Improving Robustness of Neural Networks against Bit Flipping Errors during Inference

Minghai Qin, Chao Sun, and Dejan Vucinic
Western Digital Coporation, San Jose, California, USA
Email: {minghai.qin, chao.sun, dejan.vucinic}@wdc.com

***Abstract*—We study the trade-offs between prediction accuracy and storage redundancy of neural networks that are stored in noisy storage media. Parameters of a trained neural network are commonly stored as binary data and it is usually assumed that the data storage and retrieval are error-free. This assumption is based upon the common use of Error Correcting Codes (ECCs) that correct bit flips in storage media. However, ECCs incur capacity and power overhead (10% to 20%) and thus increase cost and reduce the effective bandwidth when retrieving trained parameters from storage during inference. We measured the robustness of several deep neural network architectures and datasets when bit flipping errors exist but ECCs are not used during inference. It is observed that more sophisticated architectures and datasets are generally more vulnerable to bit flipping errors. We propose a simple parameter error detection method, called *weight nulling*, that can universally improve the robustness from twice to several orders of magnitude depending on network architectures.**

***Index Terms*—neural networks, bit flips, error detection**

I. Introduction

Neural Networks (NNs) [1], [2] are layered networks that try to fit the function of neurons in a human brain during object recognition, decision making, etc. They are one of the most widely-used machine learning techniques due to their good performance in practice. Some variants of neural networks are shown to be more suitable for different learning applications. For example, deep Convolutional Neural Networks (CNNs) [3], [4] are found to be effective in recognizing and classifying images. Recurrent Neural Networks (RNNs) [5], [6] provides stronger performance in sequence prediction, e.g., speech or text recognition. Compared to standard feed-forward neural networks with full connections, CNNs have much less number of connections in the convolutional layers and thus much fewer parameters to train, possibly avoiding the over-fitting problems. RNNs have memory units like Long-Short-Term-Memory (LSTM) [7] that can be trained without vanishing/exploding gradient problems.

A neural network is defined by the connections between neurons, each of which is associated with a trainable parameter called a *weight*. There is another parameter associated with each neuron, called a *biase*. Since a bias can be viewed as a weight from a neuron with constant input, we will indiscriminately call it a weight as well. The set of all trainable weights are usually acquired by back-propagation algorithm [8], [9].

In order to fit highly non-linear functions and thus achieve a high rate of correctness in practice, neural networks usually contain millions to billions of weights trained from a large dataset in a careful manner. Current computer technology requires the weights to be stored in Non-volatile Memories (NVMs) and they are loaded to CPU/GPU caches during inference. NVMs are noisy media where bit flipping errors can happen during writing, reading, or retention. Error Correction Codes (ECCs) [10] are ubiquitously used in NVM systems to guarantee data reliability by adding 10% to 20% storage redundancy. There are two major reasons that we study the robustness of neural networks when weights stored in noisy NVM media are not fully recovered by ECCs. Firstly, the GPU caches have limited size (usually in Giga-byte range, but will be much smaller for embedded systems) but the size of the neural network models grow fast as a result of smaller cost of collecting big data. If caches in a single or multiple GPUs, in particular for embedded applications, cannot hold all weights of a neural network, the bandwidth of loading the weights from NVMs to GPU caches would become a bottleneck of system performance, e.g., applications with massive throughput requirement, such as video recognition during self-driving where the number of frames processed per second positively correlates to the safety factor. The storage overhead brought by ECCs will add latency and reduce the effective throughput of the NVM chips. Secondly, there is a growing trend of research on in-memory/neuromorphic computing [11] where computing units for a NN are moved from CPU/GPU to NVMs themselves where the resistance of a memory device will be used as the value of the weight such that error correction is not feasible. The benefit of this in-memory/neuromorphic computing comes from higher parallelism and lower power consumption of NVM systems, but ECCs might have to be weakened or abandoned depending on the design of the computing system.

Robustness of neural networks has been studied against random and adversarial noise to the input of the NNs. Ref. [12] provides adversarial attack algorithms on input and defensive distillation towards evaluating the robustness of neural networks. Ref. [13] proposed

Manuscript received August 10, 2018; revised December 14, 2018.

doi: 10.18178/joig.6.2.181-186

training algorithms that address the issue of output instability when the input is slightly distorted. Ref. [14] and [15] studied neural networks with binary or ternary weights, whose training algorithms are adjusted. But to our knowledge, there has been no study in improving the robustness of weights (represented as binary arrays) against bit flipping errors.

In this paper, we explore the robustness of trained neural networks when they are stored in noisy storage media. For each dataset, an originally undistorted neural network is trained by GPUs and the real-valued weights are stored as binary arrays by fixed-point representations. Each bit in the binary array will be flipped independently with some probability, called *Raw Bit Error Rate* (RBER) of the storage media, and the prediction accuracy of the distorted neural network will be examined. The typical RBER for current NVM technology will range from 10^{-2} to 10^{-6}, depending on their materials and requirement (throughput, latency, cost, etc.). We then propose a detection method, called *weight nulling*, by adding a single check bit for each weight. When reading a weight from storage, we first calculate if the check bit equals the modulo-2 sum of all other bits. If they are not equal (called a check fail), it is guaranteed that some bit in this weight are erroneous and we null the weight by setting it to be a zero. Multiple check bits, e.g., Cyclic Redundant Checks (CRCs), are not used since 1-bit check brings the smallest overhead and the performance improvement is already prominent. This weight nulling method can detect a single bit flipping error, which dominates multiple bit flipping errors in probability. Note that the weight nulling method is closely related to DropConnect [16] or Dropout [17], which were only used during training and the connections/neurons to drop are randomly selected. On the other hand, the weight nulling method is used during inference and is targeted to all connections that cause check fails. The tolerance of RBER at the same prediction accuracy with weight nulling has been improved by several times to orders of magnitude, which is validated by experiments on different datasets and neural network architectures. Note that the ultimate goal is not to optimize the prediction accuracy of the original neural networks, but to maintain relatively high accuracy as RBER increases so that it can be used with high device-to-device differences and uneven quality of NVM storage chips resulting from unavoidable manufacturing variability.

II. Preliminaries

A. Neural Networks and Notations

A neural network contains input neurons, hidden neurons, and output neurons. It can be viewed as a function $f : \mathcal{X} \to \mathcal{Y}$ where the input $x \in \mathcal{X} \subseteq \mathcal{R}^n$ is an n-dimensional vector and the output $y \in \mathcal{Y} \subseteq \mathcal{R}^m$ is an *m*-dimensional vector. In this paper, we focus on classification problems where the output $y = (y_1, \ldots, y_m)$ is usually normalized such that $\sum_{i=1}^{m} y_i = 1$ and y_i can be viewed as the probability for some input *x* to be categorized as the *i*-th class. The normalization is often done by the softmax function that maps an arbitrary *m*-dimensional vector $\hat{y}$ into normalized y, denoted by $y = softmax(\hat{y})$, as $y_i = \frac{\exp(\hat{y}_i)}{\sum_{i=1}^{m} \exp(\hat{y}_i)}, i = 1, \ldots, m$. For top-*k* decision problems, we return the top *k* categories with the largest output y_i . In particular for hard decision problems where *k*=1, the classification results is then $\arg\max_i y_i, i = 1, \ldots, m$.

A feedforward neural network f that contains *n* layers (excluding the softmax output layer) can be expressed as a concatenation of *n* functions $f_i, i = 1, 2, \ldots, n$, *n* such that $f = f_n(f_{n-1}(\cdots f_1(x)\cdots))$. The *i*th layer $f_i : \mathcal{X}_i \to \mathcal{Y}_i$ satisfies $\mathcal{Y}_i \subseteq \mathcal{X}_{i+1}, \mathcal{X}_1 = \mathcal{X}$. The output of last layer $\mathcal{Y}_n$ is then fed into the softmax function. The function f_i is usually defined as

$$f_i(x) = \sigma(W \cdot x + b)$$

where *W* is the weights matrix, *b* is the bias vector, and σ is an element-wise activation function that is usually nonlinear, e.g., sigmoid and rectified linear unit (ReLU). Both *W* and *b* are trainable parameters.

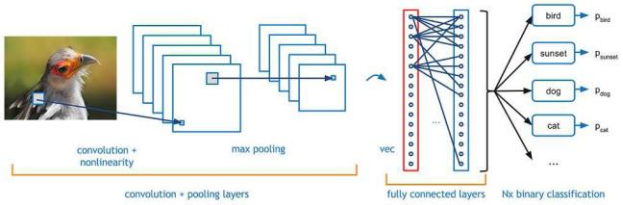

Figure 1. Example of a CNN

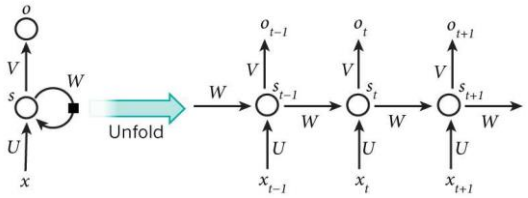

Figure 2. Unfolding a RNN

A Convolutional Neural Network (CNN) (Fig. 1) is a special class of feedforward neural network that has local weights constraints, e.g., the weights connecting neurons are all zeros except for a few pair of neurons between adjacent layers, and the value of weights between different pair of neurons in two layers with similar spatial relationships are forced to be the same. Therefore, a CNN layer has much less parameters to train compared to a fully connected layer and is good at extracting local features from the previous layer, which enables it to be the state-of-art technique for image recognition problems. A Recurrent Neural Network (RNN) is a special class of neural networks that has directed cycles, which enable it to create internal states and exhibit temporal behaviors. A RNN can be unfolded (Fig. 2) in time to form a feedforward neural network for training purposes. One of the most widely used neurons to store the states is LSTM,

consisting of forget-gate, update-gate, and output-gate. Back-propagation algorithms can be applied from the last output neurons backwards to train all weights in the RNN.

B. Real-Valued Weights and Their Binary Representations

The originally undistorted weights of a NN is trained by GPU and are represented by IEEE Standard for Floating-Point Arithmetic (IEEE 754). We will briefly review IEEE 754 and argue that floating-point representation is not an appropriate method when bit flipping errors exist. Then we introduce an unsigned fixed-point representation based on quantization and it will serve as the basis for representing real numbers in this paper.

1) IEEE Standard for Floating-Point Arithmetic (IEEE 754): IEEE 754 provides guidelines to represent a real number r as $(-1)^s \times c^q$ by 16, 32, 64, 128, and 256 bits, where s is 1-bit of sign, c is a significand, and q is an exponent. For example, a 16-bit representation assigns 5 bits to the exponent with bias equal to 15, and the rest 11 bits are used for the 10-bit significand and the 1-bit sign. The largest number that can be represented is $(2 - 2^{-10}) \times 2^{15} = 65504$ which usually is much larger than any weight in a neural network.

With the presence of media errors, IEEE 754 standard is not a proper representation of real-valued weights. The major weakness is due to the exponent representation. In particular, if the most significant bit in the exponent is erroneous, the value of that weight can be inadvertently set to a very large value. For example, the binary string 0 01101 0101010101 represents $(-1)^0 \times 2^{13-15} \times 1.3330078125 \approx 0.33$, but if the second bit is flipped and the string becomes 0 11101 0101010101, it will represent $(-1)^0 \times 2^{29-15} \times 1.3330078125 = 21840$. This large weight will destroy the learned neural network. Since the number of weights is large and each weight has a few "vulnerable" bits (e.g., some most significant bits in the exponents), it is likely that some of them are flipped, resulting in poor robustness against the media errors.

2) Fixed-Point Arithmetic with unsigned representation: Fixed-point representation avoids the troublesome exponent part in IEEE 754 standard. It can be either signed or unsigned representations depending on whether to allocate one bit for the sign. For signed representation, the maximumly (positive) and minimumly (negative) representable values are almost the same (can differ by one quantization interval). However, since the distribution of weights are not strictly symmetric around zero, we use the unsigned fixed-point arithmetic throughout this paper, where a direct quantization of real numbers between the minimum-valued weight and the maximum-valued weight is applied. Assume the minimum and maximum weight is denoted by w_{min} and w_{max}, respectively. To convert a real-valued weight into a length-q binary array, the interval $[w_{min} - \Delta, w_{max} + \Delta]$ is be quantized into 2^q consecutive subintervals with boundaries $w_{min} - \Delta = b_0 < b_1 < \cdots < b_{2^q} = w_{max} + \Delta$, where $\Delta = \frac{w_{\max} - w_{\min}}{2^q - 1}$ is the size of subintervals. For all weights $w \in [w_{min}, w_{max}]$, if w is in the ith interval, i.e., $b_i \leqslant w < b_{i+1}$, then w is represented by the q-bit unsigned binary representation of the integer i as $(i_0, i_1, \ldots, i_{q-1})$. To convert a binary array to a real-valued weight, the following equation is used for decoding:

$$w_{\text{dec}} = w_{\min} + \Delta \times \left(\sum_{j=0}^{q-1} i_j 2^{q-1-j} \right) \quad (1)$$

III. Improving Robustness by Weight Nulling

In this section, we explore the robustness of different neural network architectures for different datasets and show that weight nulling can improve the tolerable RBER at the same prediction accuracy.

Suppose we have an undistorted neural network model with N weights and each weight is represented by q bits in Section II-B2, then the total number of bits is qN. The noisy storage media is modeled as a Binary Symmetric Channel (BSC) [18], where each bit is independently flipped with probability p (called RBER of the media, denoted by BSC(p)). For each p, M distorted models are obtained by passing the undistorted model through a BSC(p) M times. Thus, each distorted model has on average pqN bit flipping errors. We will test the validation accuracy of all M distorted models and use that M values to approximate the statistics, in particular, the mean value of M validation accuracy for a certain p. Note that the fixed-point representation has a weakness in that the most significant bit (MSB), i.e., the leftmost bit in the binary array, is much more vulnerable because the distortion caused by MSB is $\frac{w_{\max} - w_{\min}}{2}$ while distortions of other bits are exponentially decreasing. Therefore, it is desirable to have a small variance among all validation accuracy, which is also explored in our experiments.

In order the measure the robustness of neural networks against storage media errors, we introduce a robustness measure $R(x)$, which is defined as follows. Suppose the undistorted model has validation accuracy $A \in [0, 1]$, then $R(x)$, $x \in [0, 1]$ is defined as the maximum RBER that the average validation accuracy is larger than or equal to Ax. We will use $x = 0.99$ and $x = 0.95$ as two criteria, that is, $R(0.99)$ (or $R(0.95)$) is the maximum RBER that a neural network architecture can tolerate when the average validation accuracy of distorted models degrade at most 1% (or 5%) compared to the undistorted model.

A. Weight Nulling during Inference

Assume we have a trained NN model in which each weight w is to be encoded into q bits as $(i_0, i_1, \ldots, i_{q-1})$ and stored. The weight nulling requires that $i_{q-1} = \sum_{j=0}^{q-2} i_j$ where the summation is modulo-2, i.e., i_{q-1} is the check bit for the (q – 1)-bit representation of w. For inference purpose, suppose a weight is read from storage as $(\hat{i}_0, \hat{i}_1, \ldots, \hat{i}_{q-1})$, where $i_j \neq \hat{i}_j$ with

probability p (RBER of the noisy storage media), we first check whether $\hat{i}_{q-1} = \sum_{j=0}^{q-2} \hat{i}_j$. If the equation is satisfied, we assume there is no bit flipping errors in the length-q array and will use it to calculate the input of a neuron; Otherwise, we can guarantee that at least 1 bit flipping error exists and we can then delete the connection by setting the weight as 0. The improvement of weight nulling is based on two major observations. First, RBER is usually small, thus the probability of one bit flipping error is dominating multiple errors and we can almost detect all distorted weights. Second, deleting a connection during inference is much less harmful than distorting the weight to be a random value. Note that weight nulling has 1 bit less in quantization compared to q-bit representation without weight nulling. Nonetheless, the gain in detecting distorted weights over-weighs the loss in 1 bit quantization, especially when q is not too small, e.g., we use q = 8 in our experiment.

B. A 6-Layer CNN for MNIST

The MNIST database is a database of handwritten digits that is commonly used for image processing and machine learning. It includes 60000 training images and 10000 test images of size 28 × 28. We train a 6-layer convolutional neural network to classify the 10 digits with the architecture in Table I.

TABLE I. CNN ARCHITECTURE FOR MNIST

Layer	Output shape	No. of trainable parameters
Conv. 2D(3,3)	(26, 26, 32)	320
Conv. 2D(3,3)	(24, 24, 32)	9248
Maxpooling	(12, 12, 32)	0
Conv. 2D(3,3)	(10, 10, 64)	18496
Conv. 2D(3,3)	(8, 8, 64)	36928
Maxpooling	(4, 4, 64)	0
Fully connected	(256)	262400
Fully connected	(10)	2570

Fig. 3 shows the comparison with and without weight nulling during inference. We use 8-bit fixed-point representation and each point in the figure is the average validation accuracy of 50 distorted models. The error bar surrounding each point is the standard deviation for those 50 distorted models. Two dashed horizontal lines indicates the 99% and 95% of the undistorted model's accuracy (which is 99.6%), respectively. The robustness measure (R(0.99), R(0.95)) of the 6-layer CNN have been increased from (0.006, 0.012) to (0.025, 0.034) by weight nulling. With the same RBER, it can also be observed that the variance of the distorted models has been reduced, resulting in more predictable inference performance with the presence of bit flipping errors in storage media. In order to explore the robustness of each convolutional layer or fully connected layer against errors, we test the validation accuracy when weights in each layer are distorted individually. The top figure in Fig. 4 shows the average prediction accuracy over 40 distorted models for each RBER where bit flipping errors appear in only one of the six layers. It can be observed that the robustness is decreasing from the first to the fourth convolutional layer, possibly due to the fact that the number of weights is largely increased from the first to the fourth. In order to take the total number of weights into account, the bottom figure in Fig. 4 shows the average validation accuracy versus the total number of bit flipping errors in each layer. It is reasonable to see that layers with more weights can tolerate more bit flipping errors at the same validation accuracy.

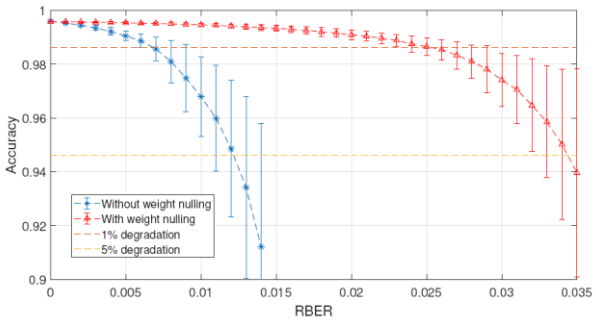

Figure 3. Validation accuracy of the 6-layer CNN for MNIST with and without weight nulling.

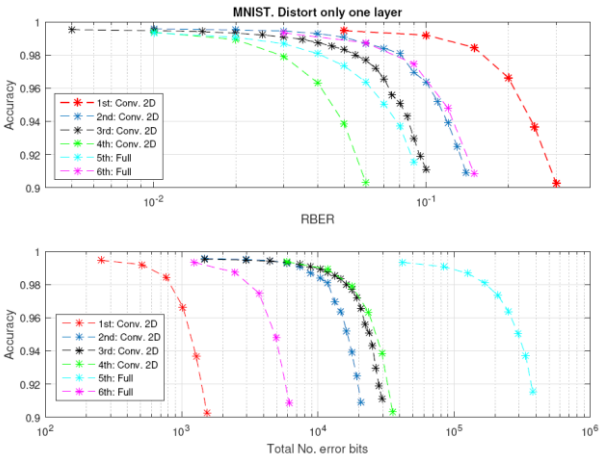

Figure 4. Validation accuracy when only one of the six layers has bit flipping errors.

C. A 28-Step LSTM for MNIST

LSTMs are typically used for sequence prediction and classification, such as language and speech, but they are also capable of recognizing images if we sequentially feed each row of the image to one LSTM cell. As a result, the number of LSTM cells (called steps) equals the number of rows in the image and the input dimension of one LSTM cell equals the number of columns in the image. Table II summarizes the LSTM architecture in our experiments. The input to each LSTM cell is a row in the image.

TABLE II. A 28-STEP LSTM ARCHITECTURE FOR MNIST

No. LSTM cells	28
Input shape	(28,1)
LSTM internal state size	128
LSTM output state size	128
Total No. trainable parameters	81674

Fig. 5 shows the comparison with and without weight nulling during inference. Similar to the setup of the 6-

layer CNN, 8-bit fixed-point representation is used and 50 distorted models are then obtained for each RBER. The undistorted LSTM model has validation accuracy 98.57%. The robustness measure ($R(0.99)$, $R(0.95)$) of the LSTM have been increased from (0.0003, 0.001) to (0.003, 0.006) by weight nulling.

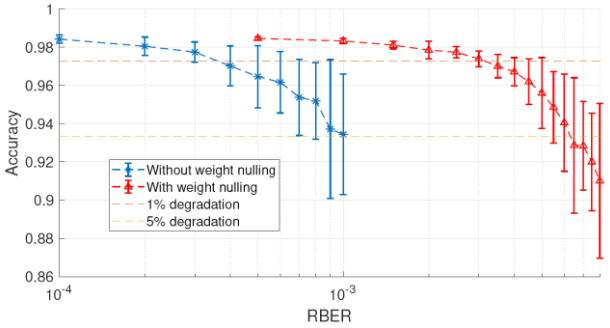

Figure 5. Validation accuracy of the 28-step LSTM for MNIST with and without weight nulling.

D. A Depth-40 Growth-Rate-12 Densely Connected Convolutional Neural Network for CIFAR-10

The CIFAR-10 dataset consists of 60000 32×32 color images in 10 classes. We use a densely connected convolutional neural network [19] (densenet), which is one of the best results for CIFAR-10 dataset. The depth is 40 and the growth rate is k = 12 in our experiment. The reported validation accuracy in [19] of the undistorted model without data augmentation is 93%, which is close to what we observe (92.5%) after 300 epochs of training. Fig. 6 shows the comparison with and without weight nulling during inference. 8-bit fixed-point representation is used and 100 distorted models are then obtained for each RBER. The robustness measure ($R(0.99)$, $R(0.95)$) of the densenet have been increased from (10^{-5}, 2×10^{-5}) to (4×10^{-4}, 9×10^{-4}) by weight nulling.

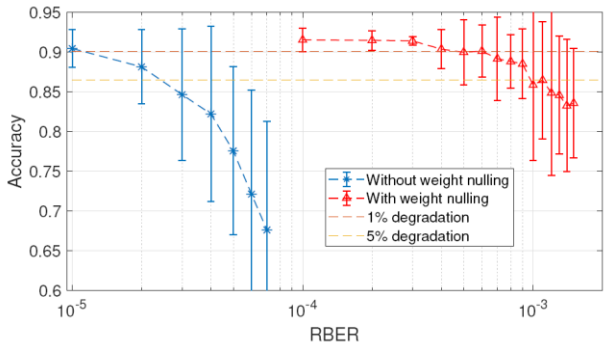

Figure 6. Validation accuracy of the densenet for CIFAR-10 with and without weight nulling.

E. A VGG-16 for Cats-Dogs Classification

VGG-16 [20] has been a widely used deep neural network for image classification. We explore the robustness of the VGG-16 by categorizing a dataset of cats and dogs provided by Kaggle. We use 20000 images for training and 5000 for validation. The convolutional layers of VGG-16 is pretrained and we train the last few fully connected layers to achieve validation accuracy of 98.76% with an undistorted VGG-16 model. The 16-bit fixed-point representation is used and 40 distorted models are then obtained for each RBER. Fig. 7 shows the comparison with and without weight nulling during inference. The robustness measure ($R(0.99)$, $R(0.95)$) of the VGG-16 have been increased from (2×10^{-7}, 10^{-6}) to (1.5×10^{-4}, 2.5×10^{-4}) by weight nulling.

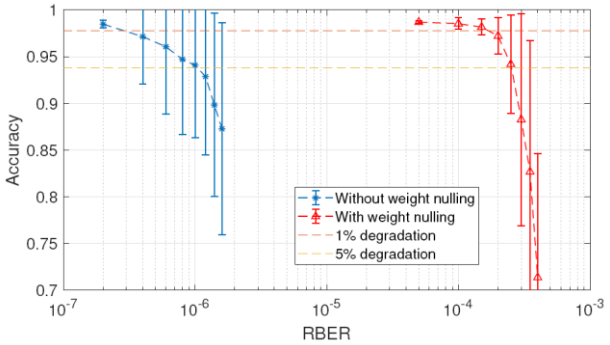

Figure 7. Validation accuracy of the VGG-16 for cat-dog classifications with and without weight nulling.

F. A Summary of Weight Nulling Improvement during Inference

Table III summarizes the comparison of robustness measure $R(0.99)$ and $R(0.95)$ of different datasets and NN architectures with and without weight nulling. According to this table, we believe that more complicated image recognition tasks with more sophisticated NNs has much less robustness against the storage media errors. However, the improvement brought by the weight nulling method during inference is more significant.

TABLE III. ROBUSTNESS IMPROVEMENT BY WEIGHT NULLING

	MNIST	MNIST	CIFAR-10	Cats-dogs
NN architecture	CNN	LSTM	densenet	VGG-16
$R(0.99)$ w/o wt. null.	0.006	0.0003	$1E{-}5$	$2E{-}7$
$R(0.99)$ w/ wt null.	0.025	0.003	$4E{-}4$	$1.5E{-}4$
$R(0.99)$ improv.	4.2x	10x	40x	750x
$R(0.95)$ w/o wt. null.	0.012	0.001	$2E{-}5$	$1E{-}6$
$R(0.95)$ w/ wt null.	0.034	0.006	$9E{-}4$	$2.5E{-}4$
$R(0.95)$ improv.	2.8x	6x	45x	250x

IV. CONCLUSIONS

In this paper, we study the robustness of neural networks against media errors for different neural network architectures and datasets. We propose a weight nulling method during inference that can improve the robustness (tolerable RBER with the same validation accuracy) by a few times to orders of magnitude, depending on the DNN architectures and datasets.

REFERENCES

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85-117, 2015.
[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, New York: Springer, 2001.
[3] K. Jarrett, K. Kavukcuoglu, and M. Ranzato, "What is the best multi-stage architecture for object recognition?" in *Proc. IEEE Int. Conf. Computer Vision*, Kyoto, Japan, September 2009, p. 2146-2153.
[4] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira,

C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097-1105.
[5] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for improved unconstrained handwriting recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855-868, May 2009.
[6] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," 2014.
[7] K. Greff, R. Srivastava, J. Koutn k, B. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Networks and Learning Systems*, pp. 1-11, July 2016.
[8] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the RPROP algorithm," in *Proc. IEEE Int. Conf. Neural Networks*, San Francisco, CA, USA, March 2009, pp. 586-591.
[9] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Proc. Int. Joint Conf. on Neural Networks*, Washington, DC, USA, 1989, pp. 593-605.
[10] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*, Cambridge University Press, 2009.
[11] W. Zhao, G. Agnus, V. Derycke, A. Filoramo, J. Bourgoin, and C. Gamrat, "Nanotube devices based crossbar architecture: toward neuromorphic computing," *Nanotechnology*, vol. 21, no. 17, April 2010.
[12] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. on Security and Privacy*, San Jose, CA, USA, 2017, pp. 39-57.
[13] S. Zheng, Y. Song, T. Leung, and I. Goodfellow, "Improving the robustness of deep neural networks via stability training," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, 2016, pp. 4480-4488.
[14] M. Courbariaux, Y. Bengio, and J. David. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. [Online]. Available: http://arxiv.org/abs/1511.00363
[15] P. Merolla, R. Appuswamy, J. Arthur, S. Esser, and D. Modha. (2016). Deep neural networks are robust to weight binarization and other non-linear distortions. [Online]. Available: http://arxiv.org/abs/1606.01981
[16] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research*, S. Dasgupta and D. McAllester, Eds., Atlanta, Georgia, USA: PMLR, 2013, vol. 28, no. 3, pp. 1058-1066.
[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929-1958, 2014.
[18] T. Cover and J. Thomas, *Elements of Information Theory*, New York, NY, USA: Wiley-Interscience, 1991.
[19] G. Huang, Z. Liu, and K. Q. Weinberger. (2016). Densely connected convolutional networks. CoRR, vol. abs/1608.06993. [Online]. Available: http://arxiv.org/abs/1608.06993
[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. International Conference on Learning Representations*, 2014.

Minghai Qin is a research staff member in Next Generation Platform Technologies, Western Digital Research since May 2016. Before that, he is with Storage Architecture, HGST, a Western Digital Company since Sept. 2014. He received his Ph.D. in Electrical Engineering in Sept. 2014 from the Department of Electrical and Computer Engineering, UCSD, where he was also associated with the Center for Magnetic Recording Research (CMRR) from 2010 to 2014.

Chao Sun received the B.S. and M.S. degrees in electrical engineering and automation from the Harbin Institute of Technology, Harbin, China, in 2009 and 2011, respectively, and the Ph.D. degree in electrical engineering and information system from the University of Tokyo, Tokyo, Japan. He was an Assistant Professor with Chuo University, Tokyo, Japan, from 2014 to 2015. Currently, he is a Research Staff Member (Sr. Principal Engineer/ Technologist) with the Western Digital San Jose Research Center, San Jose, CA, USA. His current research interests include high performance storage systems, cloud computing and artificial intelligence.

Dejan Vucinic earned a Ph.D. in experimental particle physics from MIT in 1998. He is now a director of NVM systems architecture group at Western Digital Corporation's San Jose Research Center.