIGURABLE LONG SHORT-TERM MEMORY NEURAL NETWORK HARDWARE ARCHITECTURE

*Kewei Chen, Leilei Huang, Minjiang Li, Xiaoyang Zeng, Yibo Fan**

the State Key Lab of ASIC & System, Fudan University, Shanghai, 200433, China

ABSTRACT

Neural network has been one of the most useful techniques in the area of image analysis and speech recognition in recent years. Long Short-Term Memory (LSTM), a popular type of recurrent neural networks (RNNs), has widely been implemented on CPUs and GPUs. However, software implementation cannot offer large parallelism for the complicated computation of LSTM, and most of the LSTM hardware implementations proposed yet are intensive and non-configurable. In order to accelerate the computation and reduce the resources consumption, in this work, we present a compact and configurable LSTM neural network hardware architecture. To meet the requirements of different networks, we set a wide array of hardware parameters that can be configured to balance area, power and performance. And we adopt the second-order polynomial to approximate the activation functions in LSTM, which balances the computational accuracy and resource utilization. Implemented on XCZU6EG FPGA running at 238 MHz, our work has a performance of 7.64 GOP/s. Compared to the implementation on Intel Xeon E5-2620 CPU at 2.10 GHz, our parallel hardware architecture achieves 90× speed-up for a small network and 25× speed-up for a large one. The total consumption of resources is 77% less than the state-of-the-art works', which implies the compactness of our work.

Index Terms— Neural Networks, Long Short-Term Memory, Configurable Hardware Architecture

1. INTRODUCTION

In the last few years, deep neural networks (DNNs) have made plenty of incredible achievements in many areas. Take pattern recognition as an example, DNN exceeds many excellent algorithms and now represents the new state of the art. However, unlike the human-self neural networks that can store information over time, DNNs can only work on the current input. Recurrent neural networks (RNNs) are designed to address this issue. They store previous outputs as the assistant information for current prediction. This feature makes RNNs naturally suitable for solving problems related to sequences, like speech recognition. Unfortunately, signals flowing backward in time in RNNs can fall into vanishing or exploding [1]. To overcome this shortcoming, a novel type of RNNs, named long short-term memory (LSTM), is proposed.

LSTM introduces an efficient gradient-based algorithm to enforce the internal state remains constant during the recurrent process. Since LSTM is capable of learning long-term dependencies, it has been refined and popularized by many researchers to implemented it on software. However, it is quite difficult for CPUs and GPUs to offer sufficient parallelism for the sequential components of LSTM model. Taking account of computational efficiency, a special hardware architecture is necessary to implement LSTM in embedded systems. Most of the proposed LSTM hardware architectures are non-configurable, which implies they cannot support networks of different sizes and precision. Moreover, their resources consumption is too large for the requirement of the cost-sensitive Internet of Things (IoT). Thus, in this work, we propose an optimized LSTM hardware architecture, which accelerates the computation process with minimal resource consumption. The architecture is both scalable and highly configurable that can meet the requirements of different networks. Besides, we adopt high-precision polynomials to approximate the activation functions instead of look-up tables to further reduce the resource utilization.

The next following sections present the background for LSTM, an overview of related work, implementation details of the proposed hardware architecture, the experimental results and comparisons with other related designs.

2. BACKGROUND

The concept of long short-term memory neural network was originally put forward by Hochreiter and Schmidhuber [1] in 1997. This new type of recurrent neural network was proposed to overcome the long-term memory dependence problem in traditional RNNs. In a standard RNN structure, each neuron passes the output information to a successor. However, as the propagation goes on, the information passing along the chain can fall into a situation either vanishing or exploding. Therefore, the output at the end of sequence, like h_t, cannot build a stable relationship with the input x_0 at the beginning of sequence. In LSTM, a memory cell is introduced to establish long-term memory dependency. A memory cell

The work is supported by Fudan University-CIOMP Joint Fund and Alibaba Joint Project Fund

is composed of the cell state c_t, three gates that protect and control c_t, and an output gate that decides the output h_t.

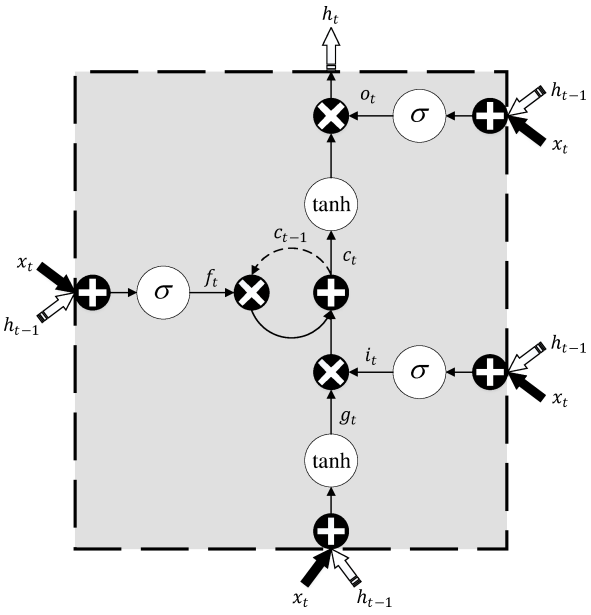

Fig. 1. A normal LSTM architecture.

There are several updated versions of LSTM models nowadays, such as [2] and [3]. And in [4], Greff et al. analyzed the role and utility of all computational components of typical LSTM models. The architecture we adopt here is a normal one without the peephole connections in order to improve the computational speed and reduce the resource consumption. The structure of this model is shown in Fig. 1, which can be characterized by the following equations:

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{b}_f), \quad (1)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{b}_i), \quad (2)$$

$$\mathbf{g}_t = \tanh(\mathbf{W}_{gx}\mathbf{x}_t + \mathbf{W}_{gh}\mathbf{h}_{t-1} + \mathbf{b}_g), \quad (3)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{b}_o), \quad (4)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t, \quad (5)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \quad (6)$$

where σ is the logistic sigmoid function, $\tanh$ is the hyperbolic function, and $\odot$ is element wise multiplication. The symbols $\mathbf{f}$, $\mathbf{i}$, $\mathbf{g}$, and $\mathbf{o}$ represent the output vectors of forget gate, input gate, candidate cell gate, and output gate, respectively. Here $\mathbf{W}_*$ terms denote weight matrices, and $\mathbf{b}_*$ terms denote bias vectors. Assuming that the layer has H neurons and X inputs (i.e. the lengths of $\mathbf{h}_t$, $\mathbf{x}_t$ are H, X, respectively), then the sizes of input weight matrices $\mathbf{W}_{*x}$ and output weight matrices $\mathbf{W}_{*h}$ are $H \times X$, $H \times H$, respectively. The length of $\mathbf{b}_*$ is the same of $\mathbf{h}_t$, i.e., H.

3. RELATED WORK

Many neural networks are conventionally executed on CPU and GPU, which is usually not energy-efficient since they cannot offer enough parallelism for large computations. Thus some application-specific hardware accelerators for DNNs have been proposed to improve the performance.

LSTM, however, is different from standard DNNs in some ways that LSTM faces greater challenges on the matrix-vector operations. Chang et al. described a LSTM hardware implementation with 2 layers and 128 neurons on FPGA in [5]. However, their design is not scalable, and explores coarse-grained computational parallelism. Later in [6] they presented three hardware accelerators for LSTM to improve the performance, while these designs are still non-configurable. Another LSTM hardware architecture in [7] is configurable as ours. They achieved a high level of parallelism at the expense of increasing design complexity, which led to a large consumption of resources. Work in [8] uses floating-point data, which brings large computational workload to the hardware system. Weight compression and pruning are used to achieve better energy efficiency in [9]. However, this architecture is complicated and consumes a lot more resources than ours.

In order to improve the performance and flexibility of hardware architecture, we present a compact and configurable LSTM design. More implementation details are given in the next section.

4. HARDWARE ARCHITECTURE

4.1. System Overview

Fig. 2 shows an overview of our full-pipelined hardware architecture. In general, the whole system can be divided into two parts: gate module and network module. In order to reduce the consumption of on-chip memory, we use the external memory to store the weights. Besides, five pieces of on-chip memory are used to store $\mathbf{b}_*$, $\mathbf{x}_*$, $\mathbf{h}_*$, and $\mathbf{c}$, while we adopt ping-pong buffers to store $\mathbf{h}_*$ of current time and the last time, respectively. To improve the computational efficiency, this design chooses the 16bit fixed-point number format.

The Gate module is responsible for producing the internal gate vectors $\mathbf{f}_t$, $\mathbf{i}_t$, $\mathbf{g}_t$, and $\mathbf{o}_t$. Due to the huge computational cost in this module, especially for large LSTM networks, we adopt several optimization strategies to speed up the module. First, to make full use of the bandwidth, we use several configurable buffers to store data. We define a parameter N as the number of parallelism. Then the length of $\mathbf{x}_*$ and $\mathbf{h}_*$ should be set as $16N$, and the depth should be X/N, H/N, respectively. Besides, we design this module to be compact and full-pipelined. $\mathbf{x}_*$ and $\mathbf{h}_*$ are fetched into the multiplier array with the corresponding weights $\mathbf{W}_{*x}$ and $\mathbf{W}_{*h}$ in turn. The results flow into the adder array to calculate the summation. According to different settings of parallelism, we use the

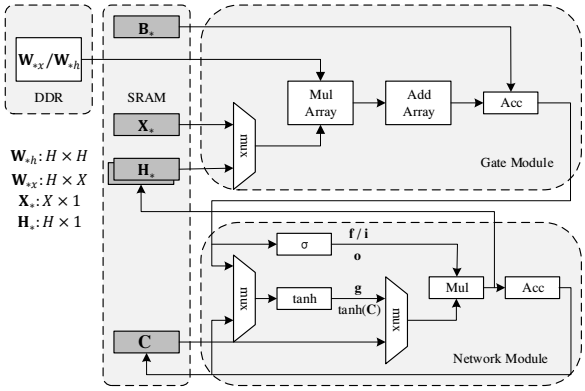

Fig. 2. The proposed full-pipelined hardware architecture.

corresponding number of multipliers and adders to complete the computation in one cycle. Ignoring the latency of data transfer, the estimated numbers of clock cycles needed of this module are

$$((X \times H + H \times H) \times 4 + 4 \times H)/N \tag{7}$$

The Network module is responsible for the activation operation and the computation of output vector $\mathbf{h}_t$. Since the computational workload of this module is far less than the Gate module, we reuse the multiplier and accumulator to reduce the utilization of resources. Furthermore, to reduce the consumption of memory, we adopt polynomial functions to approximate the transcendental activation functions σ and $\tanh$ instead of constructing tow lookup tables. The whole computation process can be described in terms of Algorithm 1. Since we use the second-order polynomial to approximate the activation functions, each activation function evaluator needs 4 clock cycles to complete the computation. Thus the total clock cycles needed are

$$(4H + H) + (4H + H) + H + (4H + H) = 16H, \tag{8}$$

which are far smaller than the costs of Gate module.

Algorithm 1 Network computation

Input: $\mathbf{f}_t$, $\mathbf{i}_t$, $\mathbf{g}_t$, $\mathbf{o}_t$, and $\mathbf{c}_{t-1}$;
Output: $\mathbf{c}$ and $\mathbf{h}_t$;
1: **if** $t = 1$ **then**
2: Initialize $\mathbf{c}_{t-1} = 0$;
3: **end if**
4: Compute $\mathbf{f}_t$ through $\sigma \Longrightarrow \mathbf{f}_t \odot \mathbf{c}_{t-1}$;
5: Compute $\mathbf{i}_t$, $\mathbf{g}_t$ through σ, $\tanh \Longrightarrow \mathbf{i}_t \odot \mathbf{g}_t$;
6: $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$;
7: $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$;
8: **return** $\mathbf{c}$, $\mathbf{h}_t$;

4.2. System Optimization

Due to the high computational complexity of the transcendental activation functions σ and $\tanh$, we need a proper approach to implement them on hardware. Building two lookup tables for σ and $\tanh$ is a simple and efficient solution, but a precise lookup table consumes a lot of resources. Taking into account speed and accuracy, we used the second-order polynomial functions approximations.

To balance computational accuracy and hardware complexity, we divided the activation functions into ten segments to fit. We used totally 65536 (i.e. 2^{16}) uniform points to fit the whole interval $(-32, 32)$. For σ, we set $y = 0$ for $x < -8$ and $y = 1024$ for $x \geq 8$. And between $[-8, 8)$, the entire interval was divided equally into 8 segments, each having a length of 1. For $\tanh$, we set $y = -1024$ for $x < -4$ and $y = 1024$ for $x \geq 4$. Between $[-4, 4)$, the interval was also divided into 8 segments, each having a length of 0.5. The final difference between the integral values of our approximation function and the true activation function is shown in Fig. 3. The average errors are 0.039 and 0.020, which indicates that our approximations are highly accurate.

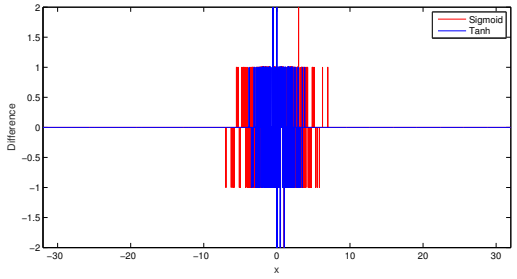

Fig. 3. Difference between the integral values of our approximation function and the true activation function.

To meet the requirements of different LSTM neural networks, we have taken a series of strategies to improve the flexibility and configurability of our hardware architecture.

First, the scale of the network is configurable. By configuring the appropriate parameters to customize a properly sized network, we can reduce resource consumption and make the architecture compact, which is not considered in other works. Second, we used different configurable values of Q for different fixed-point data. For example, we used $Q10.6$ fixed point to represent the polynomial coefficients for the σ and $\tanh$ approximation by default. We designed a separate module responsible for the conversion and calculation of Q. Furthermore, the approximations for the activation functions are also configurable. If the accuracy requirement of the network is high, we can increase the numbers of segments and the precision of the coefficients. Otherwise, we can reduce the complexity to reduce the consumption of resources.

5. EXPERIMENTAL RESULTS

The proposed LSTM hardware architecture was synthesized for a XCZU6EG FPGA core, running at 238 MHz. As we discussed before, the resources utilization of our hardware architecture depends on the degree of parallelism and the scale of the network. The implementation report of different parallelism is illustrated in Fig. 4, which implies that the resource consumption increases with increasing degree of parallelism. Besides, we synthesized the hardware system of different scales. The result is shown in Fig. 5, which implies that the scale mainly affects the utilization of memory.

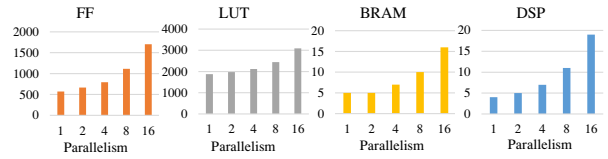

Fig. 4. Resources utilization of proposed LSTM hardware utilization for different degrees of parallelism.

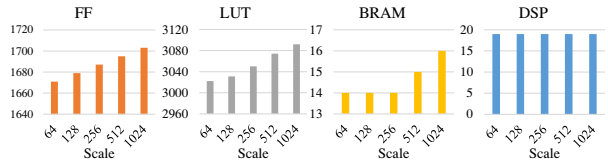

Fig. 5. Resources utilization of proposed LSTM hardware utilization for different scales.

Table 1. Resource utilization comparison with other works

Works	FF	LUT	LUTRAM	BRAM	DSP
[5]	12960	7201	426	16	50
[9]	293920	69939	453068	947	1504
[7]	14215	15423	1535	-	80
[8]	182646	198280	-	1072	1176
[6]	Total 13598 - 61834				
Ours	1703	3092	-	16	19

Compared with other works, shown in the Table 1, our proposed architecture has a great advantage in the resources utilization. The hardware implementation in [5] and [6] is a LTSM network with 2 layers of 128 units. In [9], the result is synthesized on a network with 2 layers of 1024 units. The hardware system in [7] is also scalable, while we use the result based on a size of 16 for comparison. The size of network in [8] is a layer of 123 units. Here we list the resource utilization of our proposed architecture with a parallelism of 16 and a size of 1024. Synthesized result clearly demonstrates that our proposed design is compact.

Table 2. The accuracy of LSTM hardware architecture

Scale	Total	Floating-point		Fixed-point	
		Num.	Accu.	Num.	Accu.
96	6345	6191	97.57%	6190	97.56%
256	4167	3987	95.68%	3986	95.66%
512	10378	10179	98.08%	10179	98.08%
1024	19815	19040	96.09%	19040	96.09%

Table 3. Performance comparison with other works

Works	FPGA	Freq.	Perform.	Pow.
[5]	XC7Z020	142M	264.4MOP/s	1.942W
[9]	XCKU060	200M	282GOP/s	41W
[7]	XC7Z020	140M	4534MOP/s	1.52W
[8]	XC7VX485	150M	7.26GFLOP/s	-
[6]	XC7Z045	142M	454MOP/s	2.3W
Ours	XCZU6EG	238M	7.64GOP/s	0.885W

To examine the performance of our proposed hardware, we implemented a series of LSTM network models in our hardware architecture. The dataset is based on license plate recognition, including different sizes of LSTM networks, from 96 to 1024. We tested the accuracy of the floating-point model in software system and the fixed-point model in hardware system. The result is shown in Table 2. The slight differences between the floating-point and fixed-point model imply the high accuracy of our hardware architecture. Compared to the software implementation, our parallel hardware architecture achieves 90× speed-up for a 96-sized network and 25× speed-up for a 1024-sized network. The performance comparison with other works is shown in the Table 3. The highest frequency and high computational performance indicate the fast speed of our design. Besides, the power of our architecture is the lowest among these implementations.

6. CONCLUSION

This work presents a compact and configurable LSTM neural network hardware architecture. We adopt a second-order approximation for activation functions to balance computational accuracy and hardware complexity. Besides, to meet the requirements of different LSTM neural networks, we have taken a series of strategies to improve the flexibility and configurability of our hardware architecture. Compared with other previous works, our proposed architecture has a great advantage in the resources utilization and performance. Our parallel hardware architecture achieves 90× speed-up for a small network and 25× speed-up for a large one. The total consumption of resources is 77% less than other works', which implies the compactness of our work.

7. REFERENCES

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] F.A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Network (IJCNN)*. IEEE, 2000, vol. III, pp. 189–194.

[3] F.A. Gers, J. Schmidhuber, and Cummins F., "Learning to forget: continual prediction with lstm," in *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, 1999, vol. II, pp. 850–855.

[4] K. Greff, R.K. Srivastava, J. Koutnik, B.R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[5] A.X.M. Chang, B. Martini, and E. Culurciello, "Recurrent neural networks hardware implementation on fpga," *Computer Science*, 2015.

[6] A.X.M. Chang and E. Culurciello, "Hardware accelerators for recurrent neural networks on fpga," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4.

[7] J.C. Ferreira and J. Fonseca, "An fpga implementation of a long short-term memory neural network," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. IEEE, 2017, pp. 1–8.

[8] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "Fpga-based accelerator for long short-term memory recurrent neural networks," in *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 629–634.

[9] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, and Y. Wang, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2017, pp. 75–84.