An SRAM-based implementation of a convolutional neural network

Runchun Wang, Gregory Cohen, Chetan Singh Thakur, Jonathan Tapson, André van Schaik
The MARCS Institute, Western Sydney University, Sydney, NSW, Australia
mark.wang@uws.edu.au

***Abstract*—In the last decade, computational neuroscience and machine learning communities have witnessed the emergence of several algorithms where an input signal is randomly projected to a higher dimensional space via a nonlinear activation function. These methods are increasingly popular for regression or classification tasks, but this kind of neural network has remained difficult to implement efficiently in hardware. This is partly due to the all-to-all connectivity required between the input and hidden layers in these networks. The concept of using receptive fields (RF) for classification tasks stems from biology, in which sensory neurons often respond to a limited spatial range of the input stimulus. Incorporating this methodology into a classification system often yields an increase in performance. This paper presents an SRAM-based implementation of the RF approach to implement this kind of neural network on hardware. Since SRAM has a much smaller footprint compared to logic gates, this implementation is much more efficient in terms of hardware resources. The system was implemented and verified on an FPGA to demonstrate the efficiencies and flexibility of this approach for MNIST digit recognition task.**

I. Background

The Extreme Learning Machine (ELM) is an analytical technique for solving classification and regression tasks in which the output layer weights are solved using a single-step pseudoinverse approach [1]. ELM networks have been used to solve a wide variety of different problems, ranging from non-linear predictive motor control [2] and electricity load prediction [3] to face detection [4]. Additionally, ELM techniques have been extended to deep learning [5] and MapReduce implementations [6]. Iterative methods for calculating the output weights in the ELM networks, such as the Online Pseudoinverse Update Method (OPIUM) [7], remove the problem with learning very large datasets, for which the memory needed to perform the pseudoinverse operation would otherwise become too large. Furthermore, these online methods allow the application of ELMs to non-stationary datasets [7].

Despite their efficiency, ELM networks remain difficult to implement in hardware due to the all-to-all connectivity between the input and hidden layer neurons. This requires significant hardware resources that increase dramatically as the hidden layer size increases. Hardware solutions generally assume fixed input layer and hidden layer sizes as well, greatly reducing the flexibility of these systems and their applicability to different datasets [8], [9].

In order to alleviate these problems, a new network structure based on a receptive field (RF) approach [10][11] is presented. This network uses a novel encoding SRAM-based addressing implementation to the RF approach. The results presented in this paper show that this approach can yield higher accuracies than the ELM networks that use the all-to-all connectivity.

This paper begins with a short summary of the RF approach and the ELM classifier. It then discusses some of the difficulties encountered when implementing these techniques in hardware. The hardware-optimised RF approach is then presented and described; followed by the results section and a discussion and comparison to existing convolutional neural networks.

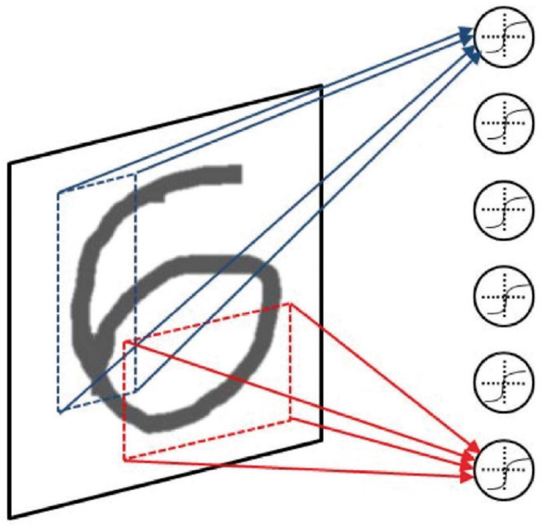

Fig. 1. Illustration of the receptive field approach. Each hidden layer neuron will only receive inputs from only a small random rectangular receptive field in the original image plane. The figure was adapted from [11].

This work has been supported by the Australian Research Council Grant DP140103001.

II. Receptive Field Approach

A. Basic Receptive Field Approach

The concept of using receptive fields ([11]. see Fig. 1) for classification tasks stems from biology, in which sensory neurons often respond to a limited spatial range of the input stimulus. Incorporating this methodology into a classification system often yields an increase in performance [11] and also has the added advantage of resulting in a sparse mapping between layers. When used in an ELM, the RF approach has shown to produce similar accuracy to conventional ELM approaches [11], but with an added advantage of easier implementation in hardware. Furthermore, the locations of the receptive fields can vary with the dimensions of the input data, with only the mapping between hidden layer neurons and their respective receptive fields needing to be changed. This is particularly beneficial when dealing with hardware implementations, which often require a fixed number of neurons in each layer.

The biggest problem with the RF approach is the difficulty in implementing it directly using hardware, especially when the position, shape, and size of the RF need to be determined randomly or via a flexible pattern, requiring random access across all the dimensions of the input data.

B. Improved RF approach

In order to effectively implement the RF approach in hardware, the proposed system makes use of a vectorised version of the input and a set of shift registers. The RF implementation presented here selects regions in the input image through sequential indexing from a shift register. This allows varying receptive fields to be extracted from the input image by simply changing the starting position within the shift register. It is important to note that the receptive fields generated using this approach are not always rectangular, and it is not possible to generate any arbitrary region using this approach. Despite these limitations, the system still performs very well, as demonstrated in the results section.

Fig. 2 demonstrates how the improved RF approach operates on the 28x28 pixels, each of which is an 8-bit grey-scale value, input from the MNIST dataset [12]. Mathematically, the proposed RF approach will first reshape the input image from 28x28x8bit to 784x8bit and then shift the RF across this reshaped image with a step size of 16 pixels. Here the region size is set to 128 pixels.

Fig. 2 shows the input shift register on the left and the resulting receptive field overlaid on the input image on the right at three different steps. When starting from the first pixel, the receptive field encompasses the first three rows of the input image as in Fig. 2a. The selected region will then be used to generate a stimulus to the first neuron of the hidden layer by multiplying with input weights, which are randomly generated for each hidden layer neuron. In the next step, when the starting index is moved, the receptive field shifts and

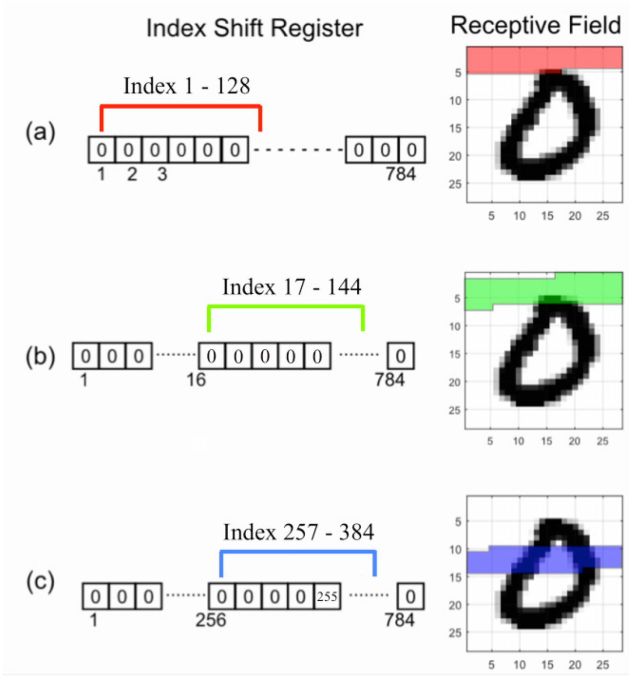

Fig. 2. Improved RF approach.

selects a different region as shown in Fig. 2b. The selected region will then be used to generate a stimulus to the second neuron of the hidden layer by multiplying with its corresponding input weights.

The procedure will be repeated many times to spread the input pixels randomly to a higher dimensional space. Fig. 2c shows the selected region for shifting step 16. After the 49th step, which is the last step of the reshaped image, the index will return to the starting point.

III. Hardware Implementation

A. Topology

In our previous work [13], we have presented an FPGA implementation of a massively-parallel pattern recognition system, which consists of an input layer (the encoder), a hidden layer with 8192 neurons and an output layer. It consists of a physical encoder, a physical neuron, a global counter, and a weight buffer. In that system, the encoder and the hidden layer are both implemented using a time-multiplexing (TM) approach [14]–[19], which leverages the high-speed of digital circuits. The global counter processes the TM encoders and neurons sequentially. The decoding weights of the physical neuron are stored in a weight buffer while the input digit remains static until all the TM neurons finish their processing. In every clock cycle, the TM encoder will generate the hidden neuron input from the input digit, and the TM neuron will pass this signal through a non-linear activation function and multiply it with the decoding weights for the generation of the output value. The output neurons then simply sum across all TM neurons.

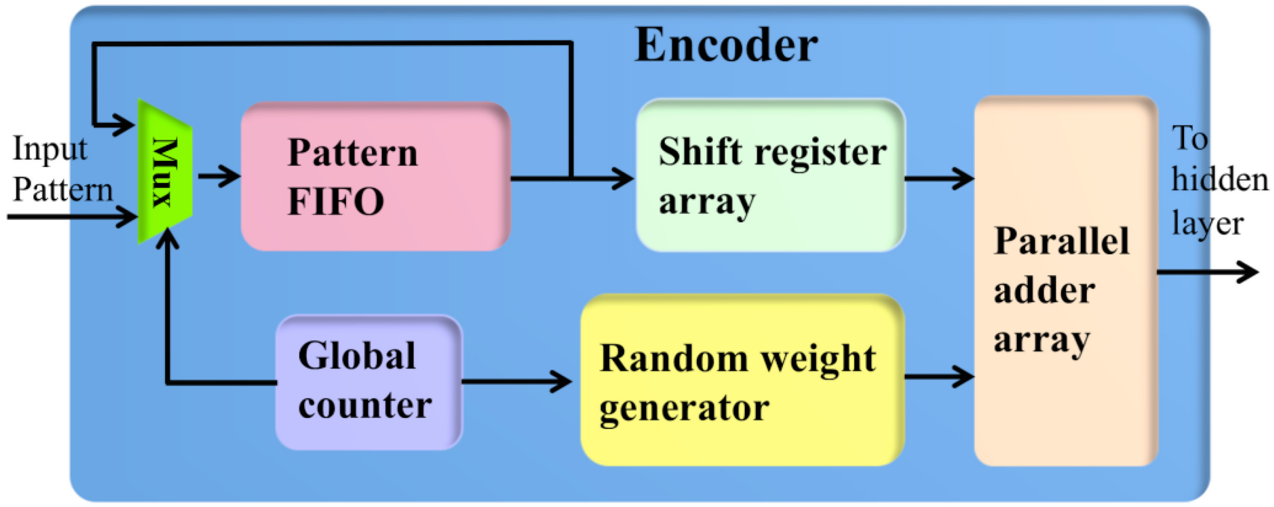

Fig. 3. The structure of the SRAM-based implementation of the improved RF approach.

In this previous work, the physical encoder was implemented using an all-to-all connectivity, which consumed significant amounts of hardware resources. This is indeed one of the most important motivations for developing an encoder using the RF approach. Fig. 3 shows the topology of the encoder, which consists of a pattern FIFO, a shift register array, a random weight generator, a parallel adder and a global counter.

B. *Pattern FIFO and Shift register array*

The pattern FIFO, as shown in Fig. 3, is efficiently implemented on an SRAM with a size of 512x128bits. Hence the maximum size of the input pattern is 8192 with 8-bit grey-scale values. The shift register array consists of eight 128-bit shift registers, which are all sequentially connected. The input of the shift register array is connected to the output of the pattern FIFO and all these eight shift registers are connected to the parallel adder array.

When an input pattern arrives, it is stored in the pattern FIFO, which is assumed to be empty. Next, the shift registers are initialised. This initialisation has eight shifts: the first 128 pixels of the pattern (128x8bits) are moved into the shift register array by reading the pattern FIFO for eight clock cycles and shifting the shift registers eight times. As the pattern FIFO will clear its contents once it has been read out, the output of the pattern FIFO also needs to be written back into the pattern FIFO for keeping the pattern during the whole encoding procedure. Otherwise, the pattern FIFO would be empty after the 49th shift.

After the eight initial shifts, one each clock cycle, the global counter will read the pattern FIFO and shift the register array by one. For instance, if the hidden layer has 8192 neurons, this RF generation operation needs to be performed for 8192 times. Since it is a pipelined design, the output to the parallel adder is still being generated every clock cycle (with a latency of one clock cycle). The pattern FIFO will be flushed right after the completion of the generation of all the regions.

C. *Random weight generator*

The random weight generator will generate a uniformly distributed binary random weight (-1 and 1) for each pixel of the input digit. These weighted pixels will be summed to generate the stimulus for each neuron in the hidden layer. The use of these binary weights saves significant hardware resources in the FPGA, since otherwise we would need 128 multipliers to compute the multiplication between all pixels and their corresponding random weights.

For digital implementations, the most efficient way to generate random numbers is to use linear feedback shift registers (LFSRs). Hence, we use LFSRs to generate the binary random weights and the output of the LFSR will be interpreted following the 2's compliment rule: 0 for +1 and 1 for -1. Input pixel values (8-bit grey-scale) are simply concatenated with the binary weights, resulting in weighted pixel values in 2's compliment notation (9-bit values).

Since an LFSR will go through all possible values in its cycle, its output will not be balanced at each shift. In other words, the number of the 0's and 1's are often not the same, which will affect the performance significantly, because the weighted pixels will be nearly all negative or positive and the generated stimulus for hidden neurons will become extremely large in amplitude. For the generation of balanced binary random weights, instead of a naïve implementation using one 128-bit LFSR, we use twelve 11-bit LFSRs with different seeds, each of which generates an 11-bit random number. For most random seeds, this results in a more evenly distributed number of 0's and 1's.

All these LFSRs will reload their own initial seed on the arrival of an input pattern. After that, they keep generating random numbers until a new input pattern arrives. In this way, we can guarantee that the encoder will generate the exact same set of random weights for all the incoming patterns. This "on the fly" generation scheme also reduces the usage of memory significantly, as there is no requirement for storing the random weights and only the LFSR seeds need to be stored.

TABLE I

Device utilisation and test error

Adaptive Logic Modules (ALMs)	RAMs	DSPs	Test Error
1439/29080	64k/4.5M	0/150	3.78%

D. Parallel adder array

The parallel adder array sums the 128 weighted pixels for generating the input to the hidden layer neurons. A naïve implementation would need a 128-input 9-bit parallel adder and create a large delay (~20 ns). Instead, we use a 3-stage pipeline consisting of sixteen 8-input 9-bit adders, four 4-input 12-bit adders, and one 2-input 15-bit adder respectively. Because of the pipelined design, the input to the hidden layer is still generated every clock cycle, but with a latency of three clock cycles. When the system clock runs at a frequency of 250MHz, this pipelined design is four times quicker than the naïve implementation.

IV. MEASUREMENT RESULTS

To validate the SRAM-based approach, we used it to implement the encoder in the massively-parallel pattern recognition system in our previous work [13], while keeping the remainder of the system remains exactly the same. The pattern recognition system was implemented on an Altera Cyclone V FPGA (on a Terasic Cyclone GX starter kit). The encoder itself uses less than 5% of the hardware resources (Table I).

We tested the pattern recognition with the MNIST dataset. 10 test runs were conducted, each with a different random seed. Since the goal of this exercise was to investigate the performance of the SRAM-based implementation, rather than to find the best possible performance, we used a simplified version of OPIUM, called OPIUM lite, which is a fast online method for calculating an approximation to the pseudoinverse [7]. It is significantly quicker than the full-scale OPIUM, but will find slightly sub-optimal output weights. The average error achieved on MNIST dataset with OPIUM-lite is 3.78% (the standard deviation is 0.096%, Table I). The system that used the all-to-all connectivity, the lowest error achieved with OPIUM lite is 4.52% [7].

V. CONCLUSIONS

We have presented an SRAM-based approach to a convolutional neural network; an RF-ELM. Our approach is highly flexible and uses few hardware resources due to the fact that it uses SRAM instead of logic gates. Furthermore, the results also show that this approach can yield better performance in terms of higher accuracies than the ELM networks that uses the all-to-all connectivity. We can now envision a large-scale fully reconfigurable neuromorphic system, which is capable of performing more complicated pattern recognition tasks.

VI. REFERENCES

[1] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1–3, pp. 489–501, Dec. 2006.

[2] V. M. Janakiraman, X. Nguyen, and D. Assanis, "Nonlinear Model Predictive Control of A Gasoline HCCI Engine Using Extreme Learning Machines," pp. 1–15, 2015.

[3] Ö. F. Ertugrul, "Forecasting electricity load by a novel recurrent extreme learning machines approach," *Int. J. Electr. Power Energy Syst.*, vol. 78, pp. 429–435, Jun. 2016.

[4] A. A. Mohammed, R. Minhas, Q. M. Jonathan Wu, and M. a. Sid-Ahmed, "Human face recognition based on multidimensional PCA and extreme learning machine," *Pattern Recognit.*, vol. 44, no. 10–11, pp. 2588–2597, 2011.

[5] M. D. Tissera and M. D. McDonnell, "Deep extreme learning machines: Supervised autoencoding architecture for classification," *Neurocomputing*, vol. 174, pp. 42–49, 2014.

[6] J. Chen, G. Zheng, and H. Chen, "ELM-MapReduce: MapReduce accelerated extreme learning machine for big spatial data analysis," *IEEE Int. Conf. Control Autom. ICCA*, pp. 400–405, 2013.

[7] A. van Schaik and J. Tapson, "Online and Adaptive Pseudoinverse Solutions for ELM Weights," *Int. Conf. Extrem. Learn. Mach.*, vol. 149, pp. 1–9, 2013.

[8] C. S. Thakur, R. Wang, T. J. Hamilton, J. Tapson, and A. van Schaik, "A Low Power Trainable Neuromorphic Integrated Circuit That Is Tolerant to Device Mismatch," *IEEE Trans. Circuits Syst. I Regul. Pap.*, pp. 1–11, 2016.

[9] C. S. Thakur, T. J. Hamilton, R. Wang, J. Tapson, and A. van Schaik, "A neuromorphic hardware framework based on population coding," in *The 2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–8.

[10] G.-B. Huang, Z. Bai, L. L. C. Kasun, and C. M. Vong, "Local Receptive Fields Based Extreme Learning Machine," *IEEE Comput. Intell. Mag.*, vol. 10, no. 2, pp. 18–29, May 2015.

[11] M. D. McDonnell, M. D. Tissera, T. Vladusich, A. van Schaik, and J. Tapson, "Fast, Simple and Accurate Handwritten Digit Classification by Training Shallow Neural Network Classifiers with the 'Extreme Learning Machine' Algorithm," *PLoS One*, vol. 10, no. 8, p. e0134254, Aug. 2015.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[13] R. Wang, C. S. Thakur, T. J. Hamilton, J. Tapson, and A. van Schaik, "A neuromorphic hardware architecture using the Neural Engineering Framework for pattern recognition," pp. 1–12, Jul. 2015.

[14] R. M. Wang, T. J. Hamilton, J. C. Tapson, and A. van Schaik, "A mixed-signal implementation of a polychronous spiking neural network with delay adaptation.," *Front. Neurosci.*, vol. 8, no. March, p. 51, Jan. 2014.

[15] R. M. Wang, T. J. Hamilton, J. C. Tapson, and A. van Schaik, "A neuromorphic implementation of multiple spike-timing synaptic plasticity rules for large-scale neural networks," *Front. Neurosci.*, vol. 9, no. May, pp. 1–17, 2015.

[16] R. Wang, G. Cohen, K. M. Stiefel, T. J. Hamilton, J. Tapson, and A. van Schaik, "An FPGA Implementation of a Polychronous Spiking Neural Network with Delay Adaptation.," *Front. Neurosci.*, vol. 7, no. February, p. 14, Jan. 2013.

[17] R. Wang, T. J. Hamilton, J. Tapson, and A. van Schaik, "A compact reconfigurable mixed-signal implementation of synaptic plasticity in spiking neurons," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 862–865.

[18] R. Wang, T. J. Hamilton, J. Tapson, and A. van Schaik, "An FPGA design framework for large-scale spiking neural networks," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 457–460.

[19] R. Wang, T. J. Hamilton, J. Tapson, and A. van Schaik, "A compact neural core for digital implementation of the Neural Engineering Framework," in *BIOCAS2014*, 2014.