Reduction of Neural Network Circuits by Constant and Nearly Constant Signal Propagation

Augusto Andre Souza Berndt
PGMICRO, UFRGS
Porto Alegre, Rio Grande do Sul, Brazil
augusto.berndt@inf.ufrgs.br

Alan Mishchenko
UC Berkeley
Berkeley, California, USA
alanmi@berkeley.edu

Paulo Francisco Butzen
PPGComp, FURG
Rio Grande, Rio Grande do Sul, Brazil
paulobutzen@furg.br

Andre Inacio Reis
PGMICRO, UFRGS
Porto Alegre, Rio Grande do Sul, Brazil
andreis@inf.ufrgs.br

ABSTRACT

This work focuses on optimizing circuits representing neural networks (NNs) in the form of and-inverter graphs (AIGs). The optimization is done by analyzing the training set of the neural network to find constant bit values at the primary inputs. The constant values are then propagated through the AIG, which results in removing unnecessary nodes. Furthermore, a trade-off between neural network accuracy and its reduction due to constant propagation is investigated by replacing with constants those inputs that are likely to be zero or one. The experimental results show a significant reduction in circuit size with negligible loss in accuracy.

CCS CONCEPTS

• **Hardware** → **Electronic design automation**; *Logic synthesis*; *Circuit optimization*; • **Computer systems organization** → Neural networks.

KEYWORDS

And-Inverter Graph, Neural Networks, Logic Synthesis

ACM Reference Format:
Augusto Andre Souza Berndt, Alan Mishchenko, Paulo Francisco Butzen, and Andre Inacio Reis. 2019. Reduction of Neural Network Circuits by Constant and Nearly Constant Signal Propagation. In *32nd Symposium on Integrated Circuits and Systems Design (SBCCI '19), August 26–30, 2019, Sao Paulo, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3338852.3339874

1 INTRODUCTION

Artificial Intelligence (AI) is the capability that machines have to understand and solve problems on the real world. Its objective is to perceive its environment and react to maximize its chance of successfully achieving its goal. A sub-field of AI is artificial neural networks, which are computer systems inspired by biological neural networks. This technology is capable of receiving information, processing this information and giving a response as output. It can be used in different fields such as prediction of events that can be quantified, regression analysis, statistical classification, pattern recognition, prediction and financial analysis, medical diagnosis, data mining, and even e-mail spam filtering [1]. It is an emerging technology which has had a significant development in the AI area. [11, 14].

A neural network is an interconnected group of neurons, inspired by the structure of a living being's brain. Each neuron inside a neural network has a weight value and they are grouped in layers, the first layer being the input layer, followed by an arbitrary number of hidden layers and an output layer that gives the neural network's solution to the input presented. The input data travels through the neural network and is modified by each neuron weight until it reaches the output layer and the solution is given. Neural network implementations go through a training phase which covers thousands of examples to learn from. Each iteration of the learning phase processes a single instance of the training set and produces a neural network more accurate than it previously was. The same set of data is processed many times as the network is continually refined [11, 14].

Neural networks were currently implemented as software [10, 11, 14, 19]. Nowadays arises the desire to implement them as hardware for embedded systems. Hardware implementation of neural networks approaches vary from analog, digital, optical and on FPGAs. All of them have different set of flaws (like accuracy loss, high space and power usage or low processing speed) and delays the commercial practice of such implementations [10]. Also, EDA tools cannot develop an efficient synthesis for neural networks in application-specified integrated circuit (ASIC) format. This approach requires a large silicon area due to a large number of logic gates and a high number of connections between layers which introduces routing congestion [2]. Neural networks are already being used by researchers to do logic synthesis [5], but it is yet to be discovered how to efficiently reproduce logic synthesis to design a neural network circuit [2, 7, 13].

To improve the design of neural networks in hardware, we adopt an approximate computing strategy. We insert small acceptable errors to decrease the project's cost. This is a common approach on the design of arithmetic, energy efficient or quality configurable

SBCCI '19, August 26–30, 2019, Sao Paulo, Brazil

ACM ISBN 978-1-4503-6844-5/19/08…$15.00
https://doi.org/10.1145/3338852.3339874

circuits [6, 15–17]. In the proposed approach we use the training set to extract the primary inputs' signal probabilities. We might then consider signals with high probability values to be constants which allow us to apply simplifications on the circuit. This process is done on already trained neural network circuits represented as AIGs.

The rest of this work is presented as follows. We first present some preliminary information to better understand the proposed work. The next section presents the proposed method to reduce the node count in neural networks represented as AIGs. Afterward a section with results presents experiments done on neural network AIGs and the accuracy impact due to the reduction applied to them. Lastly a conclusion section closures the work and the reference listing is shown.

2 PRELIMINARIES

This section first presents the definition of an AIG structure. Then the configuration and purpose of the neural networks used in this work is presented. At last, an explanation of how the neural networks were turned in to a Boolean circuit is shown.

2.1 And-Inverter Graph

An And-Inverter Graph is a Directed Acyclic Graph (DAG) composed of primary inputs, primary outputs, and AND nodes. All internal AND nodes have exactly two inputs and an arbitrary number of outputs. Direct or negated edges connect the nodes. Continous lines represent a direct edge, while dotted lines are the negated ones. Figure 1 shows an AIG representation of three input XOR logic function. Since the AIG is build with AND and NOTs (negated edges), they can represent any logic circuit. A popular format for writing AIGs is the AIGER format, which is used in this work to write AIG files [3]. The AIG data structure is broadly used on logic synthesis to apply technology independent optimizations on circuits [9, 18].

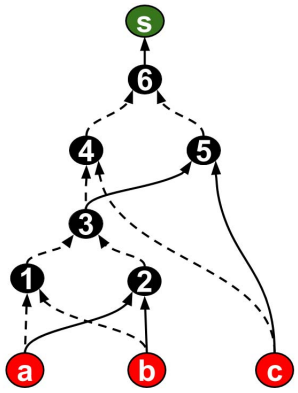


Figure 1: XOR in AIG representation

2.2 Neural Network Represented as an AIG

The database used to train and test the neural networks is the MNIST Database of Handwritten Digits [4]. The training set consists of 60,000 images and the test set 10,000 images. The images are handwritten single digits ranging from 0 to 9, written by high-school students and employees from a Census Bureau company. Those images that represent digits have 28 by 28 pixels. Each pixel is composed of a byte ranging from 0 to 255. This range is a color tone of grey: 0 means background (white) and 255 means foreground (black).

Neural networks were created and trained using the MNIST Database. Those neural networks were then transformed in to gate-level circuits. Each layer of the neural network is modeled as a sequence of multiply-add operations with optional RELUs.

The first step in converting multiply-add operations into a Boolean circuit, is to perform quantization of the floating-point numbers deriving by training [12]. It was found that enough precision, compared to the floating-point numbers, can be obtained by using 6-bit fixed-point computation, with coefficients represented as signed 8-bit numbers in the range between -2.0 and 2.0, while the intermediate values are represented as signed 16-bit numbers. The coefficients that did not fit into the range, were saturated. For example, if a value was -2.5, it was capped at -2.0.

Next, we derived a single gate-level circuit for the neural network by substituting a circuit for each multiply-add operation in the order of their appearance in the neural network [12]. Since each multiplier performs multiplication by a specific constant, one way of generating a circuit is to pre-compute the minimum sequence of shifted adders/subtractors needed to perform the multiplication. For example, multiplying by 5 requires one addition: $5x = 4x + 1x$, while multiplying by 11 required one addition and one subtraction $11x = 8x + 4x - 1x$. The coefficients whose values is 2^k are modeled as bit-vector shifts by k bits.

The RELUs were converted into circuits as multiplexers controlled by the output of a comparison operator, which selects between the number (if it is positive) and zero (if it is negative). Once the the circuits were in a gate-level description they could be easily transformed in to an AIG description.

3 PROPOSED METHOD

This section presents the method used to reduce the size of the neural network in AIG representation. The sub-sections are presented in the order of which the process is executed. We first define the input signals probabilities so we may set some of the primary input signals in the AIG as constants or nearly constants. Then we propagate those constants along the AIG which enables us to remove AND nodes from the AIG. Figure 2 presents a flowchart of the reduction process for a neural network AIG. The assignment of constants brings the occurrence of don't care nodes which allows the AIG's simplification [8, 16, 17].

3.1 Constant and Near Constant Signals

To apply the image on the AIG each of the image's bits are mapped into a primary input in the AIG. Meaning that the number of primary inputs in the AIG is the total number of bits that represent a single image: $28 * 28 * 8 = 6272$.

The database has each digit in the center of the image, by setting a pixel that is its center of mass and then translating the image to position this point at the center of the 28 by 28 field. We may take

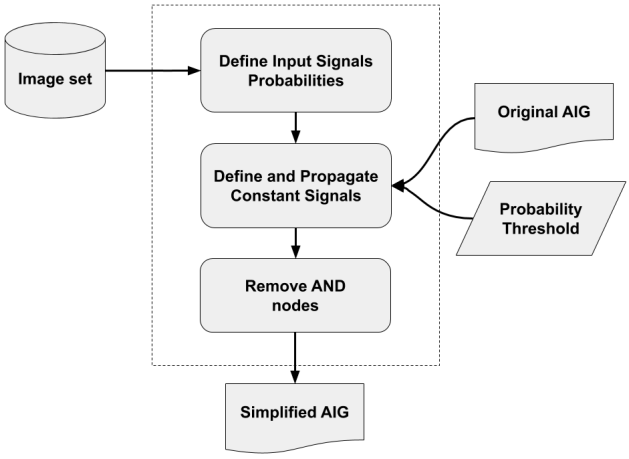


Figure 2: Flowchart.

advantage of this circumstance by ignoring the bits in the image corners in which are most likely not used.

Figure 3 shows each primary input probability of the AIG of being 0; this probability is calculated based on each bit in all the images in the training set. The figure area is the same as an image in the database, 6272 bits. They are ranging from 100% to 43.7%, with green color for larger, yellow for intermediate and red for smaller probability values. It clearly shows how the primary inputs that represent the image's edges are not often set to 1; there are 697 primary inputs that were never set to 1 among all the 60,000 training images. This set of primary inputs may be defined as constants as they have 100% probability of being 0.

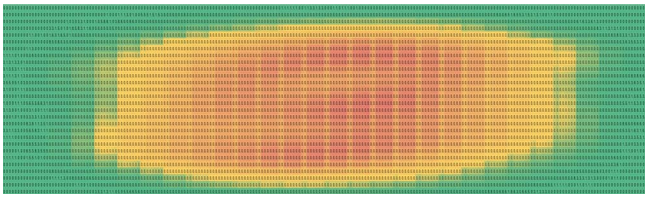


Figure 3: Primary Inputs Probabilities.

This phenomenon in which the corner bits are not used is what allows us to execute this method to reduce the AIG size. Although other applications may happen to have the same phenomenon for other implementations of neural networks. The aspect of analyzing the input vectors to make use of an approximate computing strategy is the core instrument in which one should interpret the problem presented herein.

After setting the probability of each input signal, we can define which ones will be considered to be a constant. A threshold variable is used to do so. This threshold will directly influence the number of input signals that will become a constant. Notice that a constant signal means that this signal will always have the same Boolean value, either 1 or 0. In this situation, we can only assume that the Boolean value is 0, because there no primary input with high probability of being 1. Figure 4 presents the set of primary inputs that have 100% probability of being 0; in other words, the primary inputs shown are the ones within the probability threshold of 100%.

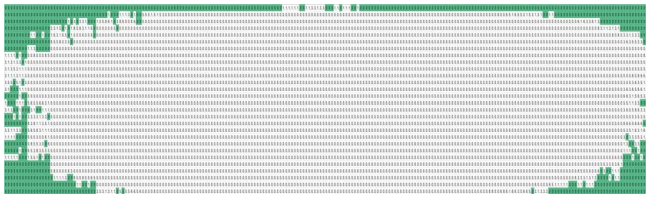


Figure 4: Primary Inputs Within Threshold Probability of 100%.

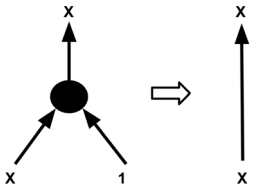
If those bits will always have the same value, there is no point in having logic in the circuit to process those values that never change; it is simply unnecessary. It also opens the possibility of considering inputs with high signal probability to be considered constants, in this case, the selected primary inputs, just like in figure 4, would embrace a larger portion of the image area, getting closer to the yellow region from figure 3. This is achieved by setting a lower probability threshold, although one should be careful because if we are too greedy using a threshold too low, the circuit might have too much nodes removed, it might have a contrary influence on the neural network ability to recognize the given input. It is worth mentioning the primary inputs that are not colored are left unchanged; in other words, the primary inputs that have a probability lower than the threshold.

3.2 AIG Reduction

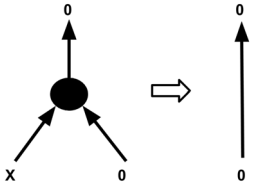
The AIG reduction consists of removing nodes from the AIG. To do so, we apply a Depth First Search (DFS) in the graph to propagate the constants. Not only the primary input nodes that are set to constants can be removed, but also every other AND node that the constant is propagated to. Figure 5 presents the behavior of an AND node when receiving a constant signal in two circumstances. The X represents any logic function that is not a constant. Due to the behavior of the logic AND function, we can be sure of the output from nodes that receive a constant value as input. In both situations, the AND node is removed. When the constant input is 1 (figure 5a) the constant and the node ceases to exist, while the non-constant signal is kept. When the constant is 0 (figure 5b), it still propagates along the circuit after removing the node.

A constant might be either 1 or 0, but it will not necessarily keep the same Boolean value while it travels the circuit, as an AIG consists of inverters also. A constant will be propagated along the circuit until one of two situations happens: 1. it reaches the circuit's primary output, in this case, all the logic ranging from a primary input to the primary output in which the constant propagated to is removed. 2. the constant reaches a node simultaneously with a non-constant, in this case, the constant ceases to exist, just like the case from figure 5a, and the non-constant is reconnected.

To illustrate an AIG node reduction based on constant propagation, figure 6, presents the XOR AIG from figure 1 with the input node B set to be a constant 0. Notice how the constant propagates until it reaches node 3 where it ceases to exist. Nodes 1 and 3 are



(a) Constant 1.



(b) Constant 0.

Figure 5: Reduction Behaviour With Constant Signals

conditioned to the case of figure 5a and node 2 to the case of figure 5b. In this situation, a new connection between primary input node *A* with nodes 4 and 5 is done. This new connection is to replace the removal of nodes 1, 2 and 3. The polarity has to be preserved during the assembling of new connections. For example, the path *A*->1->3->4 has three inversions, so the new connection between A and 4 is negated, while the path *A*->1->3->5 has two inversions, so the new connection between A and 5 is direct. In other words, an odd number of inversions along the path will produce a negated connection, while an even number of inversions will produce a direct connection between the nodes.

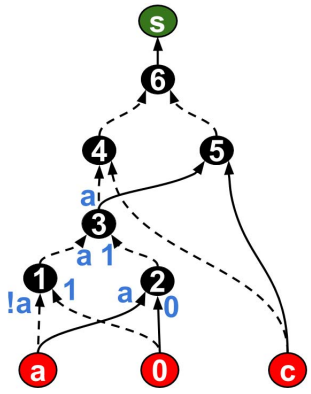


Figure 6: Constant Propagation In a 3 Input XOR AIG.

Figure 7 presents the result from the reduction of the 3 input XOR. Nodes 1, 2, 3 and primary input *B* from figure 6 were removed, remaining AND nodes are 4,5 and 6 which were renumbered. The renumbering of the nodes happens to preserve the AIGER language syntax while the proper connection between the nodes preserves the logic function of the AIG after reduction.

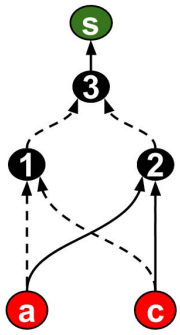


Figure 7: Reduction Sample.

4 RESULTS

To demonstrate the proposed method, experiments were performed for two different AIG circuits. These circuits are large neural networks trained to identify handwritten characters. The training set is composed of 60k character images. From the training set, it is possible to compute the probability of a input signal to be a 0 or 1 logic constant. In our experiment, we set different thresholds to consider an input signal as a constant.

Table 1 shows the results of the experiments done on the neural networks in the AIG format. Each circuit was simplified based on the number of input signals that are set to be constants. The first line of each circuit is the circuit's original configuration. The subsequent lines present its accuracy in relation to the threshold probability to transform it's input signals into constants. The columns of table 1 are explained as follows: first the neural network circuit's name, next we have the probability threshold set for the experiment instance, the circuit's number of AND nodes after simplification, followed by the percentage of ANDs removed, the number of PIs after simplification, the percentage of PIs removed, at last the neural network accuracy after reduction.

If we wish to have precisely no drawback in the neural network accuracy, we can still have a considerable amount of reduction on the number of nodes in the AIG by considering only the input signals that relate to the image bits that have the same value on all instances of the training set. In other words, the threshold probability to transform the signal in to a constant for the input signals is 100%. In circuit A1 for example, even after removing 4,5% AND nodes and 33.5% primary inputs from the AIG it still produces the same results as the original one. Or we can reduce the threshold and consider more signals to be constants, making the circuit smaller in the cost of the neural network's being less accurate when evaluating the inputs.

For nearly constant signals, which are the ones that have less than 100% probability threshold, in both circuits they still produce acceptable results even after removing nearly half of the circuit's internal nodes. Even after removing 39.02% AND nodes and 80.4% of the PIs the A1 circuit predicted correctly 89.75% of the images.

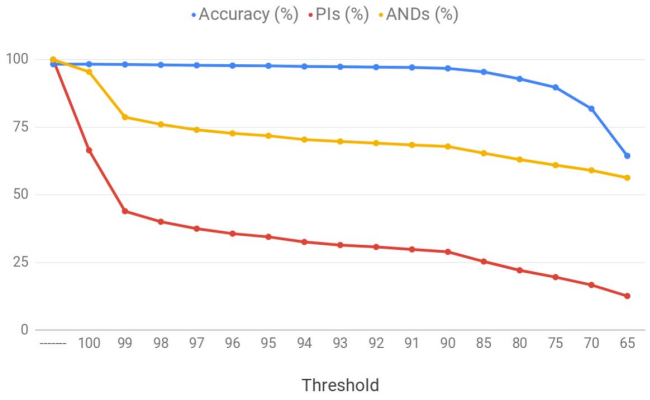
The executed experiments were used to create table 1 and the two graphics in figure 8a and 8b. The two graphics shows circuits A1 and A2 respectively and presents their behaviour under the simplification method proposed in this work. Both circuits have a

similar behaviour on accuracy loss due to the simplification applied. Higher threshold probability values show that the accuracy barely suffered from the AIG reduction, while the size reduction in the circuit is noticeable. It is only with threshold probabilities of 85% and lower that the accuracy loss starts to really deteriorate.

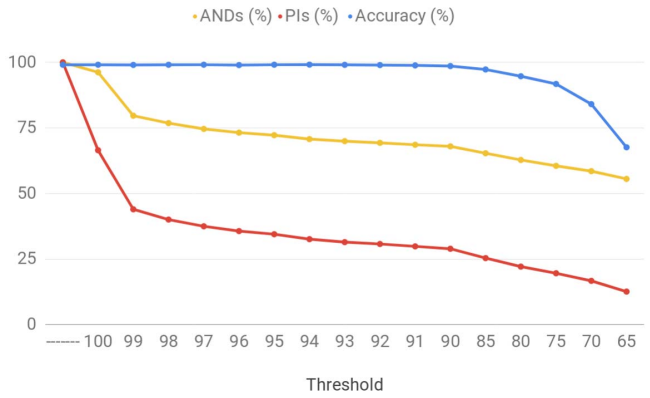
The results show that the method is viable to reduce the circuit's size, even with the cost of accuracy loss. If the threshold used to apply the proposed method is not too small, the circuit's loss in accuracy may be considered reasonable. Depending on the neural network application it is possible to use the proposed method to reduce the circuit size considerably without any major loss in the neural network's accuracy.

Table 1: Results Obtained for Neural Networks Circuit Simplification by Constant and Nearly-Constant Propagation

AIG	TH	# ANDs	ANDs Reduc. (%)	# PIs	PIs Reduc. (%)	Accuracy
	——	45103630	0.0	6272	0.0	98.32
	100	43075183	4.50	4170	33.51	98.32
	99	35514247	21.26	2758	56.03	98.21
	98	34298871	23.96	2513	59.93	98.05
	97	33398002	25.95	2352	62.50	97.91
	96	32809761	27.26	2237	64.33	97.79
	95	32406227	28.15	2162	65.53	97.72
	94	31773963	29.55	2044	67.41	97.47
A1	93	31462867	30.24	1973	68.54	97.36
	92	31184869	30.86	1929	69.24	97.23
	91	30879198	31.54	1872	70.15	97.11
	90	30618813	32.11	1815	71.06	96.78
	85	29502408	34.59	1591	74.63	95.47
	80	28442817	36.94	1387	77.89	92.87
	75	27503322	39.02	1229	80.40	89.75
	70	26646123	40.92	1047	83.31	81.86
	65	25404117	43.68	790	87.40	64.42
	——	53429856	0.0	6272	0.0	99.09
	100	51413680	3.77	4170	33.51	99.09
	99	42562997	20.34	2758	56.03	99.01
	98	41055821	23.16	2513	59.93	99.08
	97	39873257	25.37	2352	62.50	99.10
	96	39117329	26.79	2237	64.33	98.96
	95	38601829	27.75	2162	65.53	99.10
	94	37793190	29.27	2044	67.41	99.13
A2	93	37383526	30.03	1973	68.54	99.05
	92	37037408	30.68	1929	69.24	98.96
	91	36646755	31.41	1872	70.15	98.84
	90	36318062	32.03	1815	71.06	98.59
	85	34911947	34.66	1591	74.63	97.30
	80	33554008	37.20	1387	77.89	94.72
	75	32348062	39.46	1229	80.40	91.77
	70	31273517	41.47	1047	83.31	84.08
	65	29685735	44.44	790	87.40	67.62



(a) A1.



(b) A2.

Figure 8: Size Reduction and Accuracy Loss on Each Neural Network

5 CONCLUSION

This work presented a method to reduce neural networks implemented as AIG circuits. It was shown how to apply the method, and some experimental results were presented. There is still work to be done to find out how to properly implement neural networks as hardware, although this work shows a way around using approximate computing to find out an acceptable size reduction in circuit size with the cost of accuracy decrease. With smaller neural network circuits EDA tools may come closer to being able to design large neural network circuits.

ACKNOWLEDGMENTS

The work presented herein has been partially funded by CAPES and CNPq.

REFERENCES

[1] Jayesh Ahire. 2018. *Artificial Neural Networks: the Brain behind AI.* Lulu. com.
[2] Arash Ardakani, François Leduc-Primeau, Naoya Onizawa, Takahiro Hanyu, and Warren J Gross. 2017. VLSI implementation of deep neural network using integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 10 (2017), 2688–2699.
[3] A. Biere. 2007. AIGER Format And-Inverter Graph(AIG) Format. (2007).
[4] Li Deng. 2012. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine* 29, 6 (2012),

141–142.
[5] Winston Haaswijk, Edo Collins, Benoit Seguin, Mathias Soeken, Frédéric Kaplan, Sabine Süsstrunk, and Giovanni De Micheli. 2018. Deep learning for logic optimization algorithms. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.
[6] Jie Han and Michael Orshansky. 2013. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*. IEEE, 1–6.
[7] Kaiming He and Jian Sun. 2015. Convolutional Neural Networks at Constrained Time Cost. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
[8] Alan Mishchenko, Robert Brayton, Jie-Hong R. Jiang, and Stephen Jang. 2011. Scalable Don'T-care-based Logic Optimization and Resynthesis. *ACM Trans. Reconfigurable Technol. Syst.* 4, 4, Article 34 (Dec. 2011), 23 pages. https://doi.org/10.1145/2068716.2068720
[9] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. 2006. DAG-aware AIG rewriting a fresh look at combinational logic synthesis. In *Proceedings of the 43rd annual Design Automation Conference*. ACM, 532–535.
[10] Janardan Misra and Indranil Saha. 2010. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing* 74, 1-3 (2010), 239–255.
[11] Michael A Nielsen. 2015. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA:.
[12] Behrooz Parhami. 2010. *Computer arithmetic*. Vol. 20. Oxford university press.
[13] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16)*. ACM, New York, NY, USA, 26–35. https://doi.org/10.1145/2847263.2847265
[14] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.
[15] Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. 2013. Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1367–1372.
[16] Yi Wu and Weikang Qian. 2016. An Efficient Method for Multi-level Approximate Logic Synthesis Under Error Rate Constraint. In *Proceedings of the 53rd Annual Design Automation Conference (DAC '16)*. ACM, New York, NY, USA, Article 128, 6 pages. https://doi.org/10.1145/2897937.2897982
[17] Yue Yao, Shuyang Huang, Chen Wang, Yi Wu, and Weikang Qian. 2017. Approximate disjoint bi-decomposition and its application to approximate logic synthesis. In *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 517–524.
[18] C. Yu, M. Ciesielski, and A. Mishchenko. 2018. Fast Algebraic Rewriting Based on And-Inverter Graphs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 9 (Sep. 2018), 1907–1911. https://doi.org/10.1109/TCAD.2017.2772854
[19] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.