Spiking Neural Network Data Reduction via Interval Arithmetic

Spiking Neural Network Data Reduction via Interval Arithmetic

Spiking Neural Network Data Reduction via Interval Arithmetic

Sepide Saeedi[1], Alessio Carpegna[1], Alessandro Savino[1], and Stefano Di Carlo[1]
[1]*Control and Computer Eng. Dep., Politecnico di Torino* Torino, Italy
{sepide.saeedi, alessio.carpegna, alessandro.savino, stefano.dicarlo}@polito.it

***Abstract*—Approximate Computing (AxC) allows reducing the accuracy required by the user and the precision provided by the computing system to optimize the whole system in terms of performance, energy, and area reduction. Spiking Neural Networks (SNNs) are the new frontier for artificial intelligence because they better represent the timing influence on decision making, and also allow for a more reliable hardware design. Unfortunately, this design requires some area minimization strategies when the target hardware reaches the edge of computing. This seminal work introduces modeling of the approximation for data storage that supports an SNN via Interval Arithmetic (IA) by extracting the computation graph of the SNN and then resorting to IA to quickly evaluate the impact of approximation in terms of loss in accuracy without executing the network each time. Experimental results comparing our model to the real network confirm the quality of the approach.**

***Index Terms*—approximate computing, spiking neural networks, interval arithmetic**

I. Introduction

Using approximate computing techniques, it is possible to trade off the accuracy of the computation and results for gains in performance such as reductions in power consumption, memory utilization, and execution time [1]. Therefore, approximate computing techniques are widely employed in applications that are intrinsically tolerant to some accuracy loss. Among those applications, Artificial Neural Networks (ANN) are certainly a large group [2]. ANNs perform billions of arithmetic operations and require a lot of memory for saving millions of weights and activations. As a result, ANNs can benefit from approximation techniques to gain performance, especially when the deployment target is a low-power device [3]–[5].

Approximate computing solutions range from circuit-specific techniques, e.g., targeting adders or multipliers [6], to data-specific techniques, e.g., fixed- and floating-point scaling [7], [8]. All techniques can be exploited together and with different tuning of their parameters, making the choice of different approximate computing techniques difficult. Moreover, the exploration requires assessing the accuracy impact of the choices and, if necessary, weighting it with other design parameters, such as power consumption. To support the exploration of approximate techniques and their parameters' tuning, the two most common approaches comprises (i) running the application several times with different configurations [9], [10] or (ii) devising modeling techniques to evaluate them in time-optimized fashion [11], [12]. The first approach suffers from the limitation of requiring an evaluation time that grows when the exploration reaches an exhaustive search. To limit that, most of the time, pruning techniques to the exploration space are applied. In any case, the time required is bounded to the time necessary to run a single application instance. In the ANN case, it may translate into running the whole test set. On the contrary, modeling techniques are more suitable for quickly estimating the impact of a given approximation configuration at the cost of some error margin in the accuracy evaluation.

Among ANNs [13], Spiking Neural Networks (SNNs) [14] are an emerging class of ANNs in which information is exchanged between neurons in the form of binary spikes. SNNs have shown a great potential for achieving high accuracy, requiring a small area footprint, and thus potentially limiting the power/energy consumption due to their sparse spike-based operations [15]. Moreover, SNNs reduce the width of the connections between neurons to a single bit and support unsupervised learning with unlabeled data using the Spike-Timing-Dependent Plasticity (STDP) [16]. In the SNN, the computation flow of each neuron consists of arithmetic operations, i.e., additions, subtractions, multiplications, and necessary weights and thresholds to operate. Hence, when the SNN requires to be deployed into FPGA devices, it may be helpful to reduce the complexity of the arithmetic components and the storage of weights and thresholds using approximate computing techniques.

In this paper, we propose a simplified model of an SNN exploiting the Interval Arithmetic (IA) [17] concepts to evaluate the impact of the data reduction of trained weights and thresholds after the integer quantization. Previous works [18], [19] already exploited IA to analyze the effect of the computation of neural networks in general. Still, they modeled the overall functionality of the network and kept track of every single value propagating during the calculation. In contrast, in this work, we aim to reduce the impact of the analysis by compacting the representation to distinguish the error range and support further tuning optimizations based on that. It is important to note that in this paper, the word "error" refers to the error introduced or propagated by applying approximation techniques to the computations, not the error in classification by the SNN. The experiments have been conducted on a pre-trained spiking neural network [20], exploring the pruning of the parameters' size into the inference model to be transferred into the FPGA. To assess the quality of the model, all the approximated versions outputs of the SSN are compared with

the results elaborated using the IA-based model outputs.

The rest of the paper is organized as the following: In section II the entire methodology is described. First, an overview of the single neuron and the SNN architecture is provided. Then the approximation method is explained, followed by a description of the interval arithmetic-based modeling. Finally, the computation flow of the network and the usage of these methods are discussed. In section section III, the experimental results are provided and analyzed. Eventually, section IV draws conclusions and possible future works.

II. Methods

This section describes the original SNN and how the precision reduction on trained SNN data is applied. Eventually, the IA modeling explanation is provided.

A. Spike Neuron Model and Network Description

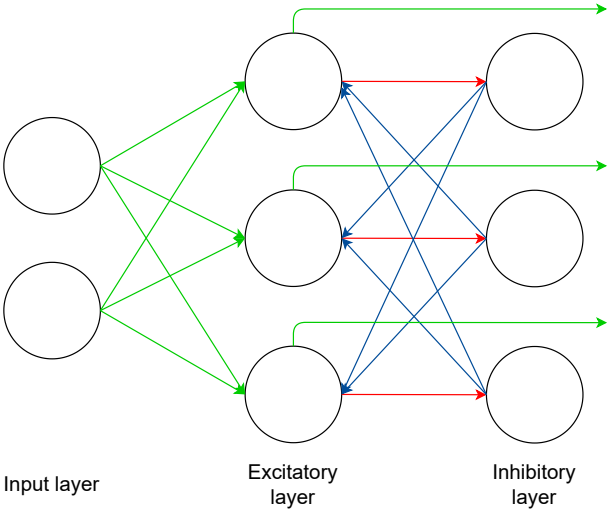

Fig. 1: An excitatory and an inhibitory layer of a SNN

Figure 1 depicts the general organization of an SNN, describing the behavior of each single spiking neuron in terms of excitatory and inhibitory layers. Spikes are generated by an input layer transforming static data, i.e., pixels of an image, into a sequence of spikes, enter the neuron through an excitatory layer and propagate back through an inhibitory layer. To appreciate the computation flow more clearly, Figure 2 shows the organization of a single layer.

The excitatory layer is responsible for increasing the *membrane potential* (V_0 in Figure 2) when input spikes are detected. Each spike is represented as a single bit set to 1. The increment is based on a weight value elaborated during the training phase and associated with each connection to the neuron ($w_{0,i}$ in Figure 2). Each time spikes enter the neuron, V_0 increases by the sum of all weights where an active spike is detected. The *membrane potential* behaves like an "electrical charge." When it reaches its maximum, the neuron fires a spike (green and red connections out of the excitatory layer in Figure 1 and $V_0 > V_{thresh}$ in Figure 2), and it is reset to a special *reset* value (V_{reset} as it can be seen following the "yes" branch in Figure 2). V_{thresh} defines the threshold potential for each neuron computed at training time. The inhibitory layer is responsible for decreasing the potential of other neurons when a spike is happening. Eventually, the *membrane potential* also falls at each step if no active spike is detected in the input using an exponential decay approach ($V_0 \gg exp_{decay}$ in Figure 2). The decay is a far more complex operation involving multiplication for an exponential time-related term but, to occupy less area on an FPGA, the work in [20] proposes a transformation of the exponential term into the closest power of two, to replace the multiplication with a shift. Since the exponent is negative, the actual shift is a right shift.

Eventually, each neuron is associated with a counter to keep track of the output spikes generated by its excitatory layer. They can be defined as the output of the SNN. Depending on the network purpose, an extra layer can be placed after the counters to make any classification decisions. For more detailed information about the model and implementation, the reader may refer to [20].

B. Data Quantization and precision reduction

The training task is responsible for computing, for each neuron, the set of weights and thresholds and the reset value necessary to operate the network. Since most of the training tools available compute those values in a floating-point fashion [21], an extra step is required to deliver the network into edge devices provided by some FPGA accelerators. To reach this goal, in [20], a quantization of the weights has been performed to reduce the memory occupation.

The quantization technique requires converting each floating-point value into a 32-bit wise fixed-point value, with the fractional part of 16 bits. Then a precision reduction of the fractional part is introduced to provide an approximated version of weights, thresholds, and reset values. The precision reduction ranged from 16 to 1 single bit. Since the integer part is not altered, each time the fractional part is reduced, the produced error ϵ follows Equation 1, where k is the number of bits removed from the fractional part and b_i the value of the original bit removed.

$$\epsilon = \sum_{i=0}^{k} b_i \cdot 2^{-(16-i)} \tag{1}$$

For a general value undergoing a precision reduction of k bits, the minimum and the maximum errors introduced are defined in Equation 2. The maximum corresponds to all bits cut equal to one, while the minimum to zero.

$$\underbrace{0}_{\forall b_i=0} \leq \epsilon \leq \underbrace{\sum_{i=0}^{k} \cdot 2^{-(16-i)}}_{\forall b_i=1} = -\frac{1-2^k}{32768} \tag{2}$$

It is crucial to notice that the precision reduction is not altering the integer part: at each drop, the number of integer bits remains 16.

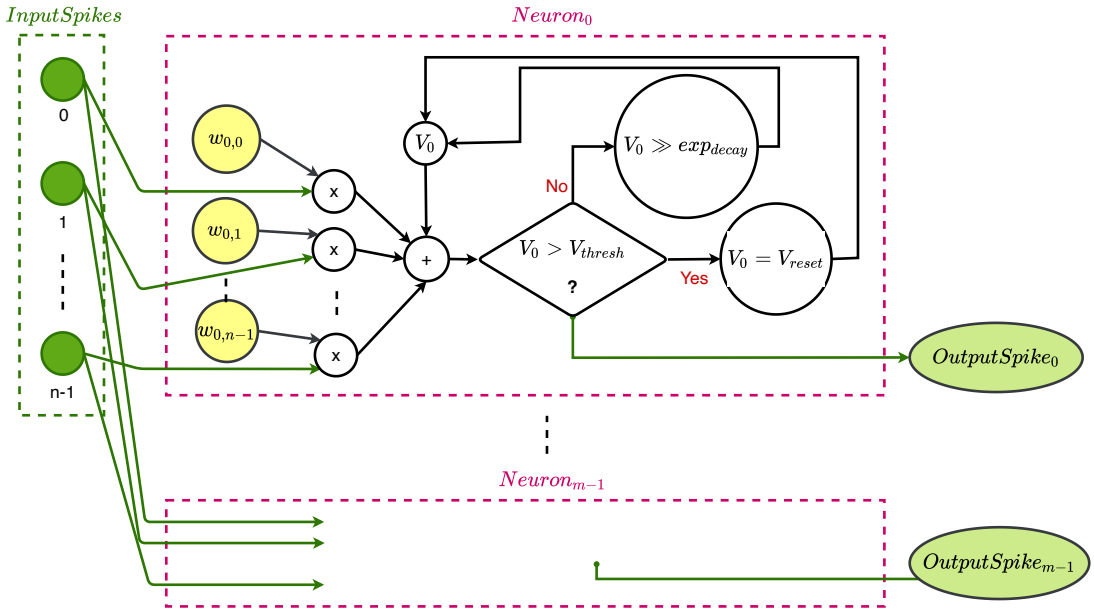

Fig. 2: Computation Flow of SNN layer

C. The Interval Arithmetic Error Propagation Model

Using the error computed in subsection II-B, it was possible to devise an Interval Arithmetic (IA) error propagation model. The foundation of IA is the ability to define mathematical operations over intervals. The most simple arithmetic operations are described, such as addition and multiplication. The standard way to refer to an interval is by resorting to the brackets notation in Equation 3.

$$[v] \equiv [v_1, v_2] \tag{3}$$

Given the general definition of interval arithmetic error-based value ($|ian|$), each value undergoing a precision reduction can be described as a pair of intervals $\{[v], [\epsilon]\}$, where $[v]$ is the range of the original values and $[\epsilon]$ the range of errors that the precision reduction may produce in the value interval. It is crucial to notice that:

1) this is a simplification because the limits of the $[v]$ interval may display errors that belong to $[\epsilon]$ but they are not necessarily its limit. This is a huge difference with respect to previous works such as [19].
2) the errors come from Equation 2, thus they are strictly monotonic, i.e., $[\epsilon] \equiv [\epsilon_{min}, \epsilon_{max}]$. The same applies to $[v]$.
3) due to the nature of the precision reduction operation, the relationship between $[v]$ and $[\epsilon]$ is subtractive.

The second and third considerations are critical to defining the mathematical operations required to translate all approximated values, i.e., weights, thresholds, and reset values, into $\{[v], [\epsilon]\}$ intervals and being able to model the same operations as in the original model. The standard [17] only defines a subset of them and only for a single interval, not for a pair. The complete set of required operations is quickly described in the following subsections.

1) Addition / Subtraction: The addition and the subtraction among two values, i.e., $|ian_1|$ and $|ian_2|$ can be defined using the linearity of the two mathematical operations and the monotonic shape of the ranges as in Equation 4 and Equation 5.

$$|ian_1| + |ian_2| = \{\underbrace{[v_{1_{min}} + v_{2_{min}}, v_{1_{max}} + v_{2_{max}}]}_{[v]}, \underbrace{[\epsilon_{1_{min}} + \epsilon_{2_{min}}, \epsilon_{1_{max}} + \epsilon_{2_{max}}]}_{[\epsilon]}\} \tag{4}$$

$$|ian_1| - |ian_2| = \{\underbrace{[v_{1_{min}} - v_{2_{max}}, v_{1_{max}} - v_{2_{min}}]}_{[v]}, \underbrace{[\epsilon_{1_{min}} - \epsilon_{2_{max}}, \epsilon_{1_{max}} - \epsilon_{2_{min}}]}_{[\epsilon]}\} \tag{5}$$

2) Multiplication: The multiplication is a special case in which the common IA applied to a single range does not suit our modeling. In fact, since the $|ian| \simeq [v] - [e]$ from a single value standpoint, the multiplication requires more complex management. Equation 6 describes the steps toward a proper $|ian_1| \times |ian_2|$ evaluation.

$$|ian_1| \times |ian_2| = \underbrace{[min(A), max(A)]}_{[v]}, \underbrace{[min(C) - min(B), max(C) - max(B)]}_{[\epsilon]} \quad (6)$$

The three components A, B, and C come from the evaluation of $|ian_1| \times |ian_2|$ as $([v_1] - [e_1]) \times ([v_2] - [e_2])$ and are fully developed in Equation 7. It is important to highlight that the modeling correctly relates the global error after the operations with the values (even in IA fashion) involved in the multiplication.

$$\begin{aligned} A \rightarrow \{&v_{1_{min}} \times v_{2_{min}}, v_{1_{min}} \times v_{2_{max}}, \\ &v_{1_{max}} \times v_{2_{min}}, v_{1_{max}} \times v_{2_{max}}\} \\ B \rightarrow \{&v_{2_{min}} \times \epsilon_{1_{min}}, v_{1_{min}} \times \epsilon_{2_{min}}, \\ &v_{2_{min}} \times \epsilon_{1_{max}}, v_{1_{min}} \times \epsilon_{2_{max}}, \\ &v_{2_{max}} \times \epsilon_{1_{min}}, v_{1_{min}} \times \epsilon_{2_{min}}, \\ &v_{2_{max}} \times \epsilon_{1_{max}}, v_{1_{min}} \times \epsilon_{2_{max}}, \\ &v_{2_{min}} \times \epsilon_{1_{min}}, v_{1_{max}} \times \epsilon_{2_{min}}, \\ &v_{2_{min}} \times \epsilon_{1_{max}}, v_{1_{max}} \times \epsilon_{2_{max}}, \\ &v_{2_{max}} \times \epsilon_{1_{min}}, v_{1_{max}} \times \epsilon_{2_{min}}, \\ &v_{2_{max}} \times \epsilon_{1_{max}}, v_{1_{max}} \times \epsilon_{2_{max}}, \} \\ C \rightarrow \{&\epsilon_{1_{min}} \times \epsilon_{2_{min}}, \epsilon_{1_{min}} \times \epsilon_{2_{max}}, \\ &\epsilon_{1_{max}} \times \epsilon_{2_{min}}, \epsilon_{1_{max}} \times \epsilon_{2_{max}}\} \end{aligned} \quad (7)$$

3) Right Shift: The last operation that is useful for the neuron is the right shift for the operation involving the exp_{decay}. Equation 8 summarizes the modeling of that operation for a $|ian|$.

$$|ian_1| \gg n = \{\underbrace{[v_{1_{min}} \gg n, v_{2_{max}} \gg n]}_{[v]}, \underbrace{[\epsilon_{1_{min}} \gg n, \epsilon_{2_{max}} \gg n]}_{[\epsilon]}\} \quad (8)$$

4) Comparison: One of the non-re-usable parts in the IA theory is the definition of the comparison between intervals. The model implements the decision making of the $|ian_1| > |ian_2|$ operator, using the definition of subtraction and applying a range reasoning on the results when compared to 0, such as $|ian_1| - |ian_2| > 0$. Equation 9 reports the evaluation of such comparison in the IA modeling.

$$\begin{aligned} |ian_1| - |ian_2| = |ian_r| > 0 \\ \Rightarrow (v_{r_{max}} - \epsilon_{r_{max}}) - (v_{r_{min}} - \epsilon_{r_{min}}) > 0 \end{aligned} \quad (9)$$

Basically, due to the monotonic property, if the difference of two endpoints of the bounds (including the errors) falls into the positive field, the comparison is true, making the comparison a majority voting. Again, in this case, this is a simplification to allow a fast model evaluation, and it might introduce some errors. In fact if both $(v_{r_{max}} - \epsilon_{r_{max}})$ and $(v_{r_{min}} - \epsilon_{r_{min}})$ are of the same sign, the comparison result is the same for any value in the interval, but if they have opposite signs, in the reality, they model a scenario where the real comparison might be either true or false depending on the value in the interval.

III. Experimental Results

After modeling the approximation on all necessary operands, a data computation flow of the SNN was extracted. The flow is similar to the one depicted in Figure 2, where all values are $|ian|$ and mathematical operations are implemented accordingly to subsection II-C. The model's input is the sequences of spikes, and the output is the sequences of produced spikes.

The trained SNN from [20] is used in this paper. To train and test the network, the MNIST dataset was used as reference [22]. The MNIST dataset comprises 60,000 training images and 10,000 testing images of handwritten digits. Since the resolution of each image is 28x28 pixels, in total, 784 pixels of an image are used to produce input spikes for the SNN. The SNN defines an input layer to transform each image pixel into a sequence of spikes using a random Poisson process [23] with an average frequency corresponding to the numeric value in the input. After this process, each image results in 3500 spikes.

The SNN structure includes one layer with 400 neurons. The membrane potential of each neuron is reset to its rest potential at the end of each image. The network's outputs include: (i) the associated spike counter and (ii) the classification decision made by the final decision layer. This network was first trained in full precision (floating-point), then all parameters were converted to 32-bit fixed-point values, verifying that the classification results were comparable. The experimental setup of this work starts after this conversion.

Figure 3 depicts the experimental workflow for each k value of precision reduction. All data from the original SNN network are extracted and stored for further use. Then, the previous work implemented the red part of the workflow. Once the k value is set, all data require re-computation to apply the precision reduction, and then the approximated SNN can run. Since we aim to assess the quality of the IA error-based modeling in this work, the final classification is not of interest. The primary output is the counting of the spikes because it is where the IA modeling ends with the comparison described in subsection II-C and replacing the $V_0 > V_{thresh}$ of the original network. Of course, the error introduced by approximation may lead to errors in different output spike counts which may affect the final classification adversely. However, it is important to note that the classification itself is not part of this work. Hence, Comparing the SNN classification accuracy of our model to the precise SNN, is just supplemental proof of the quality of the model.

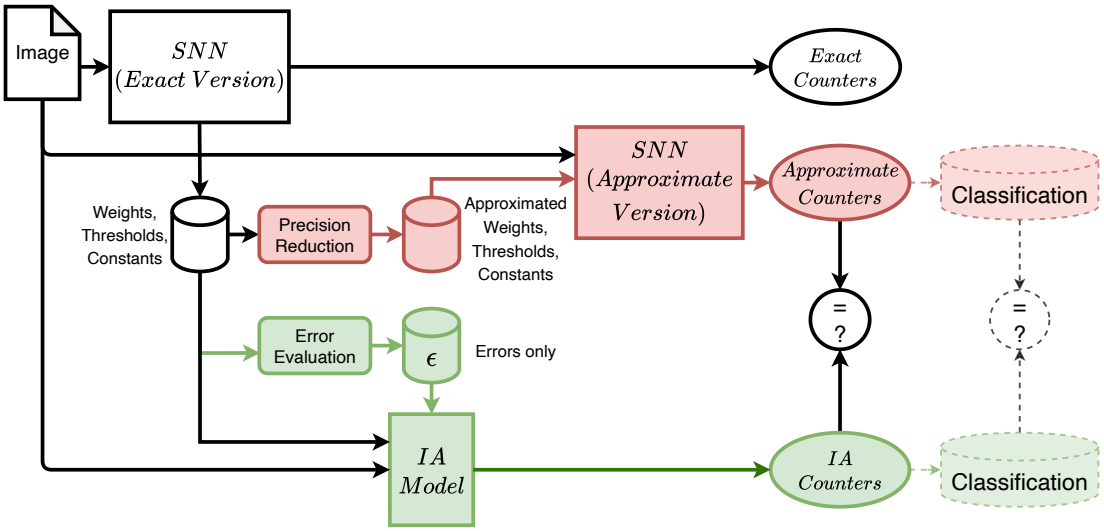

Fig. 3: Experimental workflow. For each image and k precision reduction, the red part evaluates the real values running the full network, while the green part depicts the model where only error are computed for each k.

The green blocks in Figure 3 describe the IA error-based modeling. Once the network data is defined, from each k, an automated procedure derives the $[\epsilon]$ part of each value to describe the $|ian|$ versions adequately. The process relies on 1 to estimate the $[\epsilon]$ for every data of the network undergoing the approximation. Since it's a pure mathematical elaboration, it is already faster and more flexible than evaluating all numbers anew. Then the model runs following the computational flow depicted in Figure 2 to compute the spike counter for each neuron. This is where the proposed approach still relies on all input spikes processed in the same way as the original network. The nature of the SNN requires that the sequence is respected to be able to compare the final results. Moreover, the input generated from the original SNN execution is stored and used for the model to make any counting comparable.

Since the primary goal of this work is to establish if the IA based-modeling is suitable for optimizing the SNN parameters in case of precision reduction exploitation, to check whether our model can produce similar outputs concerning the original model, two different sets of experiments have been conducted.

First, we run a complete comparison with $\epsilon = 0$. This is not the actual usage of the methodology, as described previously, but it is the proof that the model presented in subsection II-C is sound and solid. For all possible k in the exploration (from 0 to 15), the recomputed data used by the SNN have been used as the $[v]$ part of each $|ian|$ keeping the $[\epsilon]$ to zero. Being the error 0, the computation should display precisely the same results of the SNN without carrying any of the simplification errors that the modeling might introduce. Results confirmed it and, for all k, the counter difference was always 0, showing that the model using interval arithmetic is behaving the same as the original network. For the sake of simplicity, we avoid reporting these results here.

Second, we run for the same k precision reductions the complete model, computing only the errors and relying on the original data for the $[v]$ part. The results are reported in Table I. For each precision $(16 - k)$, the table displays the range of errors reported in the counters and the percentage of wrong counters among the 400 ones. Eventually, the last column reports whether the classification done using the counters computed from the model is still the same. Once again, it is crucial to highlight that the classification itself is not part of this work and this last information is just supplemental proof of the quality of the model.

As displayed by the Errors columns, for the first k reduction (from 0 to 3, referring to precision 16 to 13), the model does not show any discrepancy from the original SNN. Starting from $k = 4$ (precision 12), the simplification introduced by the error modeling introduces some deviation from the counting. A negative value indicates that the model is producing (hence counting) fewer spikes than the original network, while a positive value otherwise. Nevertheless, the average deviation is always negligible. The average increases for the higher k values (precision reduced to 6 bits or lower) confirm two facts: (i) that the simplifications are somehow present but not limiting the effectiveness of the modeling and (ii) that the SNN itself starts to be unreliable after a specific precision reduction. Both facts are linked to the fact that half of the weights in the network see the fractional part reduced to 0 when k reaches 10/11 bits.

However, looking at the counters as a whole, the worst wrong counter observed percentage is at 5-bit precision ($k = 11$). Still, it does not exceed 7%, which is still negligible because the classification output is the same. This confirms that the modeling approach is very reliable in predicting the propagation of the error introduced by approximation.

IV. Conclusion

We introduced modeling of the approximation for data storage that supports an SNN via Interval Arithmetic by

Precision (bits)	Errors			Wrong Counters (%)	Accuracy Comparison
	Min	Max	AVG		
16 (full)	0	0	0	0	Same
15	0	0	0	0	Same
14	0	0	0	0	Same
13	0	0	0	0	Same
12	-1	0	-0,0025	0,25	Same
11	-1	0	-0,0025	0,25	Same
10	-1	0	-0,005	0,5	Same
9	0	1	0,015	1,5	Same
8	-1	0	-0,0025	0,25	Same
7	-1	1	0	2,5	Same
6	-1	1	0,0275	5,25	Same
5	-1	1	-0,0625	6,75	Same
4	-1	0	-0,0575	5,75	Same
3	-1	1	0,015	3,5	Same
2	-1	0	-0,025	2,5	Same
1	-2	0	-0,035	2,5	Same

TABLE I: IA Model results on spike counters and related accuracy in prediction. Precision column accounts for the number of bits used, where 16 is a full precise reference. Errors columns display the minimum, maximum and average error (negative values corresponds to a lower counting in the IA model, while positive the opposite). Wrong counters column shows the percentage of counters showing a deviation from the original value, while Accuracy comparison reports if the final prediction is the same or not.

extracting the computation flow of the SNN and then resorting to IA to quickly evaluate the impact of approximation in terms of loss in accuracy without executing the network each time. This is especially important because exploiting the AxC techniques commonly goes into long exploration phases where the selected methods require tuning to balance the accuracy reduction with the expected improvements.

We conducted the experiments for a trained SNN executed in inference mode with the computation flow of the network modeled using interval arithmetic. We compared the results of different approximated network versions obtained with precision reduction techniques. Experimental results comparing our model to the original network confirm the quality of the approach. At the same time, the maximum percentage of wrong counters observed is only 6.75%, and the network accuracy is untouched.

For future work, we can extend the model to other Neural Networks and employ different approximation techniques to investigate this approach's opportunities further.

ACKNOWLEDGMENT

This work has received funding from the APROPOS project in the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956090.

REFERENCES

[1] P. Stanley-Marbell, A. Alaghi, M. Carbin, E. Darulova, L. Dolecek, A. Gerstlauer, G. Gillani, D. Jevdjic, T. Moreau, M. Cacciotti, A. Daglis, N. E. Jerger, B. Falsafi, S. Misailovic, A. Sampson, and D. Zufferey, "Exploiting errors for efficiency: A survey from circuits to applications," *ACM Comput. Surv.*, vol. 53, no. 3, jun 2020. [Online]. Available: https://doi.org/10.1145/3394898

[2] D. Wu and J. S. Miguel, "Special session: When dataflows converge: Reconfigurable and approximate computing for emerging neural networks," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021, pp. 9–12.

[3] T. Ayhan and M. Altun, "Approximate fully connected neural network generation," in *2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2018, pp. 93–96.

[4] Z. Peng, X. Chen, C. Xu, N. Jing, X. Liang, C. Lu, and L. Jiang, "Axnet: Approximate computing using an end-to-end trainable neural network," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[5] Z. Wang, M. A. Trefzer, S. J. Bale, and A. M. Tyrrell, "Approximate multiply-accumulate array for convolutional neural networks on fpga," in *2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2019, pp. 35–42.

[6] P. Balasubramanian and D. L. Maskell, "Hardware optimized and error reduced approximate adder," *Electronics*, vol. 8, no. 11, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/11/1212

[7] F. Cladera, M. Gautier, and O. Sentieys, "Channel-aware energy optimization of ofdm receivers using dynamic precision scaling in fpgas," in *2015 23rd European Signal Processing Conference (EUSIPCO)*, 2015, pp. 1571–1575.

[8] M. Traiola, A. Savino, M. Barbareschi, S. D. Carlo, and A. Bosio, "Predicting the impact of functional approximation: from component- to application-level," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, 2018, pp. 61–64.

[9] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer, "Neural networks designing neural networks: Multi-objective hyper-parameter optimization," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–8.

[10] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, "On the automatic exploration of weight sharing for deep neural network compression," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 1319–1322.

[11] M. Traiola, A. Savino, and S. Di Carlo, "Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications," *Microelectronics Reliability*, vol. 102, p. 113309, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271418309442

[12] A. Savino, M. Traiola, S. D. Carlo, and A. Bosio, "Efficient neural network approximation via bayesian reasoning," in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2021, pp. 45–50.

[13] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225 134–225 180, 2020.

[14] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[15] R. V. W. Putra and M. Shafique, "Fspinn: An optimization framework for memory-efficient and energy-efficient spiking neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3601–3613, 2020.

[16] ——, "Spikedyn: A framework for energy-efficient spiking neural networks with continual and unsupervised learning capabilities in dynamic environments," *arXiv preprint arXiv:2103.00424*, 2021.

[17] "Ieee standard for interval arithmetic," *IEEE Std 1788-2015*, pp. 1–97, 2015.

[18] H. Okada, T. Matsuse, T. Wada, and A. Yamashita, "Interval ga for evolving neural networks with interval weights and biases," in *2012 Proceedings of SICE Annual Conference (SICE)*, 2012, pp. 1542–1545.

[19] J. P. Turner and T. Nowotny, "Arpra: An arbitrary precision range analysis library," *Frontiers in Neuroinformatics*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/article/10.3389/fninf.2021.632729

[20] A. Carpegna, A. Savino, and S. D. Carlo, "Fpga-optimized hardware acceleration for spiking neural networks," 2022.

[21] M. Stimberg, R. Brette, and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, p. e47314, Aug. 2019.

[22] C. C. Yann LeCunn and C. J. Burges. The mnist database. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[23] D. Heeger, "Poisson model of spike generation," 10 2000.