Low Power Hardware Architecture for Sampling-free Bayesian Neural Networks inference

Antonios-Kyrillos Chatzimichail*†, Charalampos Antoniadis*, Nikolaos Bellas† and Yehia Massoud*
*Innovative Technologies Laboratories (ITL),
King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
{charalampos.antoniadis, yehia.massoud}@kaust.edu.sa
†Department of Electrical and Computer Engineering, University of Thessaly, Volos, Greece
nbellas@uth.gr

2023 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-5109-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISCAS46773.2023.10181925

***Abstract*—Standard NNs should not be employed mindlessly in critical applications due to their incapability to express the uncertainty of their predictions. On the other hand, Bayesian Neural Networks (BNNs) can measure the uncertainty of their predictions. There are two methods for BNN inference, the Monte Carlo-based method, which requires the sampling of weights distributions and multiple inference iterations, and moment propagation, where the mean and variance of a normal distribution are propagated through the BNN. Hardware implementations of moment propagation BNN inference consume less power than Monte Carlo because they complete the inference in a single forward pass. However, because the propagation of distribution moments through nonlinear activation functions leads to large hardware designs , these functions are usually approximated by polynomials. Hardware implementations of moment propagation have been studied solely for fully-connected neural networks while lacking optimal accuracy due to the approximation of the ReLU activation function with a single polynomial term. Therefore, in this work, we add one more polynomial term in the approximation of ReLU, providing better accuracy with negligible additional hardware. We also propose a polynomial approximation for another common activation function, tanh, and extend the hardware implementation to Convolutional Neural Networks (CNNs). Experimental results demonstrated that the proposed approximation of ReLU outperforms the previously suggested single-term polynomial by achieving up to 5.9% higher accuracy with merely up to $0.029\ W$ power overhead.**

***Index Terms*—Bayesian Neural Network, FPGA, Moment Propagation, ReLU polynomial approximation, tanh polynomial approximation**

I. Introduction

Artificial Neural Networks (ANNs) have shown noteworthy success in multiple fields of engineering and life, such as Electronic Design Automation [1], multimedia [2], autonomous driving [3], and predicting COVID-19 pandemic cases [4]. However, standard neural networks are unsuitable for critical applications, e.g., in healthcare systems and self-driving cars, that cannot tolerate mispredictions. On the other hand, Bayesian Neural Networks (BNNs) learn distributions as weights instead of scalars, to measure their predictions' uncertainty, and if the uncertainty is too high, they inform us that their prediction cannot be trusted.

In recent years, several works [5]–[7] have focused on hardware acceleration of BNN inference, targeting Field-Programmable Gate Arrays (FPGAs) as development devices to achieve high performance and energy efficiency. Among these works are two main approaches for the BNN inference: the Monte Carlo (MC) approach and the Moment Propagation approach. The former approach computes the average of N network inference evaluations, wherein for each network pass, the weights are obtained by random sampling of Gaussian distribution, providing different outputs on each evaluation [5]. An alternative MC suggested in [7] uses Monte Carlo Dropout, where conventional neural networks are trained, and the uncertainty quantification occurs from the dropout, which discards the neurons of the NN with some probability in every inference pass. As for the latter approach for implementing BNN inference, Moment Propagation, the neurons represent Gaussian distributions whose moments, i.e., mean and variance, propagate from the first to the last layer [6]. Even though the last method requires more computations for a single pass, since only one inference iteration is required, it leads to more power-efficient hardware implementations.

Moment propagation can be efficiently implemented in hardware for specific activation functions. For example, in the most recent hardware accelerator for Bayesian inference with moment propagation, BYNQNet [6], the quadratic activation function $f(x) = x^2$ has been used as a replacement for the Rectified Linear Unit (ReLU) activation function and tested in a fully-connected network using the MNIST dataset [8].

This work aims to widen the range of applications of the existing hardware implementation of BNN inference with moment propagation. The contributions of this work are:

- extension of the hardware implementation of moment propagation method to Convolutional BNNs
- enhancement of the polynomial approximation of $ReLU$ used in [6] ($\frac{1}{2}(x^2 + x)$ instead of x^2) to improve accuracy, and proposal of a $tanh$ polynomial approximation ($x - \frac{1}{3}x^3$) as activation function that produces an efficient hardware implementation of moment propagation
- evaluation of moment propagation performance in more complex datasets

The structure of the rest of the paper is as follows. Section II provides the background of Moment propagation in BNNs. Section III presents the proposed hardware architecture for BNN inference with moment propagation. Section IV justifies the higher accuracy with a small overhead in area and power of the proposed BNN hardware architecture, while Section V concludes the paper.

II. Background

A. Overview of Bayesian Neural Networks (BNNs)

BNNs learn probability distributions over the NN model parameters instead of scalar values, making their predictions also probability distributions. Therefore, BNNs can directly measure the uncertainty of their predictions through the output probability distribution by evaluating the average Predictive Entropy (aPE) over a dataset:

$$\text{aPE} = \frac{1}{D}\left[\sum_{d=1}^{D}\left(-\sum_{k=1}^{K} p(y_d^k|x_d)\log p(y_d^k|x_d)\right)\right] \quad (1)$$

where D is the dataset size, K is the number of classes and $p(y_d^k|x_d)$ is the probability of the BNN prediction y_d to fall in class k given input x_d.

A BNN considers a posterior probability distribution for the weights ($\mathcal{W}$) given a dataset $\mathcal{D}$, i.e., $P(\mathcal{W}|\mathcal{D})$ [5], [10]. Bayes rule states that the posterior probability distribution is given by:

$$P(\mathcal{W}|\mathcal{D}) = \frac{\mathcal{P}(\mathcal{D}|\mathcal{W})\mathcal{P}(\mathcal{W})}{\mathcal{P}(\mathcal{D})}, \tag{2}$$

where $P(\mathcal{W})$ is a prior probability distribution, $P(\mathcal{D}|\mathcal{W})$ is the likelihood and $P(\mathcal{D})$ is given by:

$$P(\mathcal{D}) = \int_{\mathcal{W}} \mathcal{P}(\mathcal{D}|\mathcal{W}')\mathcal{P}(\mathcal{W}')\,\mathrm{d}\mathcal{W}'. \tag{3}$$

Since the computation of (3), and consequently (2), is difficult, $P(\mathcal{W}|\mathcal{D})$ is approximated by a given distribution which is usually the normal distribution with trainable mean μ and standard deviation σ. Therefore, the weights take the form:

$$w = \mu + \epsilon\sigma \tag{4}$$

where $\epsilon \sim \mathcal{N}(0,1)$. In MC-based BNN inference, the output of N separate network inferences is averaged, with each inference using different sampled weights [5], [10]. On the other hand, moment propagation is a sampling-free method that is more power efficient for BNN inference.

B. Moment propagation in BNN inference

Each neuron carries two values in moment propagation, the mean and the variance of a Gaussian distribution, which are propagated through the network layers. The neurons of the input layer ($l = 0$) have zero variance ($\mathbb{V}[h_i^0] = 0$), and mean values equal to the input values of the dataset ($\mathbb{E}[h_i^0] = x_i^0$). The mean and variance of the pre-activations x_i^l (distributions as input to the activation function) of each next layer l are computed by:

$$\mathbb{E}[x_i^l] = b_i^l + \sum_j w_{\mu,ij}^l\,\mathbb{E}[h_j^{l-1}], \tag{5}$$

$$\mathbb{V}[x_i^l] = \sum_j \left[w_{v,ij}^l\,\mathbb{V}[h_j^{l-1}] + w_{\mu^2,ij}^l\,\mathbb{E}[h_j^{l-1}]^2\right], \tag{6}$$

where h_j^{l-1} refers to each of the previous layer's post-activations (neurons' output distributions), and b_i^l to the bias of i-th neuron of layer l. The parameters $w_{\mu,ij}^l = \mathbb{E}[w_{ij}^l]$, $w_{v,ij}^l = \left(\mathbb{V}[w_{ij}^l] + \mathbb{E}[w_{ij}^l]^2\right)$, and $w_{\mu^2,ij}^l = \mathbb{V}[w_{ij}^l]$, where w_{ij}^l is the weight between the i-th neuron of layer l and j-th neuron of layer $l-1$, are calculated before performing BNN inference.

The next part of the computations is the moment propagation through a non-linear activation function using the pre-activation moments. For this, operations that can not be implemented efficiently in hardware designs, such as Cumulative Distribution Function (CDF) computation, may need to be performed. On the contrary, moment propagation of polynomial activation functions is better implemented in hardware.

Because the polynomial approximations of non-linear activation functions are more precise in the range $[-1, 1]$, Batch Normalization (BN) layers are added before the activation function. BN layers recenter the pre-activations around 0 and re-scale them to maintain a standard deviation of 1. Thus, pre-activations are more likely to be in the range $[-1, 1]$. BN layers can be embedded in hidden layers so that $w_{\mu,ij}^l$, $w_{v,ij}^l$, $w_{\mu^2,ij}^l$ and b_i^l parameters in (5) and (6) are transformed into:

$${w_{\mu,ij}^l}' = \frac{\gamma_i^l}{\sqrt{(\sigma_i^l)^2 + \epsilon}} w_{\mu,ij}^l, \tag{7}$$

$${w_{v,ij}^l}' = \frac{(\gamma_i^l)^2}{(\sigma_i^l)^2 + \epsilon} w_{v,ij}^l, \tag{8}$$

$${w_{\mu^2,ij}^l}' = \frac{(\gamma_i^l)^2}{(\sigma_i^l)^2 + \epsilon} w_{\mu^2,ij}^l, \tag{9}$$

$$b_i^{l\prime} = \beta_i^l + \gamma_i^l \frac{b_i^l - \mu_i^l}{\sqrt{(\sigma_i^l)^2 + \epsilon}}, \tag{10}$$

where the BN parameters γ_i^l, β_i^l, μ_i^l, σ_i^l, and the constant number ϵ have already be defined in training.

III. Proposed method

A. Enhanced quadratic approximation for ReLU

In [11], the authors empirically show that x^2 is not the ideal second-degree polynomial to replace ReLU. Instead, they propose $\frac{1}{2}(x^2+ax)$ as a better option for approximating ReLU in the range $[-a, a]$ (see Fig. 1a). For this function to be integrated into the moment propagation algorithm, its first and second moments need to be calculated.

Considering the pre-activation Random Variable (RV) x_i^l, the first moment of the post-activation RV through $\frac{1}{2}({x_i^l}^2 + ax_i^l)$ is:

$$\begin{aligned}\mathbb{E}[h_i^l] = \mathbb{E}[\frac{1}{2}({x_i^l}^2 + ax_i^l)] = \frac{1}{2}(\mathbb{E}[{x_i^l}^2] + \mathbb{E}[ax_i^l]) = \\ = \frac{1}{2}(\mathbb{V}[x_i^l] + \mathbb{E}[x_i^l]^2 + a\,\mathbb{E}[x_i^l]),\end{aligned} \tag{11}$$

where $\mathbb{E}[x_i^l]$ and $\mathbb{V}[x_i^l]$ are computed by (5) and (6). The second moment is given by[1]:

$$\begin{aligned}\mathbb{V}[h_i^l] = \mathbb{V}[\frac{1}{2}({x_i^l}^2 + ax_i^l)] = \\ = \frac{1}{4}\{(2\,\mathbb{V}[x_i^l](\mathbb{V}[x_i^l] + 2\,\mathbb{E}[x_i^l]^2) + a^2\,\mathbb{V}[x_i^l] + \\ + 4a * (\mathbb{V}[x_i^l]\,\mathbb{E}[x_i^l])\}.\end{aligned} \tag{12}$$

Due to Batch Normalization layers, the range of values for the pre-activations is $[-1, 1]$. Therefore, since $a = 1$, the proposed quadratic polynomial approximation function for ReLU is $f(x) = \frac{1}{2}(x^2 + x)$.

B. Proposed polynomial approximation of tanh

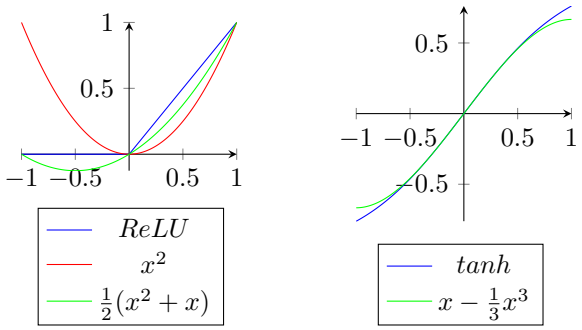
As for the tanh polynomial approximation, the first two nonzero terms of its Taylor expansion were chosen, i.e., $x - \frac{1}{3}x^3$ (see Fig. 1b). Considering a pre-activation RV x_i^l, the first moment of the post-activation RV through $x_i^l - \frac{1}{3}{x_i^l}^3$ is given by:

$$\begin{aligned}\mathbb{E}[h_i^l] = \mathbb{E}[x_i^l - \frac{1}{3}{x_i^l}^3] = \\ = \mathbb{E}[x_i^l] - \mathbb{V}[x_i^l]\,\mathbb{E}[x_i^l] - \frac{1}{3}\,\mathbb{E}[x_i^l]^3,\end{aligned} \tag{13}$$

[1]Due to the page limitation, a detailed derivation of the propagation of the first two moments through the proposed activation functions has been omitted.

and the second moment is given by:

$$\begin{aligned}\mathbb{V}[h_i^l] &= \mathbb{V}[x_i^l - \frac{1}{3}{x_i^l}^3] = \\ &= \mathbb{V}[x_i^l] + \mathbb{E}[x_i^l]^4\,\mathbb{V}[x_i^l] + 4\,\mathbb{E}[x_i^l]^2\,\mathbb{V}[x_i^l]^2 + \\ &+ \frac{5}{3}\,\mathbb{V}[x_i^l]^3 - 2\,\mathbb{E}[x_i^l]^2\,\mathbb{V}[x_i^l] - 2\,\mathbb{V}[x_i^l]^2.\end{aligned} \tag{14}$$



(a) (b)

Fig. 1: Comparison of (a) polynomial functions with ReLU and (b) tanh with its polynomial approximation

C. Moment propagation in convolutional and pooling layers

The computations in moment propagation for the convolutional layer share similar ideas with the fully connected layer. In essence, for each element of a layer's l output, there are two steps. In the first step, the mean and variance of the pre-activations, $\mathbb{E}[x^l]$ and $\mathbb{V}[x^l]$, are computed using (5) and (6), where, instead of dot products, convolution operations are performed between the post-activations of the previous layer $l-1$ and the pre-computed parameters w^l. Subsequently, the equations of moment propagation through the desired activation function are applied (for example, (11) and (12) with $a = 1$ in the case of $f(x) = \frac{1}{2}(x^2 + x)$).

Average and Max pooling layers slide a window over the input, maintaining the average or maximum value of the elements inside the window. In the case of moment propagation, the window consists of Gaussian distributions. For Average pooling layers, the output is a new Gaussian distribution with its mean and variance given by:

$$\mathbb{E}[h_i^l] = \frac{\sum_{j=1}^{N} \mathbb{E}[h_j^{l-1}]}{N}, \qquad \mathbb{V}[h_i^l] = \frac{\sum_{j=1}^{N} \mathbb{V}[h_j^{l-1}]}{N^2}, \tag{15}$$

where $\mathbb{E}[h_j^{l-1}]$ and $\mathbb{V}[h_j^{l-1}]$ are the mean and variance of the j-th element of the window and N is the number of elements in the window. Finally, for Max pooling layers, the distribution with the largest mean is maintained.

D. Design flow

The design flow of the hardware implementation (see Fig. 2) begins with the BNN training, which is implemented in Python utilizing TensorFlow and TensorFlow Probability libraries. Then, the generated weights are exported to a file to load during the inference on the FPGA. In addition, a C program is developed to designate the flow of computations in the BNN inference. After that, the C code is converted into hardware using the Xilinx Vitis High-Level Synthesis (HLS) and Vitis Kernel Flow. Next, the produced bitstream file and the files containing the weights and input images are loaded onto the Zedboard FPGA. Finally, a C program on Zedboard's ARM processor launches the hardware kernel and collects the results.

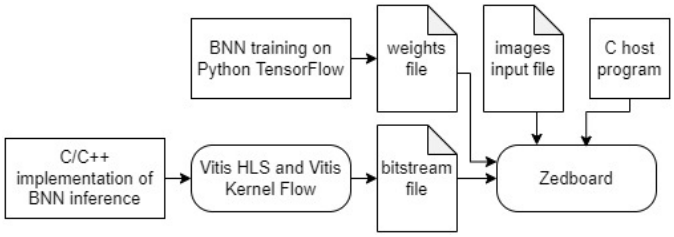


Fig. 2: Design flow of the BNN accelerator

E. Hardware architecture

In Vitis HLS, the hardware implementation is designed to build pipelines of computations in time-consuming operations, such as dot products and convolutions, to improve latency and throughput. Moreover, each layer is designed to work concurrently on a different input, leading to a continuous and efficient data flow. In addition, 17-bit fixed-point numbers have replaced 32-bit floating-point numbers, effectively reducing the area while degrading the accuracy by only up to 2% (see Fig. 3).

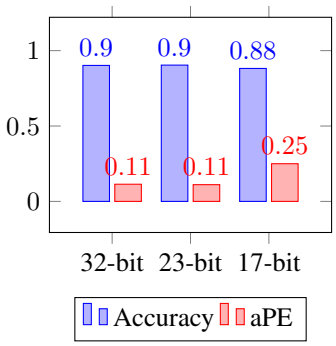


Fig. 3: Accuracy and aPE over different bit-widths for Fashion MNIST dataset on LeNet-5 with $f(x) = \frac{1}{2}(x^2 + x)$.

The architecture for a convolutional or a fully-connected layer is separated into two modules (see Fig. 4). The multiplication and accumulation module is responsible for either the dot products included in (5) and (6) or the respective convolution operations. A memory controller defines which input and weights are read and which output is updated. After the module with pre-activation computations finishes, the computation of moment propagation through the desired activation function is performed.

IV. Experiments and Results

The selected Xilinx Zedboard evaluation board includes an ARM Cortex A9 @667 MHz Processing System and Programmable Logic with 220 DSP slices, 106400 Flip-Flops, 53200 LUTs, and 560KB BRAMs. The hardware clock of the designs was set to 100 MHz.

Two NNs were selected for the experiments, the 3-layer fully connected neural network used in [5] and [6] (referred to as $FC3L\ net$ in this work) , and LeNet-5 [9], which consists of 3 convolutional and 2 fully-connected layers. Each hidden

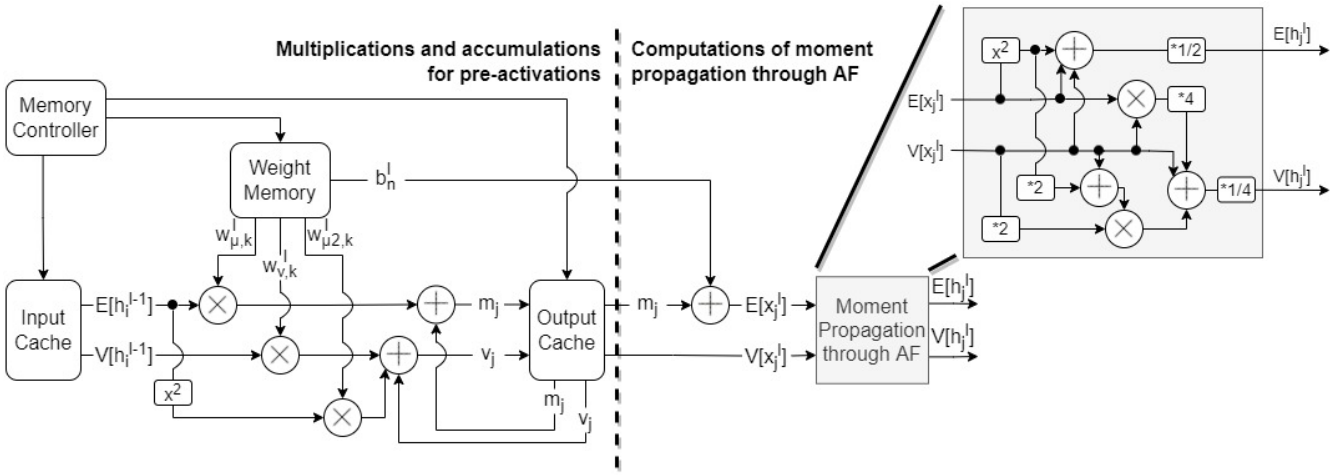


Fig. 4: Indicative architecture schematic for a convolutional or a fully-connected layer. In this figure, the polynomial approximation $f(x) = \frac{1}{2}(x^2 + x)$ of the ReLU activation function is shown.

layer in both NNs is fused with a Batch Normalization (BN) layer. The above NNs were tested on MNIST, Fashion MNIST [12], and SVHN [13] datasets.

A. Evaluation of models with polynomial activation functions

Table I contains the accuracy in the training and test set, and the uncertainty in the test set and in a random noise set, for different combinations of BNN, dataset, and activation functions. It can be seen that the accuracy achieved by $f(x) = \frac{1}{2}(x^2 + x)$ activation function is up to 5.9% higher than $f(x) = x^2$. Moreover, the accuracy of the polynomial approximation of tanh ($x - \frac{1}{3}x^3$) is the same with the accuracy of $f(x) = x^2$.

TABLE I: Accuracy and Uncertainty evaluation

BNN	Dataset	AF	Acc. Train↑	Acc. Test↑	aPE test↓	aPE rand↑
FC3L net	MNIST	x^2	0.9939	0.9821	0.017	0.000
		$\frac{1}{2}(x^2+x)$	0.9931	0.9818	0.019	0.000
	F_MNIST	x^2	0.8853	0.8354	0.170	0.001
		$\frac{1}{2}(x^2+x)$	0.8863	0.8451	0.162	0.013
LeNet-5	MNIST	x^2	0.9960	0.9930	0.009	0.843
		$\frac{1}{2}(x^2+x)$	0.9961	0.9935	0.008	0.438
		$x - \frac{1}{3}x^3$	0.9946	0.9930	0.010	0.465
	F_MNIST	x^2	0.9142	0.8958	0.127	0.701
		$\frac{1}{2}(x^2+x)$	0.9215	0.9036	0.113	0.095
		$x - \frac{1}{3}x^3$	0.9166	0.8941	0.134	0.447
	SVHN	x^2	0.8664	0.8105	0.016	0.790
		$\frac{1}{2}(x^2+x)$	0.8998	0.8693	0.017	0.806
		$x - \frac{1}{3}x^3$	0.8424	0.8126	0.229	0.657

B. Power, Performance and Area of the hardware designs

The hardware implementations are designed to achieve the same throughput regardless of the activation function, resulting in more hardware components for moment propagation through more computational heavy activation functions. However, as shown in Table II, despite the increased DSPs usage for more complex activation functions, the power overhead is negligible, as Xilinx Power Estimator (XPE) [14] reports.

TABLE II: Throughput, Area and Power of hardware designs

	FC3L net with AF $f(x) =$		LeNet-5 with AF $f(x) =$		
	x^2	$\frac{1}{2}(x^2+x)$	x^2	$\frac{1}{2}(x^2+x)$	$x - \frac{1}{3}x^3$
Throughput (Images/s)	635.45	635.45	80.22	80.22	80.23
DSPs	17 (8%)	31 (14%)	35 (16%)	59 (26%)	95 (43%)
Flip-Flops	14189 (13%)	16110 (15%)	27397 (26%)	30681 (28%)	37126 (34%)
LUTs	15980 (30%)	17071 (32%)	29449 (55%)	31193 (58%)	33513 (62%)
BRAM	13%	13%	41%	41%	41%
Power (W)	1.479	1.496	1.636	1.665	1.711

All the above measurements suggest that replacing $f(x) = x^2$ by $f(x) = \frac{1}{2}(x^2 + x)$ activation function is generally advantageous. Regarding the experiments, network designs implemented with $\frac{1}{2}(x^2+x)$ had an average accuracy boost of 1.5% compared to x^2. Furthermore, the average rise in power was only $0.024\ W$, with an acceptable area increase and no drop in throughput. Finally, focusing on the case of the SVHN dataset used in LeNet-5, merely $0.029\ W$ overhead has been required. At the same time, the design could achieve identical latency and an increase of 5.9% in accuracy.

V. Conclusion

This work has presented a low-power hardware implementation of sampling-free Bayesian inference for Convolutional Neural Networks, with high prospects in image recognition tasks. For an efficient hardware implementation, polynomial activation functions have been proposed for the replacement of $ReLU$ and $tanh$. The proposed $tanh$ polynomial approximation achieved accuracy similar to $f(x) = x^2$, while activation function $f(x) = \frac{1}{2}(x^2 + x)$ managed to surpass the accuracy of the previous suggestion of $f(x) = x^2$ for the replacement of $ReLU$ by up to 5.9% with negligible power overhead.

REFERENCES

[1] D. Garyfallou, A. Vagenas, C. Antoniadis, Y. Massoud, and G. Stamoulis, "Leveraging Machine Learning for Gate-level Timing Estimation Using Current Source Models and Effective Capacitance," ACM Great Lakes Symposium on VLSI, 2022.

[2] K. Fanaras, A. Tragoudaras, C. Antoniadis, and Y. Massoud, "Audio-visual Speaker Diarization: Improved Voice Activity Detection with CNN based Feature Extraction," IEEE International Midwest Symposium on Circuits and Systems, 2022.

[3] E. Kocić, N. Jovičić, and V. Drndarević, "An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms", MDPI Sensors, vol. 19(9), 2019.

[4] S. Namasudra, S. Dhamodharavadhani, and R. Rathipriya, "Nonlinear neural network based forecasting model for predicting covid-19 cases", Springer Neural Processing Letters, 2021.

[5] R. Cai, A. Ren, N. Liu, C. Ding, L. Wang, X. Qian, M. Pedram, and Y. Wang, "VIBNN: Hardware Acceleration of Bayesian Neural Networks", ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2018.

[6] H. Awano and M. Hashimoto, "BYNQNet: Bayesian Neural Network with Quadratic Activations for Sampling-Free Uncertainty Estimation on FPGA", IEEE Design, Automation and Test in Europe Conference, 2020

[7] H. Fan, M. Ferianc, M. Rodrigues, H. Zhou, X. Niu, and W. Luk, "High-Performance FPGA-based Accelerator for Bayesian Neural Networks", ACM/IEEE Design Automation Conference, 2021

[8] Y. LeCun, C. Cortes, and C. J. Burges, "MNIST handwritten digit database", IEEE Signal Processing Magazine, vol. 29(6), pp. 141–142, 2012.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", Proceeding of the IEEE, 1998

[10] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun, "Hands-On Bayesian Neural Networks — A Tutorial for Deep Learning Users", IEEE Computational Intelligence Magazine, vol. 17(2), pp. 29–48, 2022.

[11] R. E. Ali, J. So, and A. S. Avestimehr, "On Polynomial Approximations for Privacy-Preserving and Verifiable ReLU Networks", ArXiv, 2020.

[12] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms", ArXiv, 2017.

[13] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning", NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

[14] Xilinx Power Estimator (XPE), https://www.xilinx.com/products/technology/power/xpe.html