Hardware-aware Pruning of DNNs using LFSR-Generated Pseudo-Random Indices

Foroozan Karimzadeh*, Ningyuan Cao*, Brian Crafton*, Justin Romberg* and Arijit Raychowdhury*

*School of Electrical and Computer Engineering

Georgia Institute of Technology, Atlanta, Georgia 30332

Email: fkarimzadeh6@gatech.edu, arijit.raychowdhury@ece.gatech.edu

Abstract—Deep neural networks (DNNs) have been emerged as the state-of-the-art algorithms in broad range of applications. To reduce the memory foot-print of DNNs, in particular for embedded applications, sparsification techniques have been proposed. Unfortunately, these techniques come with a large hardware overhead. In this paper, we present a hardware-aware pruning method where the locations of non-zero weights are derived in real-time from a Linear Feedback Shift Registers (LFSRs). Using the proposed method, we demonstrate a total saving of energy and area up to 63.96% and 64.23% for VGG-16 network on down-sampled ImageNet, respectively for iso-compression-rate and iso-accuracy.

I. INTRODUCTION

Artificial Neural networks (ANNs) have shown remarkable performance in a broad range of applications, from computer vision [1], [2] to health-care data analysis [3], [4], [5]. These algorithms have gradually evolved to larger models with deeper layers and increasing number of parameters [6]. While the large DNNs are powerful and provide high accuracy for complex tasks, they cannot be easily deployed on embedded mobile applications due to two major problems [7]. Firstly, because the models are large and often over-parameterized, it must be stored in an external DRAM. The second major issue is that accessing model parameters from an external DRAM is energy consuming [8]. For example, in the 45nm CMOS technology, accessing a 32bit DRAM memory requires 640pJ, which is 3 orders of magnitudes higher than a 32bit floating point add operation (0.9 pJ) [8]. Therefore, it is hard to deploy large DNNs on battery constrained mobile platforms.

Network compression via pruning techniques is one possible solution to fit large DNNs such as VGG-16 (138M parameters, 520MB) in on-chip SRAM [9], [10], [11]. However, it is challenging because sparse matrices add extra levels of irregularity to the weights' addresses [9], [12]. In [13], a pruning method has been proposed which prunes the connections that have smaller weights than a threshold via an iterative process of pruning and retraining. Although this method prunes the network with no loss of accuracy; the method is heuristic and the threshold values need to be carefully selected. Further, resultant matrix lacks structure. In [8], the authors prune the network by learning the important connections using a thresholding mechanism and then applying weight sharing and Huffman coding to compress the networks even further. The problem with this method is that they need to store three vectors: (1) the sparse weights matrix's value, (2) the location or address of the non-zero matrix weights and (3) the pointer vector to keep track of the weights in each column. In addition, weight sharing adds another level of indirection and complexity [14]. In summary, the baseline pruning method is powerful from an algorithm perspective, but its mapping to hardware is inefficient and requires a memory foot-print as high as $2\times$ that of the model size.

In this paper, we present a hardware-aware method to prune dense DNNs which reduces the memory footprint while preserving the original accuracy. We utilize an on-die linear feedback shift register (LFSR) using a known seed, to generate a pseudo random sequence (PRS). Next, the PRS is utilized to regularize and prune the network. In the last step, we retrain the sparse network so that the model can perform better with the pruned model. In addition, during inference we use the LFSR with the same seed to generate the indices in real-time to perform multiplication between the sparse weight matrix and input/activation vector. Consequently, we no longer need to store the sparse weight addresses – thereby reducing the memory foot-print significantly. Finally, the proposed method has been synthesized with 65nm CMOS technology to measure hardware metrics.

II. PROPOSED HARDWARE-AWARE PRUNING

Proposed pruning and baseline methods are illustrated in Figure 1. The proposed pruning method consists four steps. It begins with generating a pseudo-random sequence (PRS) using two LFSRs, one for row indices and another one for column indices. We then use the generated PRS as the indices to sparsify the synapses (i.e. connections) by using standard regularization methods in an iterative approach. The specified synapses are regulated to force them to be zero in the
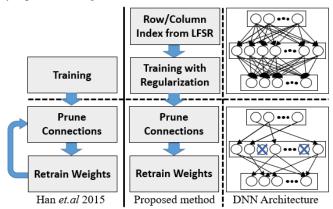

Fig. 1. From left to right, training pipeline for baseline (Han *et.al*, 2015 [13]), proposed pruning method and an example of pruning the synapses and neurons of a fully connected network.
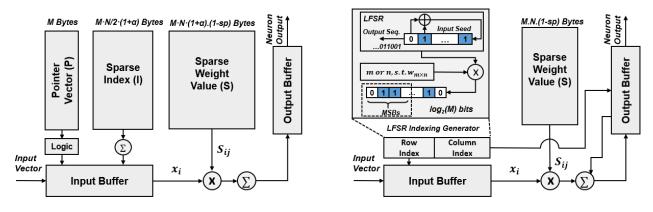
Fig. 2. From left to right, the hardware architecture of the baseline and the proposed pruning method, respectively.

training step and be pruned away in the pruning step. The last step is retraining the sparse model so that the sparse model can provide higher accuracy. Using a PRS for generating the locations of the zero weights in the connectivity matrix provides good accuracy, while making it easier to generate the indices on the fly, instead of being stored in a separate memory sub-bank.

We use Han *et. al*, 2015 [13] as the state-of-the-art baseline pruning techniques for the proposed algorithm. In [13], illustrated in Figure 1, a pruning method was proposed to prune the redundant connections in an iterative process based on a threshold. First, the neural network is trained starting with random initial conditions. Next, the connections less than a threshold are pruned iteratively and finally, the pruned network is fine tuned using several epochs of retraining. Interested readers are pointed to [13] for further details.

A. Linear Feedback Shift Register (LFSR)

LFSR [15] is one of the most commonly used topologies for implementing and generating pseudo random bit sequences [16]. The advantages of using an LFSR to generate the indices are: (1) they can be easily be implemented in hardware, (2) the PRS has key statistical properties that preserves the rank of the generated connectivity matrix [17]. LFSR, consists of an array of flip-flops with an initial state called input seed (s), followed by linear feedback performed by several exclusive-or (*XOR*) gates (c_i). LFSRs can be mathematically described through n^{th} order characteristic polynomials:

$$x^n + c_{n-1}x^{n-1} + ... + c_1x + 1 \quad (1)$$

To obtain the maximum PRS length, the characteristic polynomials have to be primitive [17] where the maximum period without repetition is $2^n - 1$. In the proposed method we utilize two LFSRs, to generate random sequence for the indices of the rows and columns, separately. The row index indicates the address of the element in the input vector while the column address encodes the address of the output vector.

B. Training with PRS based Regularization

A fully connected (*FC*) layer of DNN performs the following function:

$$a = f(Z) \quad \text{where} \quad Z = W^T x + b \quad (2)$$

Where W is a weight matrix, x is an input, b is a vector of bias values, f is a non-linear activation function, typically a Rectified Linear Unit (ReLU) [18]. For simplification, b can be merged with W by appending an additional column to the end of matrix W. Then we can write the above equation as:

$$a = ReLU(\sum_{d=0}^{n-1} W_{cd}x) \quad (3)$$

where c and d are the original weight matrix's indices correspond to rows and columns, respectively. After selecting the synapses using the PRS sequence, we regularize them during the training process. We apply strong regularization on the selected synapses, based on LFSRs indices, in order to force the network to zero-out these synapses. We have studied both L1 and L2 regularization [13] to penalize non-zero weight values. For L2 regularization, a regularizer component is added to the cost (J) , as shown in Eq.4 and weights will be updated based on Eq. 5.

$$J(W^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{c=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} ||W_{ij}^{[l]}||_F^2 \quad (4)$$

$$W^{[l+1]} = \begin{cases} W^{[l]}[1 - \frac{\beta\lambda}{m}], & \text{if } c,d = i,j \\ W^{[l]} - \beta dW^{[l]}, & \text{otherwise} \end{cases} \quad (5)$$

Where i and j are correspond to the indices from LFSR 1 and LFSR 2 for rows and columns, respectively. β is the learning rate. L is the layer's number in the network. λ is the regularization parameter and can be tuned. Larger λ will more penalized the weights values and make them closer to zero.

C. Pruning and Retraining

After heavily regularizing the selected weights, a pruning step is employed to make sure that all the selected weights are exactly zero as regularization makes the connection very close to zero, but not exactly zero. With the LFSR based pruning method, the activation computation of Eq. 6 becomes

$$a = ReLU(\sum_{j}^{n-1} S_{ij}x) \quad (6)$$

where S is a sparse weight matrix. i and j belong to the first and second LFSRs, respectively. The last step of the process is to retrain the pruned network for several epoches to fine tune the remaining synapses so that they better compensate for the removed connections.

Technology Node	TSMC 65nm
Supply Voltage	1V
Temperature	25 °C
Datapath Bitwidth	8b
Index Bitwidth	4b, 8b
Clock Frequency	1GHZ
Memory Bank Size	256B, 512B, 1KB, 4KB

D. DNN compression

To fully understand advantages and limitations of baseline and proposed algorithms, we implement both methods in digital hardware (Figure 2). The two circuit diagrams show how hardware resources/operations differ to solve a sparsely connected fully-connected layer, with N input neuron, M output neuron and Sp as sparsity. For the hardware implementation of our proposed method, the LFSR is used to generate the index for the input to be multiplied to the corresponding weight in sparse matrix weight. LFSR generate the pseudo random sequence with values between 1 to $2^N - 1$. In order to keep the values between number of input neurons, we multiply the generated value to the length of input neurons and select the most significant bits (MSBs). The goal is to avoid redundant clock cycles when the generated number is greater that the number of neurons. After doing multiplication and accumulation operations, the result is stored in the output buffer. The index comes from the LFSR with different input seed which is responsible for generating the sequence for output addresses. The exact number of memory reads from the input and the output buffer depend on the model size as well as the number of multiply and accumulate units available for parallel compute. For the baseline algorithm, the sparse weight matrix is compressed in three vectors including the non-zero values of the weights (S), location of the non-zero weights (I) and a pointer vector to point to the start of each column of the weight matrix (P), that should be saved in the memory. Moreover, each entry bit-width of S and I is designed to be four-bit or eight-bits, and additional memory usage ratio resulted from limited index representation is denoted by α. For instance, if more than 15 zeros appear before a non-zero four-bit entry, a zero is added to vectors S and I.

III. RESULTS

A. Simulation results for the proposed pruning algorithm

The proposed methodology is demonstrated on three pruned networks: LeNet-300-100, LeNet-5 and VGG-16 using MNIST, CIFAR-10 and down-sampled ImageNet [19] datasets. Training is carried out on Nvidia GTX 1080 Ti GPUs. The key parameters for hardware implementation are shown in Table I. Rate of compression along with the top-1 accuracy error before and after pruning for three mentioned models are shown in Table II. We observe that our method does not affect the rank of weight matrices (Table III). As the matter of fact the rank in the proposed approach is close to full rank (as in unpruned models) of the dense matrix. Since the PRS maintain the matrix rank, we infer that the *expressibility* of the weight matrices and accuracy of the network can remain unchanged.

Network	Error	Parameters	Compression Rate
LeNet-300-100 Unpruned	4.2%	267K	
LeNet-300-100 Pruned	4.9%	24K	11×
LeNet-5 Unpruned	1.5%	431K	
LeNet-5 Pruned	1.6%	43K	10×
Modified VGG-16 Unpruned	48.5%	23M	
Modified VGG-16 Pruned	52.1%	3.3M	7×

1) Pruning on Fully Connected Layers: Large DNNs are over-parameterized; this mostly correspond to the large fully connected layers of these networks. For instance, 124M out of 138M of the parameters are related to the 3 fully connected layer in VGG-16. Because of this, we focused on pruning fully connected layers' connection as they consume most of the energy and memory size from hardware perspective.

2) LeNet on MNIST: LeNet-300-100 is a FC network which has two hidden layers of length 300 and 100 neurons each, which achieves 4.9% error rate on MNIST. Accuracy loss versus different sparsity rates is illustrated in Figure3. Three different λ have been tested on the proposed method and the results before and after retraining are illustrated. The results show that moderate and strong regularization, λ equals to 2 and 10, respectively, have better performance before and after retraining. We picked λ equals to 2 to trade-off between pruning and convergence of the loss function while preventing over-fitting. Both L1 and L2 regularization are tested (Figure 3). L1 has better performance before retraining while L2 achieves better performance after retraining.

The second model tested on MNIST is a convolutional network, LeNet-5, that has two convolutional layers followed by two FC layers. LeNet-5 achieves 1.6% error rate on MNIST. The accuracy versus different sparsity of LeNet on MNIST is illustrated in Figure 4, which shows that our method achieves
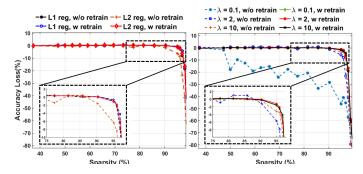

Fig. 3. Sparsity patterns for LeNet-300-100 on MNIST. Three different *lambda*, including 0.1, 2 and 10 have been tested and the accuracy loss is shown before and after retraining (right). Trade-off curves for L1 and L2 regularization (left).

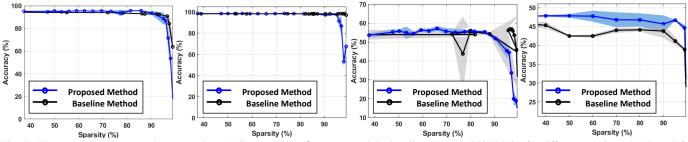| Network		FC layers (unpruned)			Our Method		
Sparsity Hidden Layer		H1	H2	H3	H1	H2	H3
50(%)		120	84	10	118	83	10
90(%)		120	84	10	118	82	10

Fig. 4. The comparison between the proposed method's $mean \pm std$ accuracy and the baseline (Han $et.al$ 2015 [13]) for different sparsity rates. From left to right figures above are for Lenet-300-100 on MNIST, LeNet-5 on MNIST and LeNet-5 on Cifar10 and VGG-16 on down-sampled ImageNet, respectively.

the same accuracy as the baseline for different sparsity rates.

3) LeNet-5 on CIFAR-10: We uses LeNet-5 on CIFAR-10 and the comparison between the $mean \pm std$ accuracy of proposed and the baseline method for 5 trials is shown in Figure 4. The results shows that our method is more reliable and has less std while preserving the original accuracy as it is not based on the thresholding method.

4) VGG-16 on down-sampled Imagenet: We used VGG-16 on ImageNet data [19] with 1000 different classes, but initially down-sampled it to 64×64 [20]. Apart from a single crop with no rotation, we have not used any other pre-processing or augmentation. We then classified down-sampled ImageNet using VGG-16 with some modification to be fit to the spatial size (i.e. 64×64) of down-sampled ImageNet which is due to the fact that the feature size should maintain enough spatial size before each pooling layer. The FC layers size was changed to 2048 and the last pooling layer was eliminated. The results have shown in Figure 4 which shows that the proposed pruning method can preserve the accuracy even in high sparsity rates.

B. 65nm CMOS Hardware Implementation

We have synthesized baseline and proposed methods with 65nm CMOS technology to measure hardware metrics. Implementation parameters are shown in Table I. The pre-layout analysis demonstrates $1.51\times$ to $2.94\times$ reduction in required memory footprint between proposed method and 4-8b indexed baseline pruning technique (Figure 5). Besides memory, the overall system (memory, multiplier, accumulator and input/output buffers) parameters are also measured. The power and area measurements are demonstrated in Table IV and V. We observe a maximum of 63.96% power savings and 68.18% area savings across varying sparsity, indexing bit-
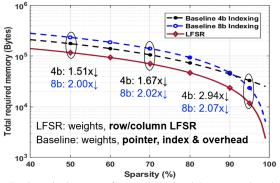

Fig. 5. Total required memory for our method and the proposed method with 4 and 8 bit precision at different levels of sparsity.

TABLE IV
MEASURED POWER (mW) OF THE OVERALL SYSTEM (MEMORY, MULTIPLIER, ACCUMULATOR AND INPUT/OUTPUT BUFFERS) FOR BASELINE AND PROPOSED METHOD.

Network		LeNet-300-100		LeNet-5		modified VGG-16	
Sparsity	Bit-width	4bits	8bits	4bits	8bits	4bits	8bits
Our Method	40%	439.9	439.9	102.9	102.9	37114	37114
	70%	118.2	118.2	31.73	31.73	9294	9294
	95%	46.74	46.74	15.91	15.91	3113	3113
Baseline Method	40%	643.4	857.8	150.4	197.8	55639.1	74184
	70%	176.4	214.4	46.99	55.46	15239.4	18546
	95%	99.74	71.48	30.11	23.82	8637.9	6182
Total Saving (%)	40%	**31.62**	**48.71**	**31.56**	**47.97**	**33.29**	**49.97**
	70%	**32.98**	**44.87**	**32.47**	**42.78**	**39.01**	**49.88**
	95%	**53.13**	**34.61**	**47.15**	**33.21**	**63.96**	**49.64**

TABLE V
MEASURED AREA (mm^2) OF THE OVERALL SYSTEM (MEMORY, MULTIPLIER, ACCUMULATOR AND INPUT/OUTPUT BUFFERS) FOR BASELINE AND PROPOSED METHOD.

Network		LeNet-300-100		LeNet-5		modified VGG-16	
Sparsity	Bit-width	4bits	8bits	4bits	8bits	4bits	8bits
Our Method	40%	1.69	1.69	0.37	0.37	146.12	146.12
	70%	0.42	0.42	0.09	0.09	36.53	36.53
	95%	0.14	0.14	0.03	0.03	12.18	12.18
Baseline Method	40%	2.55	3.39	0.57	0.76	219.20	292.26
	70%	0.71	0.86	0.16	0.20	60.05	73.07
	95%	0.40	0.29	0.10	0.07	34.04	24.36
Total Saving (%)	40%	**33.62**	**50.16**	**34.62**	**50.71**	**33.34**	**50.00**
	70%	**40.15**	**50.63**	**43.15**	**52.74**	**39.16**	**50.01**
	95%	**65.16**	**51.85**	**68.18**	**57.40**	**64.23**	**50.02**

widths and baseline designs. Although significant savings are observed for the proposed method, it should also be noted that the LFSR based column indexing introduces additional output buffer access (1 cycle read and 1 cycle write). This is included in our design and results. We note that additional power is negligible compared to the total power savings.

IV. CONCLUSIONS

In this paper, we propose a new method of indexing a sparse network for inference, to enhance the memory usage and energy efficiency of DNNs. We've utilized an LFSR based indexing by generating two PRS as indices instead of saving them in a separate memory. The generated indices are used to decide which weights to be pruned and which ones to be retained. We show that our method can prune large networks without loss of accuracy. Moreover, maximum of 63.96% power savings and 68.18% area savings across varying sparsity, indexing bit-widths can be achieved.

V. ACKNOWLEDGEMENTS

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[3] J. B. Stephansen, A. N. Olesen, M. Olsen, A. Ambati, E. B. Leary, H. E. Moore, O. Carrillo, L. Lin, F. Han, H. Yan *et al.*, "Neural network analysis of sleep stages enables efficient diagnosis of narcolepsy," *Nature communications*, vol. 9, no. 1, p. 5229, 2018.

[4] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in bioinformatics*, vol. 19, no. 6, pp. 1236–1246, 2017.

[5] R. Boostani, F. Karimzadeh, and M. Nami, "A comparative review on sleep stage classification methods in patients and healthy individuals," *Computer methods and programs in biomedicine*, vol. 140, pp. 77–91, 2017.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[7] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 20.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[9] J. Li, S. Jiang, S. Gong, J. Wu, J. Yan, G. Yan, and X. Li, "Squeezeflow: A sparse cnn accelerator exploiting concise convolution rules," *IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1663–1677, 2019.

[10] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 27–40.

[11] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area v2," in *Advances in neural information processing systems*, 2008, pp. 873–880.

[12] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[13] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[14] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 243–254.

[15] R. Mita, G. Palumbo, S. Pennisi, and M. Poli, "A novel pseudo random bit generator for cryptography applications," in *9th International conference on electronics, circuits and systems*, vol. 2. IEEE, 2002, pp. 489–492.

[16] A. A.-H. Qasem, M. Ibrahim, and A. M. Mohammad, "A double stage implementation for 1-k pseudo rng using lfsr and trivium," *Journal of Computer Science and Control Systems*, vol. 11, no. 1, pp. 13–17, 2018.

[17] T. W. Cusick and P. Stanica, *Cryptographic Boolean functions and applications*. Academic Press, 2017.

[18] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[20] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*, 2016.