Robust Binary Neural Network against Noisy Analog Computation

Zong-Han Lee
Department of Computer Science
National Tsing-Hua University
Hsinchu, Taiwan
zonghan.l@gapp.nthu.edu.tw

Fu-Cheng Tsai
Electronic and Optoelectronic System Research Laboratories
Industrial Technology Research Institute
Hsinchu, Taiwan
itriA70513@itri.org.tw

Shih-Chieh Chang
Department of Computer Science
National Tsing-Hua University
Hsinchu, Taiwan
scchang@cs.nthu.edu.tw

***Abstract*—Computing in memory (CIM) technology has shown promising results in reducing the energy consumption of a battery-powered device. On the other hand, to reduce MAC operations, Binary neural networks (BNN) show the potential to catch up with a full-precision model. This paper proposes a robust BNN model applied to the CIM framework, which can tolerate analog noises. These analog noises caused by various variations, such as process variation, can lead to low inference accuracy. We first observe that the traditional batch normalization can cause a BNN model to be susceptible to analog noise. We then propose a new approach to replace the batch normalization while maintaining the advantages. Secondly, in BNN, since noises can be removed when inputs are zeros during the multiplication and accumulation (MAC) operation, we also propose novel methods to increase the number of zeros in a convolution output. We apply our new BNN model in the keyword spotting application. Our results are very exciting.**

***Index Terms*—Deep neural networks, Analog AI, Noise tolerance**

I. Introduction

The breakthroughs in deep neural networks (DNN) have achieved significant improvement in various fields, such as image classification [1], [2], speech recognition [3], [4] and natural language processing [5], [6]. However, due to a large number of MAC operations, DNN requires high memory storage and computing energy. Large energy consumption has huge impact on the use of AI in mobile devices.

In order to overcome the energy scarcity barrier, one of the candidate hardware methods is computing in memory (CIM) technology [7], [8]. It integrates computing and storage units and uses analog computing to accelerate MAC operations. By minimizing data movement and representing large matrix multiplications as analog computation, CIM has significantly improved energy efficiency, throughput, and computing density. Conversely, from the DNN perspective, another approach to reducing energy consumption is to quantize parameters and activations from floating-point into low-precision fixed points [9]. With quantization, we can reduce MAC operations and storage overheads, leading to better throughput and energy efficiency.

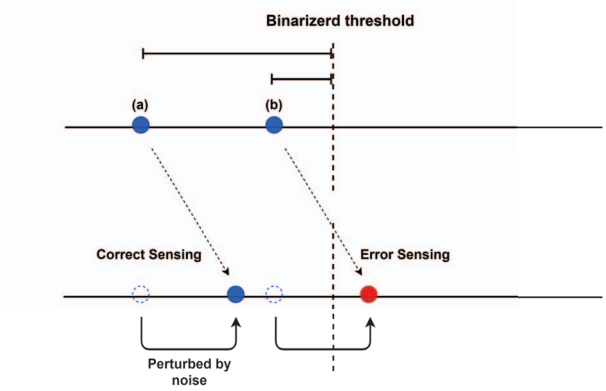

Fig. 1. The Contrast of two conditions under noise perturbation. Compared to value (a), the value (b), which has a narrow margin to the binarized threshold, has a higher probability of being wrongly flipped from 0 (1) to 1 (0) when the noise perturbs the value.

One of the most extreme quantization is binarization. Binary neural networks (BNN) [10], [11] constrain both the weights and the activations to either $+1$ or -1 (0) to reduce MAC operations and storage overheads, leading to better throughput and energy efficiency. In recent years, BNN has shown the potential to catch up with a full-precision model [12]. In addition, due to binarized 1-bit weights and activations that can easily be represented by electrical voltages or current, BNN is well-suited to deploy on the CIM architecture [13].

However, the analog MAC is very sensitive to various kinds of variations, such as process variation, especially in the advanced process. These variations are referred to as *analog computation noise* or *analog noise*. In CIM, analog noise can reduce the recognition accuracy of DNN. How to improve the robustness of a model is a very urgent issue for using CIM. To address this issue, many previous works leverage the well-known noise injection training to improve model robustness [14]–[16]. In these works, analog noise is modeled

as additive parameters and is injected during the training phase to make a model adapt to noise during the real inference phase.

A natural question that arises is what are the reasons that analog noise can cause a large accuracy drop in BNN. We observe that Batch normalization (Batchnorm) [17] can be one of the main reasons due to the following. First, Batchnorm redistributes the convolution output in modern networks to improve the convergence speed and performance. On the other hand, let us consider a convolution operation in BNN. A threshold value called the *binarized threshold* exists to determine whether the binarized value of the convolution result is either 0 or 1. We observe that Batchnorm will move the mean of the convolution output close to the binary threshold. In other words, after Batchnorm, many convolution results will be close to the binarized threshold. When there is an analog noise during CIM computation, a value close to the threshold can be easily flipped from 1 to 0 or 0 to 1. As shown in Fig. 1, point (a) is far from while point (b) is close to the binary threshold. As a result, point (a) can tolerate certain noise while point (b) cannot. The first contribution of this paper is to propose a new operation, the Batchnorm-free BNN. The alternate operation needs to maintain the advantage of Batchnorm but will not force the output value of convolution to be close to the threshold.

Another important observation is that, although analog noise is inevitable, the effect of analog noise can be "erased" by multiplying "zero." We propose a modified sign function and leverage max pooling [18] to increase the ratio of 0 in input activation to erase the analog noise in convolution computation.

To verify our solution indeed improves robustness, we evaluate our CIM design on the keyword spotting application. Note that CIM for keyword spotting has been used as commercial products [19]. The results show that our approach can be combined with the noise injection training method to achieve significant improvements.

II. Related Work

A. Improving performance under noisy computation

Many efforts are devoted to mitigating the accuracy degradation caused by analog noise. For algorithm-based approaches, [20] demonstrates the ability of common DNN components and regularization methods to improve noise tolerance. Several techniques based on noise-injection training have demonstrated significant improvement in analog computing robustness. [14], [15] inject an additive noise sampled from Gaussian distribution on either weight or matrix-vector multiplication in the forward path during the training phase. By simulating noise in the training phase, the trained weights are adjusted to deal with signal variation caused by the non-ideality of the analog circuit. Based on noise injection training, [16] further integrated with knowledge distillation which can leverage the teacher model's information to enhance noise tolerance. However, these techniques largely depend on prior knowledge of the noise on target devices; if the distribution of noise is different between training and inference, the effectiveness of this method will decrease.

B. Batch Normalization

Batchnorm is a widely used DNN training technique to stabilize and accelerate model training. The core idea of Batchnorm is reducing internal covariate shift by re-scaling and re-centering. However, Batchnorm also has disadvantages. It brings out the inconsistency between the training and validating phase and increases the resource overhead due to extra parameters and computation, which add to the difficulties in designing resource-constrained devices. According to our observation, Batchnorm also harms robustness to noisy analog computation during inference.

There are many works attempts to seek alternative approaches to Batchnorm. Many alternative normalizers [21] have been proposed to eliminate the drawback of Batchnorm. However, these methods are generally not able to exceed Batchnorm in test accuracy or demand too much extra computation. [22], [23] shows that weight standardization can potentially replace Batchnorm with high accuracy and low extra computation during the inference phase.

III. Preliminary

A. Binary Neural Network

A BNN represents a weight or an activation by only one bit. Our BNN settings constrain the binary activation to 1 or 0 while the binary weight to 1 or -1. In most BNN training schemes, the weight is stored as real value until the end of the training phase, and the optimizer will run through a straight-through-estimator to update these real value weights [24]. These weights will be binarized in the forward path according to their sign before the convolution operation [10].

In terms of activations, the sign function is used as a standard binary activation function :

$$z_i^b = Sign(z^r) = \begin{cases} +1, & if \quad z^r > 0 \\ 0, & if \quad z^r \leq 0 \end{cases} \tag{1}$$

We define the threshold which determines the binarized value of a convolution result as the binarized threshold. After the activation function, the convolution result larger (smaller) than the binarized threshold will be assigned to 1 (0).

B. Analog Noise Model

Due to the variation of quality of wafer manufacturing, stability of supply voltage, and temperature, the manufactured analog components have non-ideality, which means their behavior will deviate from the original ideal design.

Memory cell plays a critical role in CIM-based DNN accelerator to represent weights and its dot product [7]. As we mentioned above, memory cells also suffer from intrinsic noise, leading to error computation. In the DNN scenario, many previous works [15] assume the well-trained weights have variation during actual inference. To model the noise, we simulate NMOS cell based in-memory to complete convolution function in memory unit with low area and energy cost. A 28nm 128Kb CIM macro was simulated using NMOS cell based on a discharge scheme. This macro achieves clock cycle in 10ns at $V_{DD} = 0.9V$. According to the previous study [14] and our

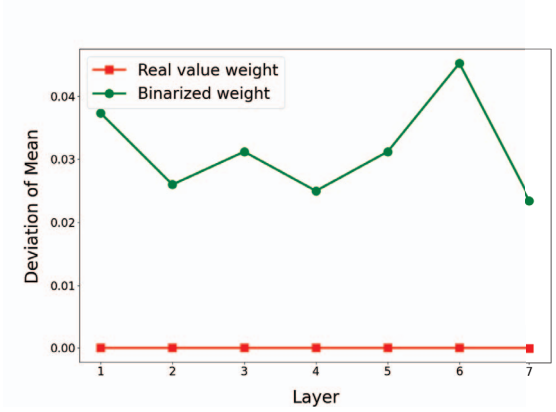

Fig. 2. Contrast the deviation of mean before/after binarization. This figure shows that after binarized, the mean of weights deviates from zero in every layer.

Monte Carlo simulation, we can assume noise on binarized weights are sampled from zero-mean Gaussian distribution :

$$W^b_{Noisy} = W^b_{Orig} + \eta, \quad \eta \sim \mathcal{N}(0, \sigma^2) \tag{2}$$

where a noise scale σ is used to control the severity of noise.

IV. Methodology

A. Batchnorm-free BNN

As we mentioned in Section I, we observe the Batchnorm amplifies the effect of noises. One property of Batchnorm to stabilize training is to eliminate internal covariate shift by centering the convolution output distribution according to its mean. After this operation, the binarized threshold is close to the mean of the convolution output distribution. Since convolution output distribution tends to be similar to the normal distribution, a large proportion of value gather around the binarized threshold, resulting in a generally narrow perceptual margin, i.e., the difference between the ideal (without noise) values and the binarized threshold (Fig. 1). It leads to a higher probability of being wrongly flipped under noisy situations.

So our idea is to design a Batchnorm-free scheme to tackle this issue. To mitigate the internal covariate shift [17] issue while removing Batchnorm, we leverage scaled weight standardization [22]. It is an alternative to stabilize training in general DNN. Unlike [25] focusing on binarized ResNet, we find it effective when applied to conventional BNN and enhanced robustness. Specifically, we transform the real value weights W^r to $\hat{W}^r$ in a given convolution kernel as follows :

$$\hat{W}^r = \left[\hat{W}^r_j | \hat{W}^r_j = \frac{W^r_j - \mu_{W^r}}{\sqrt{N}\sigma_{w^r}}\right] \tag{3}$$

where μ_{W^r} and σ_{w^r} are the mean and standard deviation of W^r :

$$\mu_{W^r} = \frac{1}{N}\sum_{j=1}^{N} W^r_j, \quad \sigma_{w^r} = \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(W^r_j - \mu_{W^r}\right)^2} \tag{4}$$

After transformation, $\mu_{\hat{W}^r}$ becomes zero. In BNN forward path, real value weights $\hat{W}^r$ will be binarized to 1 or -1, denoted as W^b, according to its sign. After binarization, the μ_{W^b} deviate from zero, shown in Figure 2. As mentioned in [22], the expected value of the convolution output $z = \sum_j^N W^b_j x^b_j$ is given by :

$$\mathbb{E}(z) = N\mu_{x^b}\mu_{w^b} \tag{5}$$

where x^b is the binarized input from previous layer. In most cases, μ_{x^b} is not zero. Because the μ_{W^b} is also not zero, the expected value of z consequently deviates from zero. Since there is no Batchnorm layer after convolution, the binarized threshold is always zero. By doing so, our model generally has wider perceptual margins. We named this modified convolution as *BinWsConv* for simplicity.

B. Reduce involved noise

Since noise is considered as an additive parameter in weight, if the input x^b_j is zero, the noise will be zeroed out. In other words, it is easy to see that when x^b_j is zero, the $\eta_j x^b_j$ will be zero in the following equation :

$$\widetilde{z^r} = \sum_j^N (W^b_j + \eta_j)x^b_j = \sum_j^N W^b_j x^b_j + \sum_j^N \eta_j x^b_j, \tag{6}$$

where N is the fan-in extent of the convolutional kernel. The former part $\sum_j^N W^b_j x^b_j$ is the expected output, which we denote as z^r and latter part $\sum_j^N \eta_j x^b_j$ is the additive noise. According to our binary setting, the summation is accumulated only when $x^b_j = 1$. We denote the quantity of that x^b_j is 1 as n. The expected value and variance of the noise summation are as follows:

$$E(\sum_{j=1}^{n}\eta_j) = \sum_{j=1}^{n} E(\eta_j) = 0 \tag{7}$$

$$Var(\sum_{j=1}^{n}\eta_j) = \sum_{j=1}^{n} Var(\eta_j) = n\sigma^2 \tag{8}$$

According to these two equations, the expected value is always zero since the noise η_j is sampled from the zero-mean Gaussian distribution. However, as the noise is added more terms, the variance is getting larger. Larger variance means the sum of noise is very different from zero, leading to a higher probability of $Sign(z^r) \neq Sign(\widetilde{z^r})$, in other words, wrongly flip. From the implementation point of view, a transistor's current can change when there is a process variation. However, if a transistor is not activated, i.e., the input voltage is low (zero), the variation of the transistor will not cause problems in analog computation. In this section, we will discuss how to cause more zeros in the input of a convolution so that the noise effect can be alleviated.

First, we observe that max pooling layer makes the convolution distribution negatively skew. Since there is no Batchnorm layer in our architecture, the max pooling layer is right after the binary activation function. The max pooling layer needs all binary activations in a sliding window are zero to output a zero; otherwise, it output 1. As a result, we can deduct that

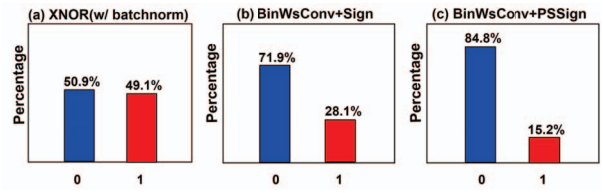

Fig. 3. The percentage of 0 and 1 in binary activation with (a) ordinary XNOR block (with batchnorm) (b) BinWsConv and sign function (c) BinWsConv and proposed PSSign.

there exist more zeros than $1s$ in binary activation to generate sufficient zeros. Recall (1), a negative number is binarized to zero. So we can believe the convolution output distribution eventually negatively skew. As the distribution tends to deviate negatively, although the ordinary sign function is used as a binary activation function, the number of zeros is larger than that of $1s$ in the binary activation.

Assume we will use the max pooling operation in the following discussion. To further increase the ratio of zeros, we propose a new scheme called *Positive Shift Sign* (PSSign). Note that in the convolution, we need a threshold value to determine whether the output is 1 or 0 in our BNN. Given the same convolution, we will have more zeros for the same convolution if we can have a larger threshold value. Our goal here is to increase the threshold value of activation function to cause more zeros. We have the following equation :

$$PSSign(z^r) = \begin{cases} +1, & if \quad z^r > p \\ 0, & if \quad z^r \leq p \end{cases} \quad ,where \quad p > 0 \quad (9)$$

where p is a trainable parameter, and p is constrained to be positive. Instead of training a p per channel, there is only a p per layer, resulting in minimal memory storage. As a result, the ratio of 0 to 1 in binary activation becomes more imbalanced than the ordinary sign function (Fig. 3). Furthermore, since the distribution tends to deviate negatively, we generally have wider perceptual margins by the positive p value.

To reduce the difficulty of deploying the BNN to practical CIM architecture, we reorder layers in a typical XNOR-NET block. The data type of tensors between layers in our order is always integer or binary. We conclude our overall flow in Fig. 4 and illustrate the difference between our Batchnorm-free scheme and the typical BNN block (XNOR-NET block).

V. Experiments

A. Settings

1) Dataset: Our BNN training scheme is mainly designed for edge devices applying the CIM technique, demanding a limited computing resource. Thus, small-footprint Keyword spotting (KWS), which is the task of pursuing low memory footprint, high accuracy, and real-time capability, is a practical application to our work. According to our scenario, we train and evaluate our model on Google's Speech Commands data set (GSCD) [30], which is a widely used dataset for the KWS benchmark. The GSCD contains 64,727 one-second-long utterances and is labeled with one of 30 target categories.

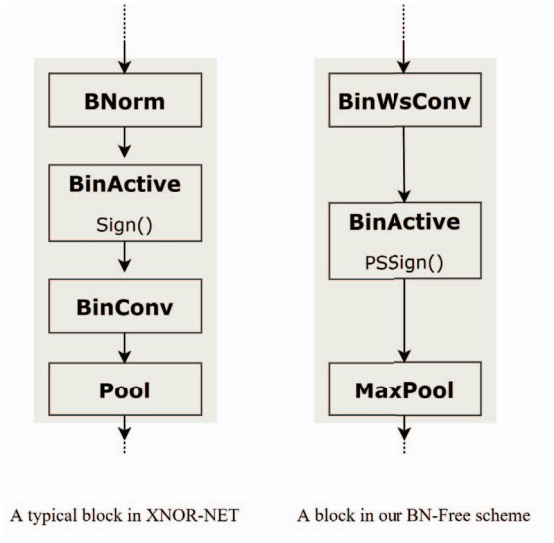

Fig. 4. This figure contrasts the block structure in our BN-free scheme (right) with a typical XNOR-NET (left). Our block eliminates the Batchnorm and reorder the layers.

We follow Google's implementation, having a classification of 12 classes: "yes," "no," "up," "down," "left," "right," "on," "off," "stop," "go," *unknown*, or *silence*. The unknown label is composed of the remaining 20 keywords.

2) Model: In order to pursue the practical BNN deployed on CIM for small-footprint KWS, we use a popular XNOR-NET block [11] as our vanilla model for comparison. As shown in Fig. 4, the left figure shows the implementation of an XNOR-NET structure, while the right figure shows our robust BNN structure.

3) Implementation: We use CIM macro to implement convolution layers of BNN. To lower power consumption for use in a battery-powered device, our keyword spotting does not use the MFCC module. Instead, we use raw audio as inputs without MFCC. The size of the CIM macro is 512kb. The measured throughput, energy efficiency, area efficiency, and energy-area efficiency product are 48.1 $TOPS$, 20943 $TOPS/W$, 37.1 $TOPS/mm^2$, and 16159 $TOPS/W$-mm^2 in the chip, respectively, representing substantial 2.5x, 3.7x, 4.2x, 4.5x improvements compared with the best-known solutions.

4) Noise setup: As described in Section III-B, we assume the distribution of noise to be a zero-mean Gaussian distribution, and the standard deviation is used to control the severity of noise.

B. Experimental result

1) Results on GSCD without noise: We have performed experiments on the GSCD. First, without inserting noise, the results are shown in Table I. Row 2-9 shows the results of other research, while Row 10-12 shows our results. Row 10 shows the results of our vanilla model, row 11 shows the results of our model without using PSSign in Section IV-B and row 12 shows the results of our completed robust model. First, Table I shows

TABLE I
ACCURACY COMPARISON OF RESULTS ON THE GSCD.

Model	Accuracy (%)	Params (Byte)	Input Data
DS-CNN-S [26]	94.4*	24K	MFCC
DS-CNN-M [26]	94.9*	140K	MFCC
DS-CNN-L [26]	95.4*	420K	MFCC
ResNet15 [27]	95.8*	960K	MFCC
TC-ResNet8 [28]	96.1*	264K	MFCC
TC-ResNet14 [28]	96.2*	548K	MFCC
TC-ResNet14-1.5 [28]	96.2*	1220K	MFCC
SincConv+GDSConv [29]	96.6*	248K	Raw Audio
Vanilla model	93.4	85K	Raw Audio
Vanilla+BN-Free block(w/o PSSign)	93.3	80K	Raw Audio
Vanilla+BN-Free block(w/ PSSign)	94.4	80K	Raw Audio

*The numbers are taken from the paper.

TABLE II
ACCURACY COMPARISON OF INTEGRATING WITH NOISE INJECTION TRAINING.

Model	Noise Scale					
	σ_{val} = 0	σ_{val} = 0.2	σ_{val} = 0.4	σ_{val} = 0.6	σ_{val} = 0.8	σ_{val} = 1.0
Vanilla + ($\sigma_{train} = 0.2$)	93.06	89.49	70.43	36.72	19.61	12.13
Ours + ($\sigma_{train} = 0.2$)	**94.36**	**93.78**	91.44	84.37	60.36	30.98
Vanilla + ($\sigma_{train} = 0.4$)	92.97	91.74	83.10	55.28	27.89	14.26
Ours + ($\sigma_{train} = 0.4$)	93.70	93.27	**92.65**	90.28	83.07	59.91
Vanilla + ($\sigma_{train} = 0.7$)	91.12	90.67	88.72	78.74	55.32	29.48
Ours + ($\sigma_{train} = 0.7$)	92.03	91.52	91.45	**90.69**	**89.38**	83.11
Vanilla + ($\sigma_{train} = 1.0$)	87.33	87.11	86.10	80.63	66.07	42.69
Ours + ($\sigma_{train} = 1.0$)	87.94	87.86	87.89	87.50	86.85	**85.28**

TABLE III
VALIDATION ACCURACY COMPARE TO VANILLA MODEL

Model	Noise Scale					
	σ_{val} = 0	σ_{val} = 0.2	σ_{val} = 0.4	σ_{val} = 0.6	σ_{val} = 0.8	σ_{val} = 1.0
Vanilla model	93.4	88.28	63.47	31.39	15.88	10.64
Vanilla+BN-Free block(w/o PSSign)	93.3	91.06	81.32	56.05	26.59	12.77
Vanilla+BN-Free block(w/ PSSign)	**94.4**	**93.03**	**88.6**	**75.50**	**47.37**	**21.34**

that replacing Batchnorm with scaled weight standardization will not affect the accuracy. Again, because our implementation mainly focuses on the ultra-low-power implementation, we use much smaller amounts of parameters with slightly lower accuracy. We would like to iterate that our power consumption is an order of magnitude less than the others.

2) Robustness Evaluation: To compare with accuracy after injecting noise, we apply noise injection training [14], [16] to the vanilla model XNOR-NET called vanilla in the table and figure. We then compare the results with our robust model applying noise injection training. In Table II, the first column shows the model of ours and the vanilla model with different standard deviations of noise during training. Column 2-7 shows the results of inference accuracy after applying different scales of standard deviation. For example, in Row 3, under $\sigma_{val} = 1.0$, the accuracy of our robust model is 83.11% while the accuracy of the vanilla drop to 29.48%. We can say that based on our model, noise injection training has a better ability to make the model adapt to noise during inference.

In Table III, our results are performed only on our robust model without the noise injection training. Same as Table II, Column 1-7 shows the results of inference accuracy after applying different scales of noise. Results show that our model outperforms the vanilla model while noise scale σ_{val} increases from 0.0 to 1.0. The results show that our model indeed boosts the robustness. We also have an ablation study that if we use ordinary sign function instead PSSign, the improvement will decline but still have better robustness in comparison with vanilla model owing to *BinWsConv*.

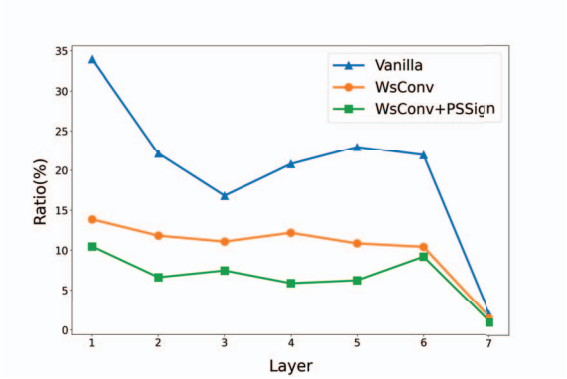

Fig. 5. The ratio of convolution outputs in fragile area in each layer with validation noise scale $\sigma_{validate} = 0.4$.

As mentioned in Section IV-A, if the convolution output is close to the binary threshold, the output is susceptible to noise. We also perform experiments to show that our robust model can "push" convolution outputs away from the binarized threshold.

In our experiments, we assume the standard deviation of noise is 0.4. Fig. 5 shows the results of the experiments. In the experiment, we define a particular interval called *Fragile area* close to the binarized threshold. The Fragile area is defined as the interval of $[BT - \sigma\sqrt{n}, BT + \sigma\sqrt{n}\ \ ]$, where BT is the binarized threshold, and the value n represents the number of $1s$ in fan-in binarized activation. The blue line shows the results of the vanilla model, the orange line shows the results of our model without PSSign, and the green line shows the results of our completed robust model. For example, in Layer 3, in our robust model, the ratio of convolution results in the Fragile area is just 7%, which is much lower than the vanilla counterpart. Again, it also demonstrates the importance of the proposed PSSign. The ratio will rise by replacing PSSign with an ordinary sign function. These experimental results show that we effectively expand the distance between convolution outputs and the binarized threshold to lower noise participation in convolution operation.

VI. Conclusion

BNN deployed on the CIM architecture holds the potential to speed up and reduce power consumption. However, we observe that widely used Batchnorm would harm BNN robustness under noisy analog computation.This paper proposed a Batchnorm-free BNN training scheme to tackle this issue. By leveraging weight standardization and the proposed *Positive Shift Sign*, we can improve accuracy and lower the effect of analog cell noise without any extra module or effort. Besides, our method can integrate with noise injection training to achieve significant advances. Experiments on small-footprint keyword spotting tasks, commonly deployed on the edge device, show our method's effectiveness. The efficiency and improvement of robustness show that it is a promising solution to facilitate the progress of binary neural network in analog computing.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[4] M. Ravanelli and Y. Bengio, "Speaker recognition from raw waveform with sincnet," in *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2018, pp. 1021–1028.

[5] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[6] M. M. Lopez and J. Kalita, "Deep learning applied to nlp," *arXiv preprint arXiv:1703.03091*, 2017.

[7] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6t sram array," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, 2017.

[8] Y.-C. Chiu, Z. Zhang, J.-J. Chen, X. Si, R. Liu, Y.-N. Tu, J.-W. Su, W.-H. Huang, J.-H. Wang, W.-C. Wei, J.-M. Hung, S.-S. Sheu, S.-H. Li, C.-I. Wu, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, and M.-F. Chang, "A 4-kb 1-to-8-bit configurable 6t sram-based computation-in-memory unit-macro for cnn-based ai edge processors," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 10, pp. 2790–2801, 2020.

[9] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[10] 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 preprint arXiv:1602.02830*, 2016.

[11] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[12] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, vol. 105, p. 107281, 2020.

[13] L. Chang, X. Ma, Z. Wang, Y. Zhang, W. Zhao, and Y. Xie, "Corn: In-buffer computing for binary neural network," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 384–389.

[14] M. Qin and D. Vucinic, "Training recurrent neural networks against noisy computations during inference," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2018, pp. 71–75.

[15] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature communications*, vol. 11, no. 1, pp. 1–13, 2020.

[16] C. Zhou, P. Kadambi, M. Mattina, and P. N. Whatmough, "Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation," *arXiv preprint arXiv:2001.04974*, 2020.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[18] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[19] S. Ward-Foxton, "Mythic ai accelerator targets high-end edge with 35 tops," Nov 2020. [Online]. Available: https://www.eetimes.com/mythic-ai-accelerator-targets-high-end-edge-with-35-tops/

[20] M. Klachko, M. R. Mahmoodi, and D. Strukov, "Improving noise tolerance of mixed-signal neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[21] L. Huang, J. Qin, Y. Zhou, F. Zhu, L. Liu, and L. Shao, "Normalization techniques in training dnns: Methodology, analysis and application," *arXiv preprint arXiv:2009.12836*, 2020.

[22] A. Brock, S. De, and S. L. Smith, "Characterizing signal propagation to close the performance gap in unnormalized resnets," in *International Conference on Learning Representations*, 2020.

[23] S. Qiao, H. Wang, C. Liu, W. Shen, and A. Yuille, "Micro-batch training with batch-channel normalization and weight standardization," *arXiv preprint arXiv:1903.10520*, 2019.

[24] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[25] T. Chen, Z. Zhang, X. Ouyang, Z. Liu, Z. Shen, and Z. Wang, "" bnn-bn=?": Training binary neural networks without batch normalization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 4619–4629.

[26] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.

[27] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5484–5488.

[28] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha, "Temporal convolution for real-time keyword spotting on mobile devices," *arXiv preprint arXiv:1904.03814*, 2019.

[29] S. Mittermaier, L. Kürzinger, B. Waschneck, and G. Rigoll, "Small-footprint keyword spotting on raw audio data with sinc-convolutions," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7454–7458.

[30] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.