A Non-conventional Sum-and-Max based Neural Network layer for Low Power Classification

This is the author's version of the article that has been presented at IEEE ISCAS2022
The editorial version of the paper is available at http://dx.doi.org/10.1109/ISCAS48785.2022.9937576

A Non-conventional Sum-and-Max based Neural Network layer for Low Power Classification

Luciano Prono‡, Mauro Mangia*, Fabio Pareschi‡†, Riccardo Rovatti*† and Gianluca Setti‡†
‡DET, Politecnico di Torino, Italy - Email: {luciano.prono, fabio.pareschi, gianluca.setti}@polito.it
*DEI, †ARCES, University of Bologna, Italy - Email: {mauro.mangia, riccardo.rovatti}@unibo.it

***Abstract*—The increasing need for small and low-power Deep Neural Networks (DNNs) for edge computing applications involves the investigation of new architectures that allow good performance on low-resources/mobile devices. To this aim, many different structures have been proposed in the literature, mainly targeting the reduction in the costs introduced by the Multiply and Accumulate (MAC) primitive. In this work, a DNN layer based on the novel Sum and Max (SAM) paradigm is proposed. It does not require either the use of multiplications or the insertion of complex non-linear operations. Furthermore, it is especially prone to aggressive pruning, thus needing a very low number of parameters to work. The layer is tested on a simple classification task and its cost is compared with a classic DNN layer with equivalent accuracy based on the MAC primitive, in order to assess the reduction of resources that the use of this new structure could introduce.**

I. Introduction

Nowadays, the processing of information directly at the edge of an acquisition system has become increasingly important in the field of Internet of Things (IoT). This means that the study of low power algorithms and architecture for signal processing has become fundamental, as it could unlock on edge/mobile devices many operations that, as of today, can be performed mainly on resource hungry and expensive servers.

In particular, the implementation of low-power and low-resources Deep Neural Networks (DNNs) for edge computing has more than ever become essential, because of the great flexibility and computational capability of these structures [1]–[3]. Examples of DNNs applied to edge computing can be found in the fields of Computer Vision [4], Natural Language Processing [5], Augmented Reality [6] and also Compressed Sensing for biomedical signals [7]–[9].

The search for small and portable DNN structures has been adressed in many different ways. From the technological point of view, many structures based on the novel Phase Change Memory (PCM) have been proposed [10]–[12], working in an analog domain. Instead, with regard to Von-Neumann digital architecturess, many optimization techniques have been proposed, such as parameters quantization and pruning [13], [14]. Also, special architectures have been proposed. Two examples are binary neural networks [15], [16], that use 2's complement operations instead of multiplications, and XNOR-net (and more recently, XOR-net) [17], [18], an extreme quantized version of classic DNNs that uses XNOR gates instead of the expensive multiply operations and employs only 1-bit boolean values. Finally, Logarithmic Neural Networks (LogNet) [19]–[21] approach deserves a mention. It employs data decoded in the logarithmic domain, allowing the substitution of multiplications with the much more simple sum operations. The drawback of these type of networks is identified in the accumulate operation, that in the logarithmic domain becomes a non-linear operation that require the use of Look-Up Tables (LUT) or a maximum operation combined with bit-shift operations. Also, the accumulation of negative values easily becomes extremely imprecise.

In this work, we propose a novel DNN architecture for classification tasks that completely avoid the use of multiply operations, while also avoiding any complex non-linear operation, ideal for low-power implementations on cheap devices.

The paper is structured as follows. Section II contains a complete description of the structure of a low-power DNN layer based on a novel map-reduce paradigm. Section III describes the case study used to assess the performance of the new DNN layer along with the final performance and cost in terms of memory footprint and number of operations. Finally the conclusion is drawn.

II. Non-conventional layer description

Classic DNNs rely inference on a massive use of Multiply and Accumulate (MAC) operations that, despite being typically heavily optimized, make a large use of multiplications, greatly expensive both in terms of speed and power consumption. Aim of this work is to propose a new DNN structure capable of completely avoid multiply operations.

For this reason, a new layer structure is proposed. In order to understand the new structure, we start from the MAC structure, that is a typical map-reduce operation. It is common of dense layers – excluding bias values and the non-linearity introduced by the activation function – and can be represented as

$$\boldsymbol{o} = \boldsymbol{x}\boldsymbol{W} \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the input row vector of the layer, $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ is the matrix of the parameters and $\boldsymbol{o} \in \mathbb{R}^m$ is the output row vector. This can be rewritten more explicitly as

$$\boldsymbol{o}_j = \boldsymbol{x}\boldsymbol{W}_{\cdot,j} = \sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{W}_{i,j} \quad \text{for } j = 1, 2, .., m \tag{2}$$

where $\boldsymbol{o}_j$ is the j-th scalar element of $\boldsymbol{o}$, $\boldsymbol{W}_{\cdot,j}$ is the j-th column of matrix $\boldsymbol{W}$, $\boldsymbol{x}_i$ is the i-th scalar element of vector $\boldsymbol{x}$ and $\boldsymbol{W}_{i,j}$ is the scalar element of matrix $\boldsymbol{W}$ at row i and column j.

This is the author's version of the article that has been presented at IEEE ISCAS2022
The editorial version of the paper is available at http://dx.doi.org/10.1109/ISCAS48785.2022.9937576

Now, we can rethink the whole MAC paradigm as follows. Let us imagine that among the input values, which are weighted by parameters $\boldsymbol{W}$, only a few are prevalent for influencing the output, while the others have an almost negligible contribution. This concept can be roughly implemented by substituting the accumulate operation with the maximum operation, i.e., we search for the maximum scalar among all the weighted inputs. At this point, the parameters $\boldsymbol{W}$ are actually used to increase or decrease the probability that an input is chosen to be the maximum value, i.e., they add value to an input that must be selected and remove value from inputs that must not be chosen (or vice-versa while selecting the minimum, as seen later). In this sense, the parameters $\boldsymbol{W}$ can be either multiplied or added to the inputs, and the best choice is to use additions as they are faster and more energy efficient. So, by replacing multiplications with additions and summation with maximum, (2) is rethought as

$$\boldsymbol{o}_j = \max_i(\boldsymbol{x}_i + \boldsymbol{W}_{i,j}) \quad \text{for } i \leq n,\ \ j = 1, 2, .., m \tag{3}$$

where $\max_i(\cdot)$ selects the maximum scalar value among the elements of its argument vector. We name this new map-reduce paradigm Sum and Max (SAM) and we build our new type of layer over this.

The first thing that we can notice from this new structure is that, contrary to MAC structures and excluding uncommon degenerate cases, only the highest positive values $\boldsymbol{x}_i + \boldsymbol{W}_{i,j}$ are promoted to the output, while anything else is completely ignored. This lead to a potential instability, with output values that are always high and positive with low-variance, eventually killing the information brought by data. A possible solution is the insertion of an inhibitory term in (3), such as

$$\boldsymbol{o}_j = \max_i(\boldsymbol{x}_i + \boldsymbol{W}^+_{i,j}) - \max_i(\boldsymbol{x}_i + \boldsymbol{W}^-_{i,j}) \quad \text{for } i \leq n,\ \ j = 1, 2, .., m \tag{4}$$

where $\boldsymbol{W}^+_{i,j}$ and $\boldsymbol{W}^-_{i,j}$ are the values at row i and column j of the potentiation and inhibitory matrices, respectively. Now, even if (4) can effectively work as a valid DNN layer, we observe that both its terms are influenced by the highest positive values $\boldsymbol{x}_i$, while negatives values are typically ignored. Then, a further modification of (4) results in

$$\boldsymbol{o}_j = \max_i(\boldsymbol{x}_i + \boldsymbol{W}^+_{i,j}) + \min_i(\boldsymbol{x}_i + \boldsymbol{W}^-_{i,j}) \quad \text{for } i \leq n,\ \ j = 1, 2, .., m \tag{5}$$

where $\min_i(\cdot)$ selects the minimum scalar value among the elements of its argument vector. Now, the second term of (5) promotes negative and high (in module) values of $\boldsymbol{x}_i + \boldsymbol{W}^-_{i,j}$. Because of this, this term works actually as an inhibitory term.

Remarkably, the activation function is not required in SAM-based layers as the $\max_i(\cdot)$ and $\min_i(\cdot)$ functions already introduce a non-linearity.

In Fig. 1 we highlight the structures of the DNN neurons based on MAC and on SAM structures, which correspond to the operations for the generation of a single scalar value $\boldsymbol{o}_j$. Also, it is shown how the neurons are disposed inside a dense layer to generate the whole output $\boldsymbol{o}$, i.e., the graphical representation of (2) and (5).

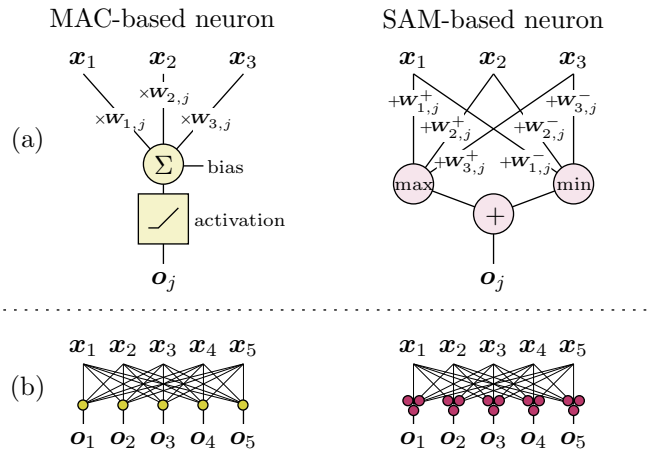

Fig. 1. Comparison of a MAC-based neuron and a SAM-based neuron (a) and their disposition inside a dense layer (b). Note that in the SAM-based neuron parameters are added and not multiplied, while bias and activation function are not needed, as opposed to MAC-based neurons.

The resulting layer uses additions instead of multiplications for the application of the parameters to the inputs. Also, it needs only subtractions to perform the comparisons between values needed by $\max_i(\cdot)$ and $\min_i(\cdot)$ function.

As we want to train this structure with supervised Stochastic Gradient Descent (SGD) based training, it also needs a gradient. The gradient for $\max_i(\cdot)$ and $\min_i(\cdot)$ operations is quite simple: it is set to 1 (linear) for input x_i that is selected by $\max_i(\cdot)$ ($\min_i(\cdot)$) function, 0 for all the others.

III. Performance and cost

We want to assess the performance of a layer built with (5) by using a simple working toy-case. To do so, we prepare a simple classification task with 10 classes and 2 different DNN structures, one based on classic MAC operation, the other one based on the new SAM paradigm. The two DNNs are built and trained so that they have the same final performance in terms of accuracy. After training, pruning – that is the removal of the least influent parameters from the layer – is used to compress the two structures and finally the cost in terms of memory footprint and computational operations is assessed.

A. Classification task

In order to study the performance of SAM-based layers in DNNs, a simple classification problem is prepared as a case study. Starting from a covariance matrix $\boldsymbol{X}$ of size $n \times n$, a signal $\boldsymbol{x}$ with n samples can be generated with a zero-mean multivariate gaussian random process. Matrix $\boldsymbol{X}$ is generated starting from a value $r \in (-1, 1)$ as

$$\boldsymbol{X}_{i,j} = r^{|i-j|} \tag{6}$$

where $\boldsymbol{X}_{i,j}$ is the element corresponding to row i and column j of matrix $\boldsymbol{X}$. Different values of r corresponds to a different

This is the author's version of the article that has been presented at IEEE ISCAS2022
The editorial version of the paper is available at http://dx.doi.org/10.1109/ISCAS48785.2022.9937576

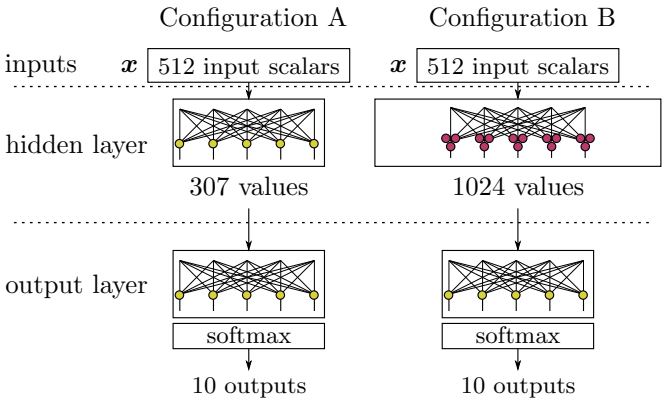

Fig. 2. Structures of configuration A, that has a MAC-based hidden layer with 307 outputs, and configuration B, that has a SAM-based hidden layer with 1024 outputs.

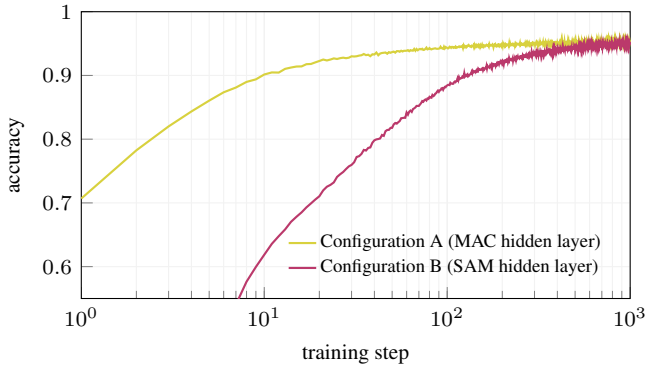

Fig. 3. Accuracy trend during training of configuration A (containing MAC-based hidden layer) and B (containing SAM-based hidden layer). The dataset is generated randomly on-the-fly, so no validation curve is required.

distribution of the resulting signal $\boldsymbol{x}$, such that if $r < 0$ the frequency band of the resulting signal is high-pass, while if $r > 0$ it is low-pass, and the higher is $|r|$, the more localized is the signal, meaning the frequency band is less wide [22]. This means that $r = 0$ corresponds to flat band white noise, while values near -1 and 1 correspond to high frequency and low frequency and thin band signals, respectively.

Given a signal $\boldsymbol{x}$ generated from an unknown covariance matrix $\boldsymbol{X}$, the classification task consists in retrieving the correct value r associated with it. In particular, we generate the data set starting from 10 different values of r equally distributed in the interval $[-0.9, 0.9]$, so the DNN must choose the correct value of r among 10 different possibilities.

B. DNN configuration

Our test structure consists in a DNN with $n = 512$ inputs, one hidden layer and one output dense layer with 10 outputs and a softmax activation function. The hidden layer can be either a MAC-based dense layer (configuration A) or a SAM-based layer (configuration B). The number of outputs of the hidden layer is selected in order to have a similar classification performance for configurations A and B, so that it is possible to make a fair comparison between the two structures. To do so, different DNN with different sizes for the hidden layer are trained and compared and a good compromise between hidden layer size and performance of the two configurations is found. With this, configuration A is set with $0.6n \simeq 307$ outputs (that is, 157,286 weights plus 307 bias values) and configuration B is set with $2n = 1024$ outputs (that is, 1,048,576 parameters). A summary of the structure of the two configurations can be seen in Fig. 2. Even tough configuration B initially has a far greater number of parameters, their number can be largely reduced at post-training optimization time.

C. Training and performance

During training, pairs of signals $\boldsymbol{x}$ and labels associated to one of the 10 values of r are fed to the DNN. Training is performed for both configurations A and B with Adam optimizer [23] through 1000 epochs. As described in Section III-A, the dataset is generated randomly on-the-fly using 10 different covariance matrices generated from as many values r, so that validation data is not required during training. Configuration A uses a batch size of 64 with a learning rate of 0.001 and performs 1000 steps per epoch. Configuration B uses a batch size of 256 with a learning rate of 0.008 and performs 250 steps per epoch. So, both configurations feed 64,000 randomly generated signal-label pairs to the DNN each epoch, with a total of $64 \cdot 10^6$ different instances fed thorough the whole training procedure.

The final accuracy – defined as the number of correct classifications over the total number of inferred input vectors – is the same for both configurations A and B, as they have been structured to have the same performance, and about equal to 0.95. This result is enough to show that (5) may be effectively used to substitute a dense MAC-based layer. Nonetheless, configuration B needs more epochs to converge to the final result, even if the training hyper-parameters (such as number of epochs and learning rate) have been selected to maximize the training speed. This is due to the way the gradients of $\max_i(\cdot)$ and $\min_i(\cdot)$ are built, which allows the gradient of each output to propagate to only one input for each signal fed to the SAM structure. Thus, for each signal only one parameter per output is actually updated, slowing the whole process. Training accuracy trend – that coincide with validation accuracy in this case, as signals are synthetically generated – can be seen in Fig. 3.

D. Optimization by pruning and cost

After training, the 2 configurations are tested against a fixed dataset of 10^5 random generated signals. In particular, the confusion matrix of configuration B is shown in Fig. 4, highlighting what are the values of r that the DNN predicts against the actual target values.

We also apply pruning to the hidden layer [24], i.e., the removal of interconnections to the DNN layer. Removing an interconnection fundamentally means completely ignoring a parameter during the inference of the DNN. While for MAC-based layers this is mathematically obtained by setting the

This is the author's version of the article that has been presented at IEEE ISCAS2022
The editorial version of the paper is available at http://dx.doi.org/10.1109/ISCAS48785.2022.9937576

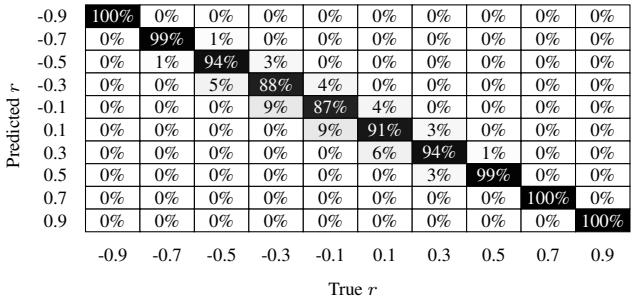

Fig. 4. Confusion matrix of configuration B (with SAM-based hidden layer) with a test set of 10^5 random signals. As the data set is equally distributed through the 10 classes, values are normalized over 1000 instances for each class.

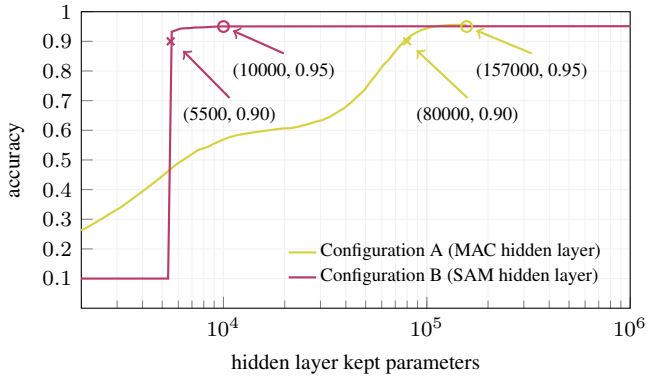

Fig. 5. Accuracy vs number of parameters that are kept after pruning in the hidden layer, both for configuration A (containing the MAC-based hidden layer) and B (containing SAM-based hidden layer). The numbers of parameters with accuracy 0.95 and accuracy 0.90 for the two configurations are highlighted.

parameter to 0, for SAM-based layers it is necessary to set the parameter to $-\infty$ or ∞ for $\max_i(\cdot)$ and $\min_i(\cdot)$, respectively. Ignoring a parameter means reducing the memory footprint of the DNN and the number of operations that it is necessary to perform.

For the sake of simplicity, a naive approach is used for configuration A: all the parameters whose absolute value is below a given threshold are set to zero. Many thresholds are tested and the resulting accuracy is tested for each of them. Of course, the greater the threshold, the lower the accuracy, as less parameters are employed.

Configuration B does not follow the same procedure. During SAM operations, $\max_i(\cdot)$ (or $\min_i(\cdot)$) selects only one value $x_i+W^+_{i,j}$ (or $x_i+W^-_{i,j}$) for each $j = 1, 2, .., m$. The parameter $W^+_{i,j}$ (or $W^-_{i,j}$) associated with the value selected is then considered *activated* and is the only value for a given whole input vector $\boldsymbol{x}$ that actually influences output $\boldsymbol{o}_j$. So, we count how many times each parameter is activated and we call this quantity *activation rate*. An activation rate threshold is set and all the connections that are below this value are removed, i.e., the parameters that are less activated are ignored. Different activation rate thresholds are set and for each of them the classification performance is evaluated.

Actually, the analyses show that for most of the parameters in configuration B the activation rate is completely null, so this structure is naturally prone to aggressive pruning. Because of this, even if during training configuration B requires more parameters compared to configuration A, during inference it actually uses a far lower number.

In Fig. 5 we see the classification performance of configurations A and B versus the number of parameters that are kept in the hidden layer. Without pruning, both configurations have an accuracy of 0.95. The MAC-based hidden layer uses about 157000 parameters, while the SAM-based layer employs only about 10000, after removing the useless (null activation) parameters. If, instead, we apply pruning and we lose 0.05 in accuracy performance, with configuration A we have about 80000 parameters, while with configuration B we have only about 5500 parameters.

Even if we are focusing on the hidden layer, it is important to notice that the number of parameters in the output layer for configuration A is lower than configuration B, having 3080 parameters against 10250, respectively. Nonetheless, configuration B remains a far cheaper implementation.

TABLE I
MEMORY FOOTPRINT AND COMPUTATIONAL COST OF THE HIDDEN LAYER OF CONFIGURATIONS A AND B

Configuration	Parameters	Multiplications	Additions
A (MAC-based)	$80.0 \cdot 10^3$	$80 \cdot 10^3$	$80 \cdot 10^3$
B (SAM-based)	$5.5 \cdot 10^3$	0	$11 \cdot 10^3$

Finally, we evaluate the cost in terms of number of additions and multiplications of the hidden layer of the two configurations. We define N_A as the number of parameters kept in the hidden layer of configuration A, N_B as the same for configuration B. In MAC-based configuration A layer, we perform a number of multiplications $\mathrm{mul}_\mathrm{A} = N_\mathrm{A}$ and a number of additions $\mathrm{add}_\mathrm{A} = N_\mathrm{A}$ for each inference step. In SAM-based configuration B hidden layer instead we need $\mathrm{mul}_\mathrm{B} = 0$ and $\mathrm{add}_B = 2N_\mathrm{B}$, as $\max_i(\cdot)$ and $\min_i(\cdot)$ operations are performed through successive comparisons (i.e., subtractions with sign check of the result). The memory footprint and the cost in terms of multiplications and additions is shown in Tab. I, where the big advantage in terms of hardware resources of the SAM-based layer over the MAC-based layer is highlighted.

IV. Conclusion

We have presented a multiplier-free DNN layer based on a SAM paradigm, opposed to the classic MAC structure. The performance of this layer has been tested on a simple classification task, and its functioning has been proven, even though a further research on the training process could improve the results. Finally, this new layer has been shown to be prone to aggressive pruning, thus actually needing only a few parameters and because of this being very cheap in terms of memory footprint and number of operations.

REFERENCES

[1] M. Merenda, C. Porcaro, and D. Iero, "Edge Machine Learning for AI-Enabled IoT Devices: A Review," *Sensors*, vol. 20, no. 9, p. 2533, Jan. 2020. doi:10.3390/s20092533

[2] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019. doi:10.1109/JPROC.2019.2921977

[3] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, Jan. 2020. doi:10.1109/TWC.2019.2946140

[4] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The Design and Implementation of a Wireless Video Surveillance System," in *21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*, Sep. 2015, pp. 426–438. doi:10.1145/2789168.2790123

[5] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, "FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network," in *32nd International Conference on Neural Information Processing Systems (NIPS'18)*, Dec. 2018, pp. 9031–9042.

[6] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *12th annual international conference on Mobile systems, applications, and services (MobiSys '14)*, Jun. 2014, pp. 68–81. doi:10.1145/2594368.2594383

[7] A. Mousavi, A. B. Patel, and R. G. Baraniuk, "A deep learning approach to structured signal recovery," in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sep. 2015, pp. 1336–1343. doi:10.1109/ALLERTON.2015.7447163

[8] M. Mangia, L. Prono, A. Marchioni, F. Pareschi, R. Rovatti, and G. Setti, "Deep Neural Oracles for Short-window Optimized Compressed Sensing of Biosignals," *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 3, pp. 545–557, Jun. 2020. doi:10.1109/TBCAS.2020.2982824

[9] L. Prono, M. Mangia, A. Marchioni, F. Pareschi, R. Rovatti, and G. Setti, "Deep Neural Oracle With Support Identification in the Compressed Domain," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 458–468, Dec. 2020. doi:10.1109/JETCAS.2020.3039731

[10] S. Ambrogio *et al.*, "Accelerating Deep Neural Networks with Analog Memory Devices," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Aug. 2020, pp. 149–152. doi:10.1109/AICAS48895.2020.9073978

[11] A. Chen *et al.*, "Enabling High-Performance DNN Inference Accelerators Using Non-Volatile Analog Memory (Invited)," in *2020 4th IEEE Electron Devices Technology Manufacturing Conference (EDTM)*, Apr. 2020, pp. 1–4. doi:10.1109/EDTM47692.2020.9117896

[12] A. Mukherjee, K. Saurav, P. Nair, S. Shekhar, and M. Lis, "A Case for Emerging Memories in DNN Accelerators," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, Feb. 2021, pp. 938–941. doi:10.23919/DATE51398.2021.9474252

[13] R. Ding, Z. Liu, R. D. S. Blanton, and D. Marculescu, "Quantized deep neural networks for energy efficient hardware-based inference," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2018, pp. 1–8. doi:10.1109/ASPDAC.2018.8297274

[14] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv:1510.00149 [cs]*, Feb. 2016.

[15] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with Binary Weights During Propagations," in *28th International Conference on Neural Information Processing Systems (NIPS'15)*. MIT Press, Dec. 2015, pp. 3123–3131.

[16] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv:1602.02830 [cs]*, Mar. 2016.

[17] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *14th European Conference on Computer Vision (ECCV 2016)*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Oct. 2016, pp. 525–542. doi:10.1007/978-3-319-46493-0_32

[18] S. Zhu, L. H. K. Duong, and W. Liu, "XOR-Net: An Efficient Computation Pipeline for Binary Neural Network Inference on Edge Devices," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2020, pp. 124–131. doi:10.1109/ICPADS51040.2020.00026

[19] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional Neural Networks using Logarithmic Data Representation," *arXiv:1603.01025 [cs]*, Mar. 2016.

[20] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "LogNet: Energy-efficient neural networks using logarithmic computation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2017, pp. 5900–5904. doi:10.1109/ICASSP.2017.7953288

[21] J. Cai, M. Takemoto, and H. Nakajo, "A Deep Look into Logarithmic Quantization of Model Parameters in Neural Networks," in *10th International Conference on Advances in Information Technology (IAIT 2018)*, Dec. 2018, pp. 1–8. doi:10.1145/3291280.3291800

[22] M. Mangia, F. Pareschi, V. Cambareri, R. Rovatti, and G. Setti, *Adapted compressed sensing for effective hardware implementations: A design flow for signal-level optimization of compressed sensing stages*. Springer International Publishing, 2018. ISBN 978-3-319-61372-7

[23] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017.

[24] 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*, vol. 28. Curran Associates, Inc., 2015.