A Study on Hardware-Aware Training Techniques for Feedforward Artificial Neural Networks

Sajjad Parvin and Mustafa Altun
Department of Electronics Engineering
Istanbul Technical University
parvins17, altunmus@itu.edu.tr

***Abstract*—This paper presents hardware-aware training techniques for efficient hardware implementation of feedforward artificial neural networks (ANNs). Firstly, an investigation is done on the effect of the weight initialization on the hardware implementation of the trained ANN on a chip. We show that our unorthodox initialization technique can result in better area efficiency in comparison to the state-of-art weight initialization techniques. Secondly, we propose training based on large floating-point values. This means the training algorithm at the end finds a weight-set consisting of integer numbers by just ceiling/flooring of the large floating-point values. Thirdly, the large floating-point training algorithm is integrated with a weight and bias value approximation module to approximate a weight-set while optimizing an ANN for accuracy, to find an efficient weight-set for hardware realization. This integrated module at the end of training generates a weight-set that has a minimum hardware cost for that specific initialized weight-set. All the introduced algorithms are included in our toolbox called ZAAL. Then, the trained ANNs are realized on hardware under constant multiplication design using parallel and time-multiplexed architectures using TSMC $40nm$ technology in Cadence.**

***Index Terms*—artificial neural networks, hardware-aware training, parallel and time-multiplexed architecture, weightset approximation**

I. Introduction

Artificial neural networks (ANNs) have been successfully implemented on various design platforms, such as, analog, very large scale integrated circuits (VLSI), etc [1]. ANN is constructed from basic blocks called neuron, shown in Fig. 1 (a). The behavior of each neuron can be defined mathematically as $y = \sum_{i=1}^{n} w_i x_i$ and $z = \phi(y+b)$ where n denotes the number of input signals, weights, and ϕ denotes the activation function on the summation. By connecting several cascaded neurons together we can form an ANN, shown in 1 (b), where each circle denotes a neuron.

From Fig. 1, we can conclude, the hardware implementation of ANN is severely dependent on the weight and bias values and it is heavily influenced by a large number of multiplication of constant weights by input signals. In recent years, many studies have been investigating approaches for efficient implementation of ANN on chip, devising both new training algorithms and novel hardware architectures [2]–[9]. In this paper, we solely investigate training algorithms and techniques that result in more efficient hardware implementation. We utilize two hardware architectures to implement our ANNs as a

This work is supported by the TUBITAK-1001 project #119E507.

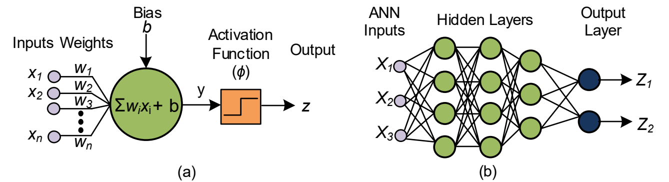

Fig. 1: (a) Artificial neuron; (b) ANN with two hidden layers.

proof of concept for our proposed algorithms; constant multiplication design of parallel and time-multiplexed architectures. These two architectures are implemented in a behavioral fashion and then synthesized and optimized in the Cadence Genus tool. Moreover, there is a trade-off between area, latency, and consumed energy for each of these architectures. Hence, based on the size of each ANN, we choose one of these architectures for the hardware implementation. But the main contribution of this paper is the hardware-aware training algorithms.

As in conventional ANN training algorithms, the trained weight and bias values are high precision small floating-point numbers. Since floating-point multiplication hardware implementation occupies a large area and consumes a great amount of energy, ANN's weight-set must be converted to integer with a quantization factor [10] for hardware implementation. In this work, we explore a pre-quantized weightset based training where the network is trained based on the large floating-point values. Unlike conventional approaches for hardware implementation where the weightset is quantized after training, in this approach, the quantization value is set before starting the training. Eventually, we apply a smart ceiling/flooring approach to convert the trained weightset to an integer weightset with having minimum hardware cost. Furthermore, we propose a hardware-aware training algorithm for considering the hardware cost during training phase and try to train an ANN with minimum hardware cost while optimizing the network for accuracy. Plus, we investigated the effect of weight initialization on hardware implementation as well. The experimental results are in favor of our proposed techniques in terms of hardware implementation.

This paper is organized as follow. Section II discusses the preliminaries and related works. Our hardware aware-training and ZAAL toolbox are discussed in Section III. The experimental results are discussed in Section IV. Finally, Section V concludes this paper.

II. Preliminaries And Background

A. ANN Basics

In this work we use feedforward neural networks for hardware implementation for their simple structure. For a given ANN configuration, the number of inputs, outputs, hidden layers, and activation function of each layer is determined and the network is trained using an iterative optimization algorithm to minimize the difference between desired output values and computed output values. After training, the weight and bias values of the ANN are determined. State-of-art ANN toolboxes [9], [11], [12] include various initialization techniques, optimizers, stopping criteria, and various choices of activation functions for each layer of the ANN. ANN training is done on processors and/or GPUs. In the testing procedure, the trained weight and bias values of the model are used to compute the response of the ANN upon unseen data. The testing phase of the ANN is computed on a hardware design platform such as application-specific integrated circuit (ASIC) and FPGAs, for on-field operation.

B. Cost Metric for Hardware During Training

For our hardware-aware training algorithms, we use the CSD[1] representation as a hardware cost metric during the training of a network. This means an ANN with a weightset consisting of less number of non-zero digits in their CSD representation results in a smaller area on chip. This is because while realized in the synthesis tool, the constant multiplications can be divided into sub-expressions and then the constant multiplication can be implemented using additions/subtraction, and shifts. The hardware complexity of an ANN is heavily influenced by how large the weight and bias values of the ANN are. Hence, CSD is a suitable metric for hardware before implementation. CSD representation gives a sense of how large an ANN can be on hardware without even realizing it on hardware under constant multiplication design.

C. Related Works

Multiplierless design has been extensively studied in the literature, to eliminate the need for conventional multipliers which require a huge area on chip in the design of ANNs. For example in [2], [3], binary neural networks (BNNs) are studied where the weights, biases, and activation functions are all constrained to -1 or +1. It is shown that BNNs reduces the memory size and number of access time to memory while training. Plus hardware implementation of BNNs are simple and efficient; since all the multipliers in BNNs are substituted with XOR operators. Since all the weights and output of each neuron are limited to -1 or +1, the accuracy of BNNs are worse than the conventional ANNs [4]. In [4], [7], while training the network, the weights are determined to have less number of nonzero digits. Hence, this resulted in the multiplication operation of each neuron to be implemented with fewer adders and subtractors. Moreover, training algorithm of [4] just quantizes the weights for the forward path to calculate the output but the backpropagation is done on the dequantized values of a weightset. In [5], the floating point weights of the network are mapped to 8-bit dynamic fixed-point with integer power-of-two weight. This allows the network to be implemented more efficiently. Moreover, since the multiplication of weights and input signals is the major bottleneck of ANN implementation multiplierless design of ANN using bit-serial fashion multiplication is proposed in [8].

[1]An integer can be written in CSD using n digits as $\sum_{i=0}^{n-1} d_i 2^i$, where $d_i \in \{-1, 0, 1\}$. The nonzero digits must not be adjacent and a constant is represented with a minimum number of nonzero digits under CSD.

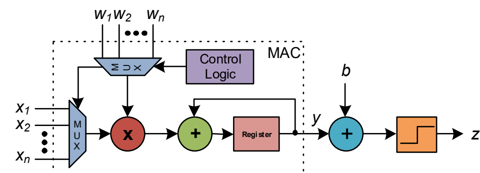

Fig. 2: Neuron Computation of a time-multiplexed architecture using multiply-accumulate (MAC) unit.

D. Design Architecture

In this work, we implement constant multiplication based realization of parallel and time-multiplexed architecture. Basically, in parallel design architecture, when the input for the ANN is applied, neurons in each layer compute the result concurrently until the results are propagated to the output neurons. Parallel architecture results in smaller delay but larger area overhead. On the other hand, for larger ANNs, we utilize time-multiplexed architecture, illustrated in Fig. 2, where it uses its resources multiple time. This results in having smaller area overhead but worse delay than parallel architecture. In this study, for pen-digit recognition dataset [13] which is small, parallel architecture is used and for MNIST dataset [14], due to its larger size, time-multiplexed architecture is used.

III. Hardware-Aware Training

In this section, we will discuss our hardware-aware training approaches for training an ANN to be efficiently implemented on hardware. Firstly, we will discuss the effect of the weight initialization on the hardware implementation cost of the trained ANN and propose a simple uncanonical weight initialization technique. Secondly, we discuss a pre-quantized weightset training (PQ) approach to train an ANN using large floating-point values. This approach allows us to have integer weight and bias values after the training is done with just ceiling/flooring ANN's weightset. Thirdly, we take our PQ training approach and integrate the training algorithm with modules to both reduce the size of weight and bias values during training and also to track and choose the final cost of the hardware implementation.

A. Weight Initialization Effect on Hardware Cost

Many initialization techniques have been proposed in the literature for ANNs depending on what type of activation function hidden layers utilize [15], [16]. These initialization techniques are proposed to prevent the training algorithm from experiencing weight explosion or weight vanishment. Because, if the weights are initialized too large or too small

the network in deep network structures will suffer from the weight explosion and vanishment. This means that the training algorithm will not be able to find good minimas due to the bad weight initialization. However, these initialization techniques take the assumption that for a certain network in each training iteration, the weightset of each layer will be having a variance of 1 and the mean of 0 [15]. In reality, during the training, the variance of each layer will not be 1 and it deviates far from the value of 1, especially for deep structures. This assumption is just valid on paper.

Moreover, we observed that the initialization of weightset can have a major effect on the hardware implementation of the network. This means when the weight and bias values are initialized from a distribution with a smaller standard deviation, the trained weightset will be smaller in comparison to the time when ANN is initialized from a distribution with a larger standard deviation. Hence, less non-zero digit in the CSD representation of the weightset values after quantizing the floating point weightset values.

To answer the question of how small we can choose the initialization standard deviation of distribution for initializing our weightset, we apply a greedy search starting from the Xavier or/He method. Then, we search for the standard deviation ranges for random initialization of weightset that results in the least hardware cost while keeping the accuracy the same as Xavier [15] or He [16] initialization methods or even better. In this process, we will be initializing the ANN from a Gaussian distribution and we will be shrinking the standard deviation after each run of the ANN training. We keep the standard deviation of all the ANN's layer the same for the sake of simplicity. We can continue this process until we find a standard deviation range that results in weight vanishment or until we have enough resources to run the search.

B. Training with pre-quantized weightset

We train the network with pre-quantized weightset. Hence, the quantization value and the output bitwidth are determined before starting the training algorithm. This means that the activation function bounds are set to a pre-determined quantization value that we have in our hardware. As an example, if our hardware is capable of 8 bits computations at the output of each neuron, we set the boundaries of activation function to 8 bits, which includes the numbers between [-256, 256] (with the assumption that the activation functions are the linearized version of $sigmoid$ and $tanh$ with boundaries constraint to 2^q where q is the predefined quantization value. Moreover, the training is done using the conventional algorithms in the literature such as Adam [17], stochastic gradient descent, etc. Also we need to multiply the initialized weightset with the same pre-defined quantiization value. Otherwise, if we initialize the network's weightset for training with the numbers between [-1, +1], the training will not converge to a good minima. This is because during the training process, all the weights will be operating around the linear region of the activation function. We can say that no non-linearity exists from the input side of the ANN to the output of the ANN. Moreover, it should be mentioned that if the initialized weights are multiplied by any factor less than the quantization factor "q" for the output bitwidth ($W_i \times 2^q$), then it takes longer for the ANN to converge to a good minima. This is because, during the backpropagation, it takes longer for the weights to grow and operate on the whole region of the each neuron's activation function.

1) Smart Ceiling/Flooring the Trained Network Weight Set: To convert all the weightset values to integer, we need to ceil/floor the weightset. Since, our ANNs is going to be implemented in constant multiplication fashion according to Section II, we need weight and bias values with less non-zero digit in their CSD representation. Hence, we go through all the weight and bias values, and check which operation, ceiling/flooring results in less non-zero digit in CSD representation of each weight and bias values of the weightset. It is worth to mention if the pre-defined quantization value is chosen large enough, a fractional loss in accuracy is cause after ceiling/flooring the weightset.

C. Hardware-Aware Training

Our hardware-aware training basically trains the ANN based on the training technique discussed in Section III-B, but the difference is, while performing the regular back-propagation and forward-propagation to maximize the ANN's accuracy, we perform an approximation on the weight and bias values of the ANN. The approximation technique on the weight and bias values essentially removes the least significant non-zero digit in the CSD representation of each weight or/bias value in the ANN with a tolerable range of loss in the accuracy of the ANN on validation dataset. After applying this approximation on the CSD representation of each weight and bias value, the algorithm resumes performing back-propagation and forward-propagation to find better local minimas. Actually this approximation introduces a new form of stochastic search into the optimizer's algorithm, which results in better accuracy in software and better hardware cost (fewer non-zero digit in the CSD representation of the model's weightset). For large ANNs, approximating all the weights is not efficient, hence, for large networks instead of going through all the weights, we approximate all the weights that are connected to a neuron. According to [18], a network is less sensitive to a change in the whole neuron rather than a change in each and every single weight in the network. Hence, this approach is used for approximating weights during training of a large networks. It must be concluded, since we do this approximation of weights during the training using integers, this approximation must be done in every p iterations of the optimization algorithm; here we use the Adam optimizer [17]. If we do it in each and every iteration of the optimizer, since the rate of changes in the weight and bias values are relatively small, after ceiling/flooring of the weight and bias values, the optimizer stucks at a point in the search space and cannot proceed further to find a better minima. During the training, while performing this approximation on the weight and bias values, we set the network to have a tolerance on the

accuracy loss while performing this approximation (we choose $z = 1\%$ loss in the accuracy). This allows the ANN to find better models with better accuracy and hardware cost, after the training is done.

Moreover, while training we also keep track of the accuracy and hardware cost. This allows us at the end of training to have a list of weightset that have close accuracy result but different hardware cost. In this approach, while training, at each q iterations, we save the model's accuracy and weightset. And if in the upcoming iterations of the optimizer algorithm, we find a model with better accuracy more than the predefined tolerable accuracy drop (we use $z = 1\%$ tolerable accuracy drop), we save the current network model. And at the end, we choose the model with the best hardware cost, for hardware implementation.

D. ZAAL: The Hardware-Aware Training Tool for ANN

We present our ANN training tool called ZAAL developed to include all the hardware-aware algorithms mentioned in this work. For a given dataset, ZAAL can train the network both with conventional float-point based numbers (ZAAL FP) and also it can train the network with pre-quantized weightset ZAAL PQ as discussed in Section III-B, and pre-quantized weightset training with approximation (ZAAL PQAX) discussed in III-C . ZAAL's training optimizers include Adam [17], stochastic gradient descent and conventional gradient descent algorithm. Also, it includes various initialization techniques such as Xavier [15], He [16] and random weight initialization method. Our tool includes several optimization stopping criterias, e.g., number of optimizer's iterations, early stopping using validation data set, and saturation of loss function. Also, it includes several activation functions for each layer of the network, namely, sigmoid, hard sigmoid (hsig), hyperbolic tangent, hard hyperbolic tangent(htanh), linear (lin), rectified linear unit (ReLU), saturating linear (satlin), and softmax [19].

IV. EXPERIMENTAL RESULTS

To show the performance of our hardware-aware techniques for training of an ANN, we used the pen-based hand-digit recognition dataset [13] and the MNIST dataset [14]. We used different network structures with different number of hidden layers and different number of neurons in each hidden layer. We show the network structure as follow; $p_{in}/p_{h_1}/p_{h_2} - ... - p_{h_n}/p_{out}$, where the p_{in} and p_{out} denote the number of input and output signals for the network, respectively, and p_{h_j} where $1 \leq j \leq n$ shows the number of neurons in j^{th} hidden layer of the network. These ANNs are trained using our hardware-aware algorithms, and the results are compared with other ANN tools which are not targeting hardware such as PYTORCH and MATLAB [12] toolboxes. To keep everything consistent with our techniques, we used the $hardtanh(htanh)$ for hidden layers and $hardsigmoid(hsig)$ for output layer as the activation functions for ANNs trained model using PYTORCH and ZAAL and for MATLAB ANN toolbox, we used $satlins$ for hidden layers and $satlin$ at the output layer.

The weightset is initialized using Xavier initialization [15] for PYTORCH and MATLAB due to having $htanh$ and $satlins$ as activation function. Then due to existence of randomness in the training process, the network is trained 30 times, with various initialization points. Then the network is trained using Adam optimizer [17]. The optimizer is stopped once the accuracy on the validation set started to deviate from the training set's accuracy. And after the 30 rounds of training, we chose the trained parameters that resulted in the best software accuracy for hardware realization. It must be noted that for floating-based training the weights were quantized with minimum quantization value [9] and for our pre-quantized weightset training algorithm discussed in Section III-B, the weightset is converted to integers using the smart ceiling/flooring discussed in Section III-B1. In our training approaches, we use the same quantization value as found using MATLAB and PYTORCH. The quantization value found by toolbox in [9] for the weightset generated using MATLAB and PYTORCH for the pen-digit hand-based dataset and the MNIST dataset is 7 and 10, respectively.

For our training approaches we used the initialization technique discussed in Section 3. To investigate the effect of initialization on the hardware implementation, we used PYTORCH and ZAAL FP training with floating-point values (conventional training with small floating-point values) to show the effect of the initialization. We needed to use a large network with a large number of parameters to see the effect of shrinking distribution's standard deviation on the hardware realization. Hence, we used the MNIST dataset [14] where it has 784 input feature. Then, we have trained each the model for 30 run but the random weight and bias values were chosen from a smaller distribution with standard deviation rather than the Xavier initialization [15]. And after the model was trained after 30 run, we chose the model with best accuracy among all 30 runs of training and implemented the hardware using toolbox in [9]. And we ran the tool to find the minimum quantization value for each network to be implemented under time-multiplexed architecture. Fig. 3 shows that the smaller the standard deviation of initialized network is the smaller area of realized ANN on chip will be, in comparison to the conventional initialization technique used in the literature, in this case Xavier technique [15]. The smallest standard deviation yielded 119% better area using ZAAL and 19% less area occupation using PYTORCH. This simple approach can save good amount of area on hardware. This is because when the distribution of weights become smaller during the process of training the quantized weight and bias values become smaller. Yet, this initialization approach is random and might not always result in huge improvement on the occupied area (Compare ZAAL and PYTORCH area from Fig. 3; area(μm^2), latency (ns), and energy (pJ)). It must be noted that we trained the networks to prevent from weight vaninshment and explosion. This means the all the results in Fig. 3 were set to have the accuracy of 96% using the *784-256-256-10* architecture. In this experiment, all the layers are set to have the same standard deviation for the sake of simplicity.

Moreover, we implemented our proposed training algo-

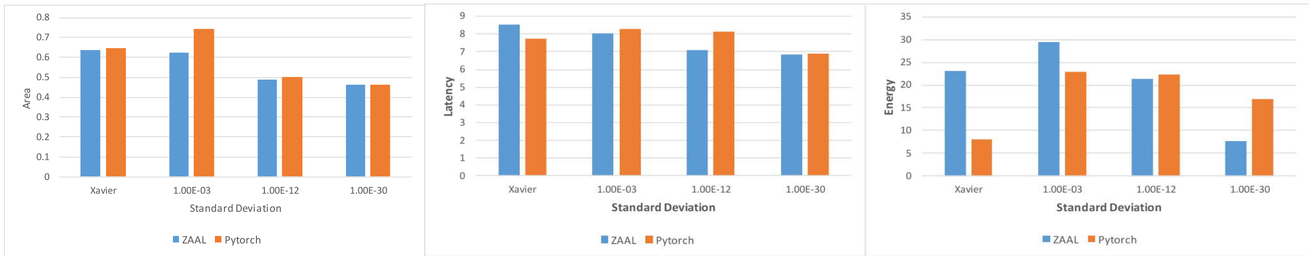

Fig. 3: A performance comparison on the effect of initialization of ANN weightset on hardware of the trained ANN.

rithms discussed in Section III on hardware. Table I and Table II compare the results of the accuracy and hardware of each training algorithm for the pen-digit recognition dataset [13] and the MNIST [14] dateset, respectively. In these tables, *sta*, *hta*, and *tnzd* indicate software accuracy, hardware accuracy, and total non-zero digits in the CSD representation of integer weight and bias values of the trained network, respectively. Table I and Table II, shows that different training algorithms results in different hardware complexity but it yields quite similar hardware accuracy. For example, according to Table I, our proposed hardware-aware training algorithm with pre-quantized weighset (ZAAL PQ) and pre-quantized weightset training with approximation (ZAAL PQAP) outperformed the conventional floating point training approaches (MATLAB and PYTORCH) by very good amount in terms of having less *tnzd*. Our proposed training algorithms ZAAL PQ using pen-digit recognition dataset, on average resulted in 19% and 30% less *tnzd* in comparison to MATLAB and PYTORCH, respectively. Beside ZAAL PQAP using pen-digit recognition dataset, on average resulted in 42% and 39% less *tnzd* in comparison to MATLAB and PYTORCH, respectively. According to Table II, using the MNIST dataset [14], on average ZAAL PQ outperformed the MATLAB and PYTORCH by 537 % 526% in terms of less $tnzd$, respectively. Again, according to Table II, using the MNIST dataset [14], on average ZAAL PQAP outperformed the MATLAB and PYTORCH by 606 % 594% in terms of less $tnzd$, respectively.

After the networks are trained, we implemented all the networks in constant multiplication fashion under parallel and time-multiplexed architectures as discussed in Section II. Hardware description of our networks is generated using toolbox discussed in [9] and all the design are synthesized in Cadence Genus tool using TSMC 40nm technology. To show the effectiveness of our algorithms on hardware implementation, we calculate the values for area (in μm^2), latency in (ns), and energy consumption in (pJ).

As illustrated in Fig. 4 and Fig. 5, our training algorithms resulted in very good hardware implementation in comparison to the conventional floating point based training. For pendigit dataset, we used parallel architecture due to its small number of ANNs parameters. On average, hardware implementation of ZAAL PQ achieved 20% and 25% less area occupation in comparison to PYTORCH and MATLAB, respectively. On average energy consumption of ZAAL PQ using pen-digit dataset is 75% and 78% less than the MATLAB and PYTORCH, respectively.

Moreover, our proposed ZAAL PQAP on pen-digit recognition dataset on average resulted in 40% and 43% less area occupation on chip in comparison to the the MATLAB and PYTORCH, respectively. Plus, the ZAAL PQAP achieved 56% and 50% less energy consumption in comparison to the the MATLAB and PYTORCH, respectively.

Furthermore, we tested our proposed techniques on the MNIST dataset [14]. As shown in Fig. 5, both of our proposed approaches outperformed the conventional floating-point based ANN training drastically. For the MNIST dataset [14], we utilized the time-multiplexed architecture due to ANN's large size. Otherwise, using the parallel architecture would have resulted in huge area on chip. MATLAB and PYTORCH resulted in 4.18 and 3.34 times larger area in comparison to our ZAAL PQ training algorithm, respectively. And also the energy consumption of MATLAB and PYTORCH is 17.7 and 8.7 times larger than the ZAAL PQ, respectively. Beside, for our proposed ZAAL PQAP, on the MNIST dataset [14], the overall performance on hardware had 10% improvement in comparison to the ZAAL PQ.

The reason our PQ based training algorithm achieved this much improvement is that the majority of trained ANN's weight and bias values become zero. On the other hand, in floating-point based training, most weight and bias values are small and after quantizing them, still they will not be zero. Hence, they will add up to the cost of the ANN. When a weight becomes zero, it means that synapse can be remove from the ANN. As a result, smaller area is achieved on hardware.

V. CONCLUSION

In this work, we investigated hardware-aware training algorithms for training ANNs. Our training algorithm is based on training an ANN using a pre-quantized weightset. Unlike conventional floating-point based training, the quantization value is fixed before training. Besides, we also minimize hardware cost during training. Above all that, we showed our unorthodox approach to weight initialization can have a descent impact on final hardware implementation cost.

REFERENCES

[1] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, 2010.

[2] M. Courbariaux and et. al, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *ICNIPS*, 2015.

TABLE I: A design comparison among various ANN toolboxes using pen-digit dataset [13].

Structure	ZAAL PQ			ZAAL PQAP			MATLAB			PYTORCH		
	swa	hwa	tnzd	swa	hwa	tnzd	swa	hwa	tnzd	swa	hwa	tnzd
16/10	88.3	87.5	407	87.1	87.1	294	89.1	89.3	374	85.5	85.1	374
16/10/10	94.6	94.5	667	95.2	95.7	474	95.9	95.9	857	95.9	95.2	950
16/16/10	94.4	94.2	1004	96.5	96.4	698	96.9	95.0	1291	95.6	95.6	1338
16/10/10/10	95.6	95.5	897	96.1	96.2	904	96.4	94.7	1121	95.8	95.6	1190
16/16/10/10	96.5	96.4	1218	96.6	96.5	797	96.6	95.2	1560	96.7	96.7	1608
Average	93.9	93.6	839	94.3	94.4	633	95.0	94.0	1041	93.9	93.6	1092

TABLE II: A design comparison among various ANN toolboxes using the MNIST dataset [14].

Structure	ZAAL PQ			ZAAL PQAP			MATLAB			PYTORCH		
	swa	hwa	tnzd	swa	hwa	tnzd	swa	hwa	tnzd	swa	hwa	tnzd
784/10	91.1	90.0	13839	91.0	89.8	12432	91.9	91.1	29285	91.8	90.7	15277
784/128/10	94.8	94.5	67442	94.6	94.2	62721	98.0	97.8	262938	96.4	95.8	252113
784/256/256/10	96.5	95.8	79987	96.7	95.8	70317	98.2	96.1	735221	96.8	96.1	742584
Average	94.1	93.4	53756	94.1	93.2	48490	96.0	95.0	342481	95.0	94.2	336719

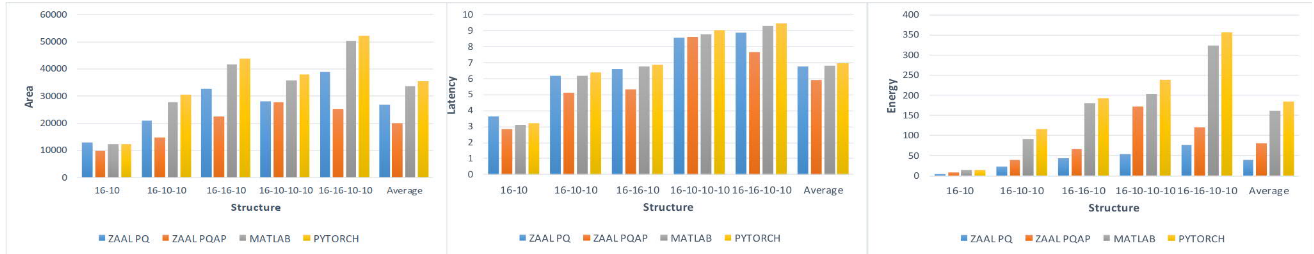

Fig. 4: Hardware results comparison of various ANN toolboxes using parallel architecture design for the pen-digit dataset [13].

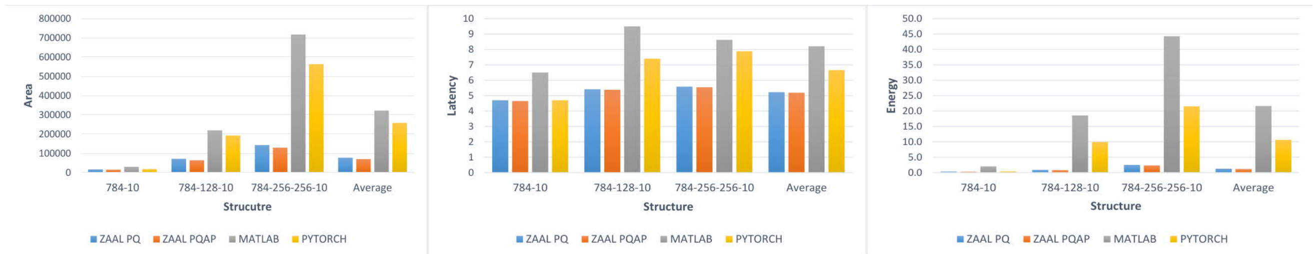

Fig. 5: Hardware results comparison of various ANN toolboxes using time-multiplexed arhitecture design for the MNIST dataset [14].

[3] 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 e-prints*, 2016.

[4] R. Ding, Z. Liu, R. D. Blanton, and D. Marculescu, "Quantized deep neural networks for energy efficient hardware-based inference," in *Asia and South Pacific Design Automation Conference*, 2018, pp. 1–8.

[5] H. Tann and et. al, "Hardware-software codesign of accurate, multiplier-free deep neural networks," in *DAC*, 2017.

[6] H. Park and T. Kim, "Structure optimizations of neuromorphic computing architectures for deep neural network," in *DATE 2018*, pp. 183–188.

[7] S. S. Sarwar and et. al, "Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing," in *DATE 2016*.

[8] T. Szabo, L. Antoni, G. Horvath, and B. Feher, "A full-parallel digital implementation for pre-trained NNs," in *IJCNN*, 2000, pp. 49–54.

[9] L. Aksoy, S. Parvin, M. E. Nojehdeh, and M. Altun, "Efficient time-multiplexed realization of feedforward artificial neural networks," in *ISCAS*, 2020.

[10] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE ISSCC*, 2014.

[11] A. Paszke and et. al, "Automatic differentiation in pytorch," in *Conference NIPS, Autodiff Workshop*, 2017.

[12] The MathWorks Inc., *Deep Learning Toolbox*, Natick, Massachusetts, United States, 2020.

[13] F. Alimoglu and E. Alpaydin, "Combining multiple representations and classifiers for pen-based handwritten digit recognition," in *ICDAR*, 1997.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *ICAIS*, 2010, pp. 249–256.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *arXiv e-prints*, 2015, arXiv:1502.01852.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv e-prints*, 2014, arXiv:1412.6980.

[18] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: Energy-efficient neuromorphic systems using approximate computing," ser. ISLPED, 2014.

[19] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv e-prints*, 2018.