



Titre: Robust Quantization for Enhanced Energy Efficiency and Bit Error
Title: Tolerance in DNNs

Auteur: Kamran Chitsaz Zade Allaf
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Chitsaz Zade Allaf, K. (2023). Robust Quantization for Enhanced Energy Efficiency
and Bit Error Tolerance in DNNs [Mémoire de maîtrise, Polytechnique Montréal].
Citation: PolyPublie. <https://publications.polymtl.ca/53397/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/53397/>
PolyPublie URL:

**Directeurs de
recherche:** François Leduc-Primeau, & Jean Pierre David
Advisors:

Programme: Génie électrique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Robust quantization for enhanced energy efficiency
and bit error tolerance in DNNs**

KAMRAN CHITSAZ ZADE ALLAF

Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie électrique

Mai 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

**Robust quantization for enhanced energy efficiency
and bit error tolerance in DNNs**

présenté par **Kamran CHITSAZ ZADE ALLAF**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

Pierre LANGLOIS, président

François LEDUC-PRIMEAU, membre et directeur de recherche

Jean Pierre DAVID, membre et codirecteur de recherche

Sarath CHANDAR ANBIL PARTHIPAN, membre

DEDICATION

To all brave women of Iran

RÉSUMÉ

Assurer la robustesse des réseaux de neurones profonds (RNP) est crucial pour leur déploiement fiable dans des scénarios incertains, tels que des environnements à faible consommation d'énergie ou non sécurisés, et contre les attaques adverses. Les perturbations de poids dans les réseaux quantifiés peuvent dégrader considérablement les performances, soulignant la nécessité d'améliorer la résilience des RNP dans des scénarios incertains. Ce mémoire présente deux nouvelles approches pour améliorer la robustesse d'un RNP contre les inversions de bits lors de l'inférence.

Tout d'abord, nous introduisons un mécanisme d'apprentissage qui minimise la plage de quantification des différentes couches pendant l'apprentissage et établit un équilibre entre robustesse et performance. Notre travail démontre le rôle critique de la plage de quantification dans la robustesse d'un réseau et comment elle peut être ajustée pour équilibrer la performance en l'absence de bruit et la robustesse aux perturbations du modèle.

Nous proposons également une méthode probabiliste pour estimer avec précision le gradient du modèle de bruit non dérivable, ce qui aide à entraîner les réseaux de neurones de manière robuste contre les erreurs d'inversion de bits. Nous utilisons l'astuce Gumbel-Max pour échantillonner la distribution de probabilité perturbée et réduire la variance de l'estimation du gradient pour obtenir une meilleure convergence.

Nos résultats expérimentaux démontrent que les méthodes de quantification proposées peuvent tolérer un bruit élevé tout en maintenant une précision similaire à un modèle non perturbé pour différentes architectures de réseaux et plusieurs ensembles de données, ce qui peut améliorer la fiabilité et la sécurité des accélérateurs RNP dans des domaines critiques. De plus, notre approche améliore considérablement la robustesse contre les attaques d'inversion de bits adverses. Par rapport aux méthodes d'entraînement robustes existantes, nous améliorons constamment la frontière de Pareto concernant la performance du réseau et sa consommation d'énergie sur une variété de modèles.

ABSTRACT

Ensuring the robustness of deep neural networks (DNNs) is crucial for their reliable deployment in uncertain scenarios, such as low-power or unsecured environments, and against adversarial attacks. Weight perturbations in quantized networks can significantly degrade performance, underscoring the need to improve DNN resilience in uncertain scenarios. This thesis presents two novel approaches to enhance DNN robustness against bit-flip errors during inference.

First, we introduce a learning mechanism that minimizes the quantization range of different layers during training and strikes a balance between robustness and performance. Our work demonstrates the critical role of the quantization range in a network’s robustness and how it can be adjusted to balance performance in noiseless conditions and robustness in uncertain scenarios.

Second, we propose a probabilistic method to accurately estimate the gradient of the non-differentiable noise model, which helps to train neural networks robustly against bit-flip errors. We use the Gumbel-Max trick to sample from the perturbed probability distribution and reduce the variance of the gradient estimation to achieve better convergence.

Our experimental results demonstrate that the proposed quantization methods can tolerate high noise while maintaining accuracy similar to the baseline on different model architectures and datasets, which can improve the reliability and security of DNN accelerators in critical domains. Moreover, our approach significantly enhances robustness against adversarial bit-flip attacks. Compared to state-of-the-art robust training methods, we consistently improve the Pareto frontier concerning test accuracy and energy consumption across a variety of models.

TABLE OF CONTENTS

DEDICATION	iii
RÉSUMÉ	iv
ABSTRACT	v
TABLE OF CONTENTS	vi
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF SYMBOLS AND ABBREVIATIONS	xi
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 LITERATURE REVIEW	5
2.1 GPU-Based Accelerators	5
2.2 Data Redundancy for Optimizing DNNs	6
2.3 Pruning	7
2.4 Quantization	8
2.5 Processing in Memory	10
2.6 Energy Estimation	10
2.7 Design Space Exploration	11
2.8 Sources of Bit Flips in Memories	11
2.9 Bit-Flip Attack	12
2.10 Defense Methods	14
CHAPTER 3 THE WCAT METHOD FOR ROBUST QUANTIZATION	15
3.1 Preliminaries	15
3.1.1 Optimization Challenges	16
3.1.2 Quantization Range and Robustness: Insights from a Toy Model	17
3.2 WCAT Method	19
3.2.1 Weight Clipping	19
3.2.2 Weight Quantization	20
3.2.3 Straight-through Gradient Estimation	20

3.2.4	Learned Quantization Range	21
3.2.5	Weight Decay	23
3.2.6	Bit Flip Noise (BFN) Injections	23
CHAPTER 4 EXPERIMENTAL RESULTS ON WCAT		24
4.1	Experimental Settings	24
4.2	Regularization	27
4.3	Gradient Scaling	28
4.4	Quantization Range Initialization	28
4.5	Random Attack	29
4.6	Adversarial Attack	30
4.7	Energy Consumption	31
CHAPTER 5 ROBUST QUANTIZATION WITH BINARY GUMBEL		33
5.1	Background	33
5.1.1	Gumbel-Max Trick	34
5.2	Method	35
5.2.1	Binary Gumbel	36
5.2.2	Using Perturbed Logits to Imitate Bit-Flip Noise Injection	37
5.2.3	Bias-variance tradeoff in gradient estimation	37
5.3	Experimental results	38
CHAPTER 6 CONCLUSION		41
REFERENCES		42

LIST OF TABLES

Table 2.1	Top-1 accuracy of ResNet-50 on ImageNet for different quantization methods. The baseline floating-point top-1 accuracy is 76.9%	9
Table 2.2	Summary of defense techniques against bit-flip attacks benchmarked on CIFAR-10	14
Table 4.1	Robustness of ResNet-20 on CIFAR10 for Normal 4-bit Quantization with Different Levels of Bit-Flip Noise Injection During Training . . .	25
Table 4.2	Robustness of ResNet-20 on CIFAR10 and ResNet-18 on Imagenet with 4 bit quantization for different gradient scales	27
Table 4.3	Comparison of robustness for ResNet-18 on ImageNet using different quantization range initialization	28

LIST OF FIGURES

Figure 1.1	Illustration of the impact of bit-flip attacks on a deep neural network. On the left, it shows how bit flipping of weights results in misclassification. On the right, the decision boundary of a clean network (black) and an attacked network (blue) is shown, demonstrating the increase in misclassification caused by bit-flip attacks	2
Figure 2.1	Illustration of a dataflow mapping for DNN inference	7
Figure 2.2	Bit error rate measurements show the impact of supply voltage scaling on SRAM memories in Quentin SoC (Figure from [51])	12
Figure 2.3	Effect of Individual Bit-Flips on WideResNet with 4-bit Quantization in the First Convolutional Layer	13
Figure 3.1	Effect of Random Bit-Flip Noise and Additive Gaussian Noise on Weight Perturbations for 8-bit Quantization	17
Figure 3.2	Loss landscape comparison between regular and perturbed loss functions for a simple linear regression with 4-bit quantization weights and varying levels of noise	18
Figure 3.3	Optimal quantization range under bit error rate noise	19
Figure 3.4	Comparison of the forward pass and gradient of the quantizer with respect to the quantization range and real-valued weights for WCAT and LSQ	22
Figure 4.1	Weight distribution of each layer of ResNet-20 trained with different regularization coefficients	26
Figure 4.2	Robustness of different quantization methods on ResNets with 4-bit weight quantization	29
Figure 4.3	Robustness of different quantization methods on ResNets with 8-bit weight quantization	30
Figure 4.4	Impact of adversarial bit flip attacks on network performance using ResNets with 4 and 8-bit weight quantization	31
Figure 4.5	The modified Eyeriss-like architecture	31
Figure 4.6	Classification accuracy in terms of energy per inference on CIFAR-10	32
Figure 5.1	Bit flip noise model	33
Figure 5.2	Comparison of Gumbel and Normal noise for generating samples close to categorical distributions	35
Figure 5.3	Stochastic computation graph for the proposed method	37

Figure 5.4	Tradeoff between bias and variance of Gumbel-Softmax estimator with respect to temperature τ	38
Figure 5.5	Comparison of the accuracy and robustness of Lenet5 with 2-bit weight quantization	39
Figure 5.6	Robustness of different quantization methods on Lenet5 with 2-bit weight quantization	39
Figure 5.7	Comparison of robustness for different quantization methods on CIFAR10 using ResNet20 with 2-bit quantization	40

LIST OF SYMBOLS AND ABBREVIATIONS

ALU	Arithmetic Logic Unit
AVFS	Adaptive Voltage and Frequency Scaling
BFN	Bit Flip Noise Injections
DNN	Deep Neural Network
DSE	Design Space Exploration
FFT	Fast Fourier Transform
LSQ	Learned Step Size Quantization
MAC	Multiply-accumulate operation
MLP	Multi-layer Perceptron
NAS	Neural Architecture Search
PE	Processing Element
PIM	Processing In Memory
PLClip	Per-Layer Aggressive Weight Clipping
RHA	Row Hammer Attack
SMID	Single Instruction Multiple Data
SMIT	Single Instruction Multiple Threads
STE	Straight-Through Estimator
WCAT	Weight Clipping-Aware Training

CHAPTER 1 INTRODUCTION

Research on deep neural networks (DNNs) has risen as a popular field due to their extraordinary performance in a wide range of complex recognition and classification tasks. The widespread application of DNNs in critical domains, such as self-driving cars [1] and medical diagnosis [2], has emphasized the need to study their reliability and performance in noisy environments. Also, to improve their performance, custom hardware accelerators have emerged as a promising solution given the high computational demands of DNNs. However, it is essential to ensure that DNNs retain their efficiency and reliability when deployed on specialized hardware with a limited energy budget and in the presence of hardware faults. Quantized networks deployed on special-purpose hardware are particularly susceptible to weight perturbations, highlighting the importance of enhancing DNN robustness in uncertain scenarios such as low-power or unsecured environments. By improving the resilience of DNNs against weight perturbations, we can develop more reliable models, especially in critical domains where the consequences of incorrect classification can have significant impacts.

Despite the fact that DNNs tend to be over-parameterized and, as a result, contain some redundancy, they can still be impacted by minor bit-flip errors, making it necessary to develop methods that enhance their robustness during inference. This is particularly important for components and structures that hold DNN-related data, like memory hierarchy in GPUs and FPGAs, which can be vulnerable to attacks. While compressed and quantized DNN weights can improve robustness and reduce bit-flip errors and overall overhead [3], even a few vulnerable bits can result in significant performance degradation [4]. It is therefore imperative to develop effective methods for enhancing the robustness of DNNs in challenging environments as well as ensuring their reliability and security.

Bit flips in memories can occur randomly as a result of undesirable events such as chip fabrication variations and supply voltage variations, or targeted attacks. The row-hammer attack (RHA), where multiple accesses can disturb DRAM rows and cause bit flips in other rows, has been extensively researched [5]. DRAMs are highly susceptible to this type of attack due to their dense architecture. However, research has focused more on SRAM memory from an energy-reliability standpoint. The energy consumption of DNN accelerators is dominated by the memory accesses [6]. To address this issue, researchers have explored various dataflows that can leverage data reuse opportunities to optimize the number of memory access for different DNN shapes and sizes [6]. However, reducing the memory supply voltage has become a popular approach to reduce energy consumption, particularly in SRAM, due to the

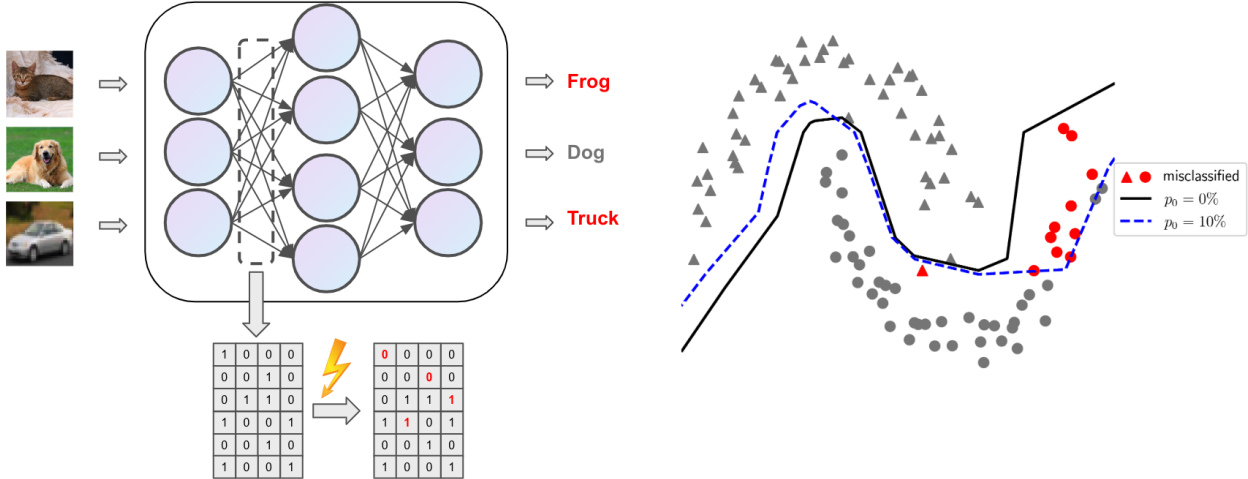


Figure 1.1 Illustration of the impact of bit-flip attacks on a deep neural network. On the left, it shows how bit flipping of weights results in misclassification. On the right, the decision boundary of a clean network (black) and an attacked network (blue) is shown, demonstrating the increase in misclassification caused by bit-flip attacks

quadratic relationship between dynamic energy consumption and supply voltage. Despite its benefits, near-threshold voltage processing can cause bit-flip errors in SRAMs, which may compromise the reliability of the DNN accelerator. Additionally, another promising solution for energy-efficient memory management is processing in memory (PIM), which involves using non-volatile resistive memory devices to perform computations in the memory. While PIM has shown promising results in terms of energy efficiency, programming the values of the conductance in memristor crossbars can be an imprecise process subject to noise, leading to potential errors in the computations. As a result, improving the robustness of DNNs is essential for ensuring resilience to bit-flip errors and promoting their reliable deployment in real-world applications.

In recent years, there has been increased focus on developing defense methods to improve the robustness of DNNs against weight perturbations in uncertain scenarios and under adversarial attacks. The fault tolerance capability of DNNs can be improved by modifying the network's size, structure, and training process. Some proposed methods involve improving the robustness of DNN models by adding memory or computational overhead [7, 8]. Another approach involves compressing and binarizing [9] the weights of the DNNs to limit the change after being attacked, but this can lead to a performance decrease on clean weights.

Additionally, the DNN can be trained to be robust against faulty data, such as by simulating bit-flip errors during training and dynamically adjusting the weights to enhance resistance to

bit-flipping [10]. Recently, aggressive weight clipping [11] has been proposed as an efficient alternative to increase robustness against bit errors during training. While defense methods for enhancing the robustness of DNNs against weight perturbation have shown promising results, they do come with certain limitations. Sacrificing noiseless DNN performance or adding computation overhead are common trade-offs with these methods. These limitations highlight the need for further research in developing efficient and effective defense methods that balance both robustness and performance.

In this thesis, we introduce two approaches that aim to enhance the resilience of DNNs against bit errors during inference. First, instead of relying on conventional techniques, we take a novel approach by introducing a learning mechanism for the quantization range of different layers during training. Our experimental results demonstrate that the quantization range plays a crucial role in network robustness, as it is a delicate trade-off between the quantization error and the dynamic range. While reducing the quantization range decreases the quantization error due to rounding, it also leads to a decrease in the dynamic range, which is vital in neural networks for representing weights with large absolute values [12]. To this end, our approach aims to minimize the quantization range, striking a balance between robustness and performance on clean weights. Second, by emphasizing challenges in training a robust neural network against bit-flip error, we propose a novel solution to accurately estimate the gradient in presence of bit-flip noise. We use the Gumbel-Max trick to estimate the gradient of the XOR operation by sampling from the perturbed probability distribution. By incorporating these learning mechanisms during training, we ensure that the network maintains its performance under noiseless conditions while achieving greater robustness against bit-flip errors during inference.

Our contributions can be summarized as follows:

- Proposing a robust quantization scheme that learns the quantization range of different DNN layers and reduces it to promote robustness against bit errors at inference time. Simulated bit-flip errors are incorporated during training for further improvements.
- Introducing a careful gradient approximation technique to improve convergence during training and avoid reducing the performance of the clean network.
- Proposing probabilistic model that leverages the Gumbel-Max trick to facilitate accurate gradient estimation. Specifically, we utilize perturbed probability distributions to sample from and simulate the effect of bit-flip noise during backpropagation.

This thesis is organized into six chapters:

Chapter 2, Literature Review: We review previous work related to DNN accelerators, quantization techniques, unreliable memories, and defending against random and adversarial bit flip attacks.

Chapter 3, The WCAT Method for Robust Quantization: In this chapter, we propose a mathematical definition of the problem and discuss its limitations. We present “weight clip aware training” to improve robustness.

Chapter 4, Experimental Results on WCAT: In the fourth chapter, we describe the implementation details of WCAT and demonstrate that our approach outperforms existing defense techniques on multiple networks and datasets.

Chapter 5, Robust Quantization with Binary Gumbel: We start with the motivations behind using the Gumbel-max trick for approximating the gradient of non-differentiable noise models, then propose our binary Gumbel estimation.

Chapter 6, Conclusion: Finally, we summarize our key findings and suggest future research directions.

CHAPTER 2 LITERATURE REVIEW

Inference and training are two distinct phases in the lifetime of a DNN model. Although training is computationally expensive and requires a significant amount of resources, the inference is executed repeatedly, leading to its energy consumption playing a dominant role in overall compute energy. Hence, researchers have focused on developing efficient hardware implementations for DNNs inference to make them suitable for embedded devices. DNN accelerators aim to reduce their computational complexity using architectural optimization and simplification techniques. In this chapter, we discuss different techniques for accelerating DNN inference. We start with general-purpose architectures (Section 2.1), and then explain the opportunities in specialized accelerators (Section 2.2). We discuss other techniques such as compression (Section 2.3, 2.4), near-data processing (Section 2.5), and design space exploration (Section 2.7), which are essential for optimizing the hardware implementation of DNNs. Additionally, we discuss the sources of bit flip in memories (Section 2.8) and various types of attacks, including random and adversarial attacks (Section 2.9), and recent defense techniques against them (Section 2.10). Finally, we conclude by highlighting areas for further research in this field to achieve high-performance and reliable DNN accelerators.

2.1 GPU-Based Accelerators

The computations of neural networks can be simplified into basic components, including convolution, matrix multiplication, and element-wise operations. These fundamental components are implemented with multiply-accumulate (MAC) operations which can be executed efficiently by parallelization. GPUs with their numerous ALUs are well-suited for accelerating matrix multiplications through parallelization techniques such as SMID and SMIT, which allow for high-performance computations of MAC operations [6]. Additionally, computational transform techniques like FFT based convolution [13], the Strassen method, and the Winograd method [14] can help to reduce the number of multiplications and accelerate matrix multiplications on these platforms. Recent years have seen the development of DNN-specific GPU designs, such as NVIDIA’s Ampere GPUs [15], which support structured sparsity within the tensor cores, resulting in significantly faster performance for deep learning workloads.

Reference [16] discusses a fault injection attack on GPU kernels using overdrive-based data corruption induced by perturbing the voltage and frequency of the GPU. The paper highlights the vulnerability of adaptive voltage and frequency scaling (AVFS) to this type of

attack and evaluates the susceptibility of different GPU instructions. Building on this work, Reference [2] shows that the GPU overdrive attack was shown to be a serious threat to deep learning-based recognition engines in medical image classification, and software and hardware countermeasures are suggested.

2.2 Data Redundancy for Optimizing DNNs

Processing DNNs requires large amounts of data movement between memory and processing units and can be computationally intensive. This data movement bottleneck limits the performance and efficiency of DNN processing and has motivated the development of specialized accelerators designed specifically for DNN inference tasks. These accelerators are optimized to address the temporal design and memory hierarchy constraints of general-purpose architectures and often incorporate local memories to reduce memory access latency. Various dataflows have been proposed to improve memory hierarchy utilization and reduce the energy consumption of DNN processing by storing frequently accessed data closer to the processor.

A valid dataflow mapping can be represented as a nested set of loops, as illustrated in Figure 2.1, which shows a multi-channel 2D convolution(left), and its corresponding architecture with two levels of memory and a single processing element (PE) (right). The figure shows how different data types, including input, weight, and output, are transferred between the memory hierarchy levels. The loop bounds (M, P, Q, R, S) are mapping parameters that must correspond to problem dimensions and fit data structures in the relevant memory hierarchy level. Each cycle entails the transfer of input, weight, and output blocks between the main memory and buffer. This transfer is conditioned on whether the requisite blocks are available in the buffer or not. If not, they must be read from the main memory and stored in the buffer. DNN dataflows are classified based on their data handling characteristics, including weight stationary, output stationary, and row stationary, as described in [6].

Output Stationary

The output-stationary dataflow by changing the order of nested loops enables outputs (partial sums) to be stored in local memory for reuse in the next iterations. Data movement for outputs can be reduced with this approach, resulting in improved energy efficiency. In order to maximize data reuse opportunities, output activations can be generated from different channels or the same channel, depending on the network configuration.

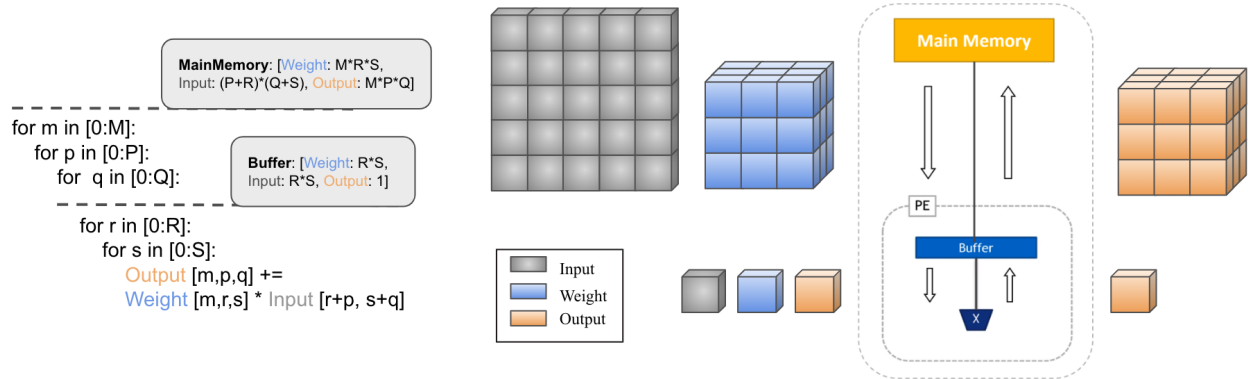


Figure 2.1 Illustration of a dataflow mapping for DNN inference

Weight Stationary

Another approach is to store the weights in local memories which is known as weight-stationary dataflow. This approach can reduce the energy consumption required for reading weights. Reference [17] proposes a 1D tile architecture for accelerating the convolutional layers in VGG-based networks, which uses a weight generator consisting of shift registers to provide the correct weight for each PE. By reusing a subset of N neurons falling in the same output row and using p parallel tiles, the computational latency and memory access can be reduced. The optimal values of N and p are determined based on the target timing constraint and silicon budget.

Row Stationary

The row-stationary dataflow used in the Eyeriss chip [18] shows impressive performance for high-dimensional convolution. In this dataflow, the number of accesses for all data types is optimized. The optimal mapping is proposed for the best energy efficiency based on the DNN configuration and hardware constraints, for example the number of PEs, the size of local memories, bitwidth, and so on. Experimental results on convolutional layers of AlexNet show that row stationary has the best energy efficiency.

2.3 Pruning

Pruning techniques have shown promising results in reducing the memory requirements for DNN by removing the weights and neurons from the network. Due to the overparameterized nature of DNNs, a significant portion of the weights are redundant and can be removed

without any degradation in accuracy. Reference [19] shows that it is possible to remove even up to 50% of the weights without any accuracy degradation. By retraining the remaining weights to compensate for the removed connections, we can improve the network performance and prune more weights in this iterative process.

Pruning methods are generally based on the magnitude of the weights. Reference [20] is able to reduce the storage size of AlexNet by $35\times$ without any drop in accuracy on Imagenet, by compressing the parameters using magnitude-based pruning, quantization, and Huffman coding. Another approach is to use the gradient of the weights as a proxy for their importance. Reference [21] uses first and second-order Taylor expansion of the training loss function to approximate the importance of each filter.

In unstructured pruning, any individual weight can be removed without any specific pattern. This can significantly decrease the model size since it can maintain better accuracy with a higher pruning rate compared to structured pruning [22]. On the other hand, structured or coarse-grained pruning involves the removal of entire filters or neurons, making it easier to accelerate and more compatible with modern devices. In convolutional layers, different levels of granularity can be applied to structured pruning, ranging from channel-level and kernel-level pruning to pattern-based pruning.

Although recent studies have demonstrated that sparsity can enhance the robustness of DNNs against adversarial attacks [23–26], pruning is not typically considered as a defense against bit-flip attacks.

2.4 Quantization

Reducing the precision of weights and activations of DNN models can significantly reduce their energy consumption. According to Nvidia’s benchmark of A100 GPUs, INT8 tensor operations are $4\times$ faster than the standard 32-bit floating-point format, resulting in improved performance [27]. Furthermore, low bitwidth quantization can reduce the memory storage needed for DNNs. There are two regimes for quantization: post-training quantization and quantization-aware training. Post-training methods aim to quantize a pre-trained model without further fine-tuning. Reference [28] proposed an innovative quantization method for large language models that can compress 65 billion parameter models on a single commercial GPU.

Also, Binary Neural Networks [29], Ternary Weight Nets [30], and other compressed models have been proposed to use simple operations such as XOR [31] or shift instead of MACs, each one exhibiting a different tradeoff between accuracy and model compression.

Table 2.1 Top-1 accuracy of ResNet-50 on ImageNet for different quantization methods. The baseline floating-point top-1 accuracy is 76.9%

Quantization Range	Method	Contribution	Bitwidth		Top-1 Accuracy (%)
			W	A	
Fix range	ACIQ [32]	Analytical range based on the mean of the absolute values of the weights	8	4	71.45
	OCS [33]	Duplicates channels containing outliers, then halve the channel values.	8	8	75.6
	SAWB [34]	Used the first and second moment of the weights to determine the quantization range.	4	4	76.5
Learnable range	NICE [35]	Finetuning with pseudo quantization noise and parameterized clipping function	4	4	76.5
	TQT [36]	Adding power-of-2 scaling constraints for to optimization.	4	8	73.2
	LCQ [37]	Optimizing non-uniform quantization levels during training	4	4	76.6
	LSQ [38]	Scaling the gradient of step size based on weight shape	4	4	76.7

Quantization-Aware Training

Low-precision neural networks can benefit from quantization operations during the training process to learn quantization-aware parameters and improve performance. Although the quantization range is usually set to the maximum absolute value of the weight to ensure that the largest integer represents the largest weight value, it may cause high rounding error and consequently low robustness. Finding the optimal quantization range can be achieved based on the statistics of the weights [12] or by duplicating channels with weight outliers and halving their values to avoid outlier distortion when performing weight clipping [33].

Meanwhile, learning the quantization range during training has emerged as a promising method for low-precision quantization. However, this approach can be difficult because the quantizer itself is not continuous, so approximating the gradient of the quantizer is required. Among these methods, QIL [39] proposed a clipping function that balances the trade-off between the quantization range and sparsification. Nice [35], introduced a parameterized clipping function that is applied before quantization. TQT [40] proposed a quantization scheme with a learnable scaling factor and defined the gradient based on the straight-through estimator (STE). LSQ [41] used a similar method but added gradient scaling to achieve state-of-the-art performance on the Imagenet dataset. We summarize these methods in Table 2.1.

2.5 Processing in Memory

A novel idea for decreasing the power consumption of memory movements is to bring computation inside the memory. By leveraging this approach, the distance between computations and memory can be reduced. Processing in memory (PIM) is one promising solution that moves some of the computations inside the memory. Recently researchers proposed efficient hardware implementation based on PIM which reduces energy consumption and speeds up DNN performance. For instance, XNOR-POP [42] uses in-memory processing for binary DNNs and reduces more than 90% of energy consumption compared to state-of-the-art methods.

The MAC operation can be integrated into a non-volatile memory such as a memristor. There are several non-volatile resistive memory devices including phase change memory (PCM), resistive RAM (RRAM or ReRAM), and conductive bridge RAM (CBRAM). Reference [43] proposes PRIME, a PIM structure for ReRAM-based main memory with carefully designed peripheral circuits that allow arrays to be used as memory, scratchpads, and dot product engines for DNN workloads.

The results show that PRIME improves the performance by $2360\times$ and the energy consumption by $895\times$, across the evaluated machine learning benchmarks compared to state-of-the-art methods. Although memristor crossbars offer energy benefits, programming their conductance values can be an imprecise process that is susceptible to noise interference [44], which highlights the need to develop DNN models robust to weight perturbations.

2.6 Energy Estimation

One way to estimate the performance of a DNN workload on a given hardware accelerator is to simulate the process of transferring data between memory hierarchies. By analyzing the access count of different components in the architecture, we can estimate the energy consumption of the DNN workload based on the energy models for each component. To achieve this, it is necessary to specify the dataflow, which shows how data is staged in the memory hierarchy. Aladdin [45] is a pre-RTL power/performance simulator that has proven to be a remarkable tool for this task. In addition, Accelergy [46] is an architecture-level design estimation framework that provides highly accurate energy estimations for evaluating DNN accelerators.

2.7 Design Space Exploration

Design space exploration (DSE) is a method to find the optimal neural architecture that balances accuracy with another criterion, such as robustness, energy efficiency, or optimal latency. Achieving high prediction accuracy while minimizing energy consumption in DNN accelerators is challenging due to the large and complex design space and the time-consuming evaluation process for each candidate. To address this challenge, several techniques have been proposed, such as hyperparameter optimization [47], Adaptive DNN [48], and neural architecture search (NAS). Additionally, Reference [49] provides a comparative analysis of different constrained optimization techniques for exploring the architectural design space. Recent research has also introduced the concept of early growth, which expands binary neural network architecture during training to improve noise tolerance and incorporate robustness as a multi-objective criterion [50].

2.8 Sources of Bit Flips in Memories

DRAM process technology is scaled continuously, making it more challenging to maintain memory reliability as the smaller cell size and tighter spacing between cells favor EM coupling effects that can affect the leakage rate of the charge stored in each cell. This can cause a fault if a cell leaks more charge than its noise margin. Reference [5] exposes the vulnerability of commodity DRAM chips to disturbance errors caused by the repeated toggling of a DRAM row's wordline, which stresses inter-cell coupling effects that accelerate charge leakage from nearby rows. By continuously reading from the same row in DRAM, it is possible to corrupt data in nearby rows, inducing errors in most DRAM modules.

Clock frequency and power supply voltage are crucial for digital circuits in a processor to operate correctly. Every frequency has a critical supply voltage, and the supply voltage should not go below this critical value, otherwise the circuits may malfunction and produce errors [52]. Reference [53] proposes a novel programmable architecture for SRAMs that dynamically boosts the supply voltage of SRAMs to different target voltages to achieve superior levels of energy efficiency without compromising on output accuracy. Also, Reference [51] provides a comprehensive study on SRAMs and measurement of the bit error rate on large SRAM to characterize this memory in ultra-low energy regimes.

The plot in Figure 2.2 demonstrates that lowering the VDD results in an exponential increase in the BER. Additionally, the quadratic relationship between the dynamic power and supply voltage provides an excellent opportunity for balancing energy consumption and reliability.

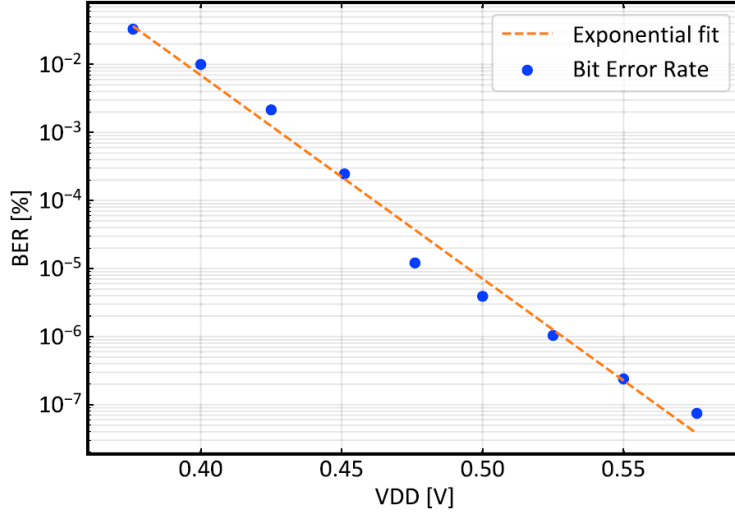


Figure 2.2 Bit error rate measurements show the impact of supply voltage scaling on SRAM memories in Quentin SoC (Figure from [51])

We can infer the following model from Figure 2.2:

$$\epsilon = \exp [a + bV + cV^2]$$

where V is the supply voltage, ϵ is the bit error rate, and $a = 22.12$, $b = -68.14$, and $c = 0$.

2.9 Bit-Flip Attack

The bit-flip attack is a malicious technique that manipulates the parameters of a DNN model at runtime to reduce its performance. Although random bit flips applied to a network’s weights can degrade performance, they can generally withstand a relatively large number of such flips before performance is significantly degraded. Therefore, an attacker is looking to identify vulnerable bits by leveraging information about the network architecture, the parameters, or the dataset.

To understand the impact of individual bits on the accuracy of a DNN model, we conducted an experiment on WideResNet with 4-bit quantization, where we flipped each individual bit in the first convolutional layer and measured the resulting accuracy. As shown in Figure 2.3, we found that most of the most significant bits (MSBs) have the most impact on accuracy, and flipping a single bit at this layer can reduce the accuracy by 2.5%. However, finding the importance of individual bits can be computationally demanding for large models, and researchers have proposed various approximation methods to solve this problem.

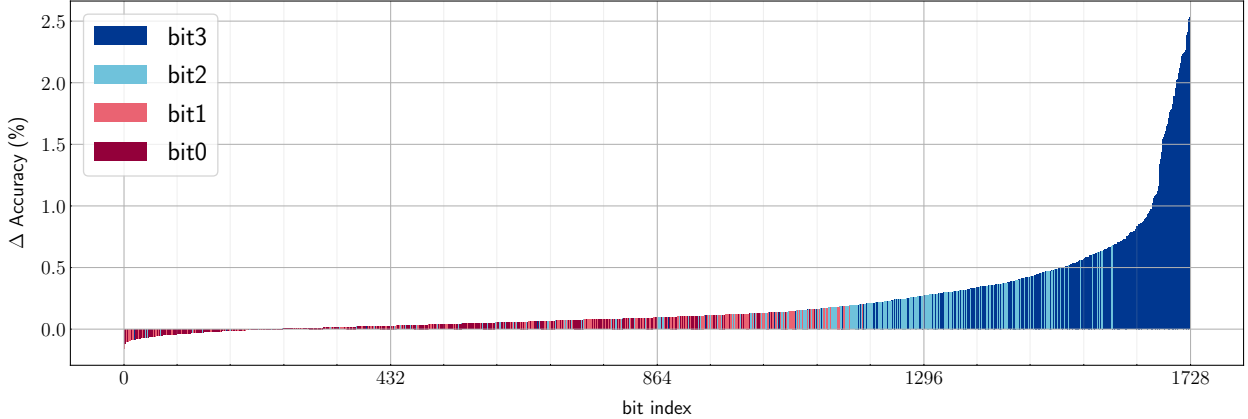


Figure 2.3 Effect of Individual Bit-Flips on WideResNet with 4-bit Quantization in the First Convolutional Layer

Reference [4] proposes a progressive bit search based on the approximated gradient with respect to the bits of the network. The goal of this method is to reduce the network’s performance to a random guess. The objective function of the attack is given by

$$\begin{aligned} \max_{\{\hat{\mathbf{B}}_l\}} \mathcal{L} \left(f_{\{\hat{\mathbf{B}}_l\}_{l=1}^L} (x), y \right) \\ \text{s.t. } \mathcal{H}(\hat{\mathbf{B}}_l, \mathbf{B}_l) \leq N_b, \end{aligned} \quad (2.1)$$

where \mathcal{L} is the loss function, \mathbf{B}_l is the binary representation of the weights at layer l , $\hat{\mathbf{B}}_l$ is the perturbed binary representation of the weights attacked by the BFA, x is a training input, and y its corresponding label. The constraint in (2.1) is given in terms of the Hamming distance \mathcal{H} between the attacked binary weights and the original binary weights, and N_b is the maximum number of bit flips allowed.

Additionally, Reference [54] proposes a Blind Data Adversarial Bit-Flip Attack, which similarly uses gradient ranking but does not require access to a dataset. In addition to the aforementioned untargeted objectives, a targeted bit-flip attack has also been proposed by Reference [55]. This technique allows for the selective targeting of only a few classes of the DNN model while keeping the accuracy for non-targeted classes unchanged. This approach results in a more controlled and specific type of bit-flip attack.

Table 2.2 Summary of defense techniques against bit-flip attacks benchmarked on CIFAR-10

Type of Attack	Method	Defense method	Challenge	Result
Random	FAT	introduce noise injection layer similar to drop-out layers	only benchmarked for stuck-at noise	improving hardware cost and DNN tolerance against stuck-at noise
	WCLIP	asymmetric fixed-point quantization with aggressive weight clipping	loose performance on noiseless network for aggressive robustness	less than 3.3% accuracy drop with 1% BER
	PLCLIP	adaptive weight clipping based on statistics of the weight	loose performance on noiseless network and adding overhead in training	less than 2.3% accuracy drop with 1% BER
Adversarial	ModelShield	optimized hash-based weight verification	adding time and memory overhead	tolerate 10× more bit flip against BFA
	RA-BNN	binarization and Early Growth to compensate noiseless accuracy	increasing model size compares to baseline	tolerate 125× more bit flip against BFA
	Binary	propose weight binarization and its piece-wise clustering relaxation	loose performance on noiseless network	tolerate 19.3× more bit flip against BFA

2.10 Defense Methods

Several defense techniques have been proposed to enhance the resistance of neural networks against bit-flip attacks. While some of these methods involve adding additional overhead, such as utilizing error-correction code units within SRAM cells [56] or implementing hash-based weight verification schemes to correct errors before the data is input to the DNN [8], others, such as binarization [9] and adversarial training [57], can increase robustness at the expense of sacrificing the performance of the unperturbed (or “clean”) model. In addition to binarization, [50] proposed a further enhancement in robustness by augmenting the model’s capacity during training. Reference [58] introduced a noise injection layer that functions similarly to a dropout layer in order to enhance the fault tolerance of a quantized DNN against stuck-at-noise.

In a recent study, [11] has proposed an innovative approach to improve the robustness of neural networks by combining asymmetric fixed-point quantization with aggressive weight clipping during training. In addition, [59] has also proposed to apply per-layer weight clipping, where a specific quantization range is set for each layer. This approach provides additional robustness, as different layers may require different quantization ranges depending on the magnitude and distribution of the weights. By applying per-layer weight clipping, the network can better adapt to these differences and optimize the quantization ranges for each layer accordingly. The effectiveness of these techniques was evaluated on the CIFAR-10 dataset, and the results are summarized in Table 2.2.

CHAPTER 3 THE WCAT METHOD FOR ROBUST QUANTIZATION

In this chapter, we present weight clipping-aware training (WCAT), our proposed method for achieving robust quantization. We begin by discussing preliminaries, challenges, and opportunities in training a robust neural network (Section 3.1). Then in Section 3.1.2 we present our insights based on the toy examples that motivate the proposed approach are presented. To reduce the quantization error, we first clip the weights (Section 3.2.1) and then apply linear weight quantization (Section 3.2.2). Then, we employ the STE [60] mechanism to update the weights that are both inside and outside the clipping range during training (Section 3.2.3). Additionally, the optimal clamping value for each layer is also learned (Section 3.2.4) and is further reduced during training using weight decay (Section 3.2.5). Finally, we discuss optionally applying bit-flip noise injection during training (Section 3.2.6) to further improve the network robustness in highly noisy regimes. We provide a detailed description of each step in the following sections.

3.1 Preliminaries

Random bit-flip noise and additive noise (such as Gaussian noise) are two distinct types of noise that can affect weights. To understand the difference between them, let us start by defining a quantization function $Q()$ that maps a real-valued weight w_i to an integer grid $\{0, 1, \dots, 2^{N_b} - 1\}$, where N_b is the bitwidth. We can also define a de-quantization function $DQ()$ that recovers the real-valued weight w_i from its quantized representation w_i^{int} . These two functions together map values in the vector of the weights to the nearest bins, and we can define the quantization range α as the largest quantization level. In random bit-flip noise, we randomly select bits and flip them. This means that the perturbed representation of the weight, denoted by \tilde{w}_i , is obtained as follows:

$$\tilde{w}_i = DQ(w_i^{\text{int}} \oplus \epsilon), \quad \epsilon \sim \text{Bernoulli}(p_o), \quad (3.1)$$

where $w_i^{\text{int}} = Q(w_i)$, \oplus denotes the XOR operation, and p_o represents the bit error rate. On the other hand, additive noise can be modeled as $\tilde{w}_i = w_i + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

3.1.1 Optimization Challenges

One key difference between random bit-flip noise and additive noise is that for large absolute values of w , the difference between the perturbed representation \tilde{w} and the original weight w can be much more significant than in the case of additive noise. This can be seen in Figure 3.1, where on the left, the bit error rate is $p_o = 0.01$, and on the right: the variance of Gaussian noise is $\sigma = 0.1$. Original weights (x-axis) are plotted against perturbed weights (y-axis). The color represents the difference between the perturbed weight and the original weight. Also, to ensure the robustness of a neural network against random noise, a desirable objective is to minimize the average loss of the network over all possible noise realizations. Let $f_w(x)$ be a network with weights w , and let (x, y) be a training input and its corresponding label from the training set. The desired objective to minimize can be defined as

$$l = \mathbb{E}_\epsilon [\mathcal{L}(f_{\tilde{w}}(x), y)], \quad (3.2)$$

where \mathcal{L} is the loss function, \tilde{w} is a perturbed representation of the weight w , and \mathbb{E}_ϵ denotes the expectation with respect to ϵ . As the number of weights increases, the expectation in (3.2) becomes intractable, and we need to approximate it using Monte-Carlo estimation:

$$l \simeq \frac{1}{m} \sum_{i=1}^m \mathcal{L}(f_{\tilde{w}_i}(x), y), \quad (3.3)$$

where \tilde{w}_i is the perturbed weights with sample of noise realization ϵ_i , and m is the number of Monte Carlo samples. We can approximate the gradient of the loss with respect to the weights of the network as follows:

$$\nabla_w \simeq \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial \tilde{w}} \mathcal{L}(f_{\tilde{w}_i}(x), y) \frac{\partial \tilde{w}}{\partial w}, \quad (3.4)$$

where $\frac{\partial \tilde{w}}{\partial w}$ denotes the gradient of \tilde{w} with respect to w . For additive noise, this gradient is trivially equal to 1. However, for bit-flip noise, the XOR operation is not differentiable, and we cannot directly compute the gradient using (3.4). This makes optimization challenging and requires specialized techniques to overcome.

Moreover, to train a robust neural network that performs well on noisy weights, the regular loss may not be sufficient, and optimizing the average loss of the network over all possible noise realizations, as defined in Equation (3.2), is preferred. However, the optimal solution for regular loss, represented by $w^* = \arg \min \mathcal{L}(f_{\tilde{w}}(x), y)$, may differ from the optimal solution for the perturbed objective, denoted as $w^{**} = \arg \min \mathbb{E}_\epsilon [\mathcal{L}(f_{\tilde{w}}(x), y)]$.

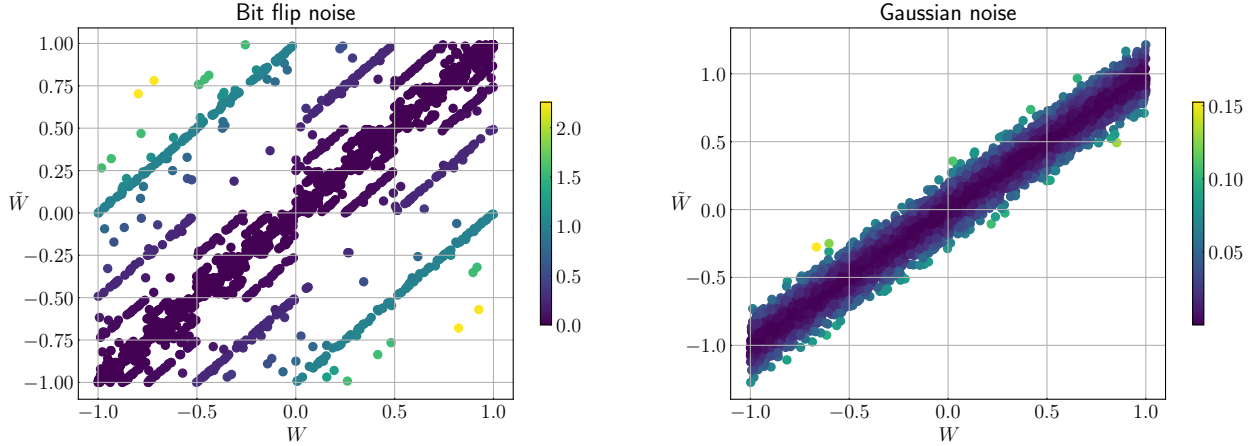


Figure 3.1 Effect of Random Bit-Flip Noise and Additive Gaussian Noise on Weight Perturbations for 8-bit Quantization

This can be proven by providing a counterexample. Consider a simple linear regression $f_w(x) = w^\top x + b$, where $w \in \mathbb{R}^{d \times 1}$ with 4-bit quantization and b is the bias. For simplicity, let us consider the case where $d = 2$. The loss function for this regression model is commonly defined as the mean squared error loss,

$$\mathcal{L}(f_w(x), y) = \frac{1}{(N-1)^2} \sum_{i=0}^{N-1} (f_w(x_i) - y_i)^2, \quad (3.5)$$

where N is the number of training samples. Suppose we have a training set consisting of the input-label pairs (x, y) , where $y = w_{true}^\top x + b_{true} + \epsilon$, and $\epsilon \sim \mathcal{N}(0, 0.1)$.

To investigate the effect of noise on the optimization of the network, we consider the loss landscape for the regular loss and the exact expected perturbed loss for different levels of noise. Figure 3.2 shows this loss landscape for the toy example considered. As we can see, the value that minimizes the perturbed objective function is different from the optimal weight for the regular loss function, and this difference becomes more significant as the level of noise increases. This observation emphasizes the difficulties involved in training a neural network to achieve robustness against random bit flip noise.

3.1.2 Quantization Range and Robustness: Insights from a Toy Model

As previously observed in Figure 3.1, bit-flip noise has a more pronounced effect on weight perturbations for larger absolute values of the weights. To further explore this phenomenon, we conduct an empirical analysis of the impact of quantization range on the resilience of

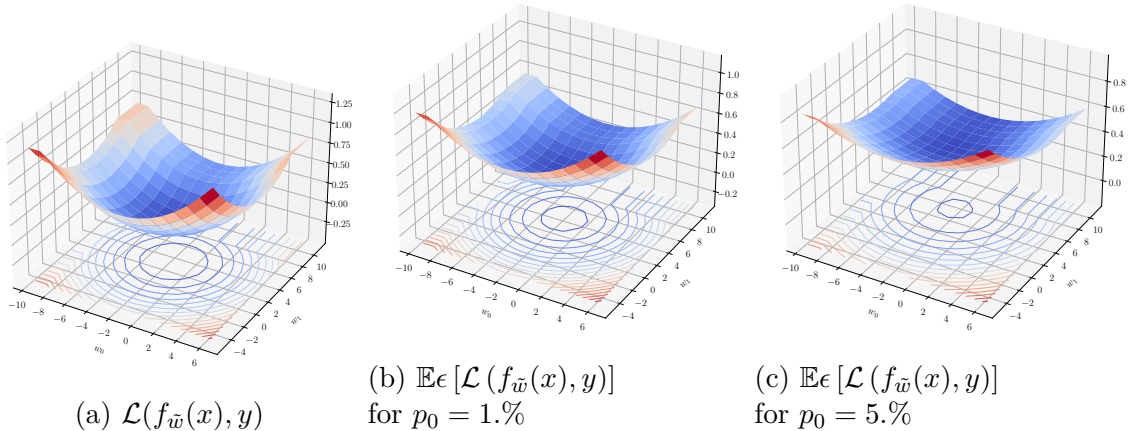


Figure 3.2 Loss landscape comparison between regular and perturbed loss functions for a simple linear regression with 4-bit quantization weights and varying levels of noise

different levels of noise, which is presented in Figure 3.3. Specifically, we examine the effect of quantization range on the optimal performance of a simple multi-layer perceptron (MLP) with a single hidden layer and 100 neurons trained on the MNIST dataset in the presence of random bit error noise, as shown in Figure 3.3a. To this end, we employ cross-entropy as the objective function and use the same quantization range for the first and second fully connected layers. We evaluate our toy model on the test set for different levels of noise, and to estimate the exact loss of each trial, we use Monte-Carlo estimation, which involves sampling random instances of the noise and computing the average loss. Our results indicate that increasing the bit error rate leads to a smaller optimal quantization range.

We further investigated the impact of the quantization range on the quantization error for perturbed weights. We know that for a given weight distribution, there is an optimal quantization range α^* that minimizes the quantization error. Specifically, this optimal range α^* can be found by minimizing the following equation:

$$\alpha^* = \arg \min \alpha_w \|w - DQ(w_{int})\|^2. \quad (3.6)$$

Moreover, we can find the optimal range for the expected quantization error under a bit error rate p_0 by minimizing the following equation:

$$\alpha^* = \arg \min \alpha_w \mathbb{E}_{\epsilon} [\|w - DQ(w_{int} \oplus \epsilon)\|^2]. \quad (3.7)$$

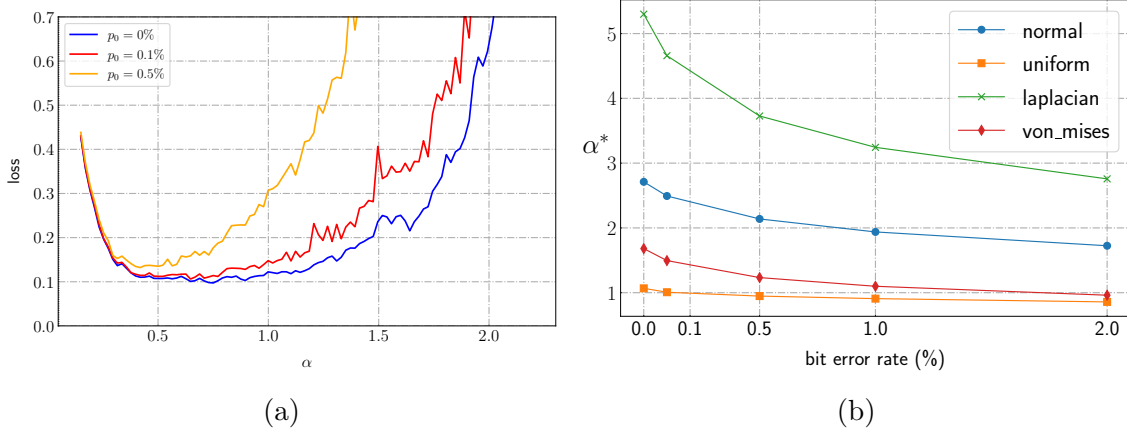


Figure 3.3 Optimal quantization range under bit error rate noise

Following a similar method in [12], in Figure 3.3b we demonstrate the α^* (y-axis) versus bit error rate (x-axis) for some commonly known weight distributions such as $\mathcal{N}(0, 1)$, $U(-1, 1)$, $\text{Laplacian}(0, 1)$, and $\text{Von mises}(0, 4)$, where the α^* is derived heuristically. Each point represents the optimal expected range for a different level of noise. As we can see, as the level of noise increases, the optimal quantization range becomes smaller, emphasizing the importance of our proposed approach to reduce the quantization range.

3.2 WCAT Method

In our approach, called Weight Clipping-Aware Training or WCAT, we focus on uniform symmetric signed quantization for the weights of convolution and fully-connected layers. Although asymmetric quantization has shown better robustness against bit error noise [11], symmetric quantization has the advantage of reducing the computational overhead of handling the zero-point offset during accumulation operations [61] and is more hardware-friendly.

3.2.1 Weight Clipping

We adopt uniform symmetric quantization and use the following clipping function that constrains the weight values, W , to be in a symmetric range controlled by $\alpha \in \mathbb{R}^+$:

$$\bar{W} = \text{Clip}(W, \alpha) = \begin{cases} -\alpha, & W < -\alpha, \\ W, & -\alpha \leq W \leq \alpha, \\ \alpha, & W > \alpha, \end{cases} \quad (3.8)$$

where $\text{Clip}(\cdot)$ denotes the clipping function and \bar{W} represents the clipped weights in the range $[-\alpha, \alpha]$. We perform such clipping in a layer-wise manner, particularly in every convolutional or fully-connected layer of the network.

We would like to emphasize that aggressive weight clipping has recently shown great success in improving DNN robustness with minimal training overhead [11, 59, 62]. However, aggressively clipping implies using a small α , which is likely to lead to performance degradation of the clean network. Hence, different α values will lead to different performance and robustness trade-offs.

3.2.2 Weight Quantization

On top of promoting robustness, weight clipping is also known to help decrease the quantization error by reducing the dynamic range of the weight distribution. After clipping, we then map the real-valued weights to a discrete grid of integers using a linear quantization scheme:

$$W_{\text{int}} = \left\lfloor \frac{\bar{W}}{\alpha / (2^{N_b-1} - 1)} \right\rfloor, \quad (3.9)$$

where N_b is the bit precision and $\lfloor \cdot \rfloor$ represents the round-to-nearest integer operator. The operator rounds the real-valued weights to the nearest integer, effectively quantizing the weights. To approximately recover the real-valued weights \bar{W} from their quantized representations W_{int} , we apply the following dequantization step:

$$\hat{W} = \frac{\alpha}{2^{N_b-1} - 1} \times W_{\text{int}}. \quad (3.10)$$

The intuition behind simply applying linear quantization rather than more complex quantization schemes, *e.g.* [36, 38, 63], is that we hypothesize that the quantization range is what most matters in terms of robustness, as recently indicated by previous work [11] and also observed in our experimental results. Nevertheless, we note that using more complex schemes may result in a better overall performance of the quantized, clean network.

3.2.3 Straight-through Gradient Estimation

After applying the clipping function (3.8), to update the weights during backpropagation, a possible approach [35] is to set the gradient to 1 for the weight values that lie inside the

Algorithm 1 Weight clipping-aware training (WCAT)

Require: bit error rate p_0 , weight decay λ , learning rate γ

```

for  $l = 1$  to  $L$  do
   $\alpha_l = c_1 \cdot \text{mean}(w_l) + c_2 \cdot \text{std}(w_l)$ 
end for
for  $t = 1$  to  $T$  do
  sample batch  $(x,y)$  from training data
  for  $l = 1$  to  $L$  do
     $\bar{w}_l = \text{Clip}(w_l, -\alpha_l, \alpha_l)$ 
     $w_l^{\text{int}} = \text{Quantize}(\bar{w}_l, \alpha_l)$ 
     $\epsilon \sim \text{Bernoulli}(p_0)$ 
     $\hat{w}_l^{\text{int}} = w_l^{\text{int}} \oplus \epsilon$ 
     $\tilde{w}_l = \text{De-quantize}(\hat{w}_l^{\text{int}}, \alpha_l)$ 
  end for
   $\Delta = \nabla \mathcal{L}(f_{\tilde{w}}(x), y) + \lambda \sum_{l=1}^L \alpha_l$ 
   $w^{t+1} = w^t - \gamma \Delta$ 
end for

```

clipping range and to 0 for the values outside the range.

$$\frac{\partial \hat{w}}{\partial w} = \begin{cases} 1, & \text{if } w \in [-\alpha, \alpha] \\ 0, & \text{otherwise} \end{cases} \quad (3.11)$$

Using this approach, the weights outside the range are not updated. Instead, the scaling factor is adjusted to bring the dequantized value closer to its optimal value. However, in our case, we want to decrease the scaling factor α to apply aggressive weight clipping and promote DNN robustness. Therefore, we propose to set the gradient to 1 for all weight values W , regardless of whether they lie inside or outside the quantization range. This is achieved using the straight-through gradient estimation technique, which allows the gradient to flow through the clipping operation without being affected by it. In the end, this provides an effective and efficient alternative to update the weights both inside and outside the quantization range without the need to increase α and compromise robustness.

3.2.4 Learned Quantization Range

In addition to updating the weights W that lie outside the quantization range, we also optimize α during training. This enables a better trade-off between quantization error reduction and dynamic range preservation and ultimately helps achieve higher robustness. First, we employ the same method proposed by Choi *et al.* [12] to initialize α based on the statistics of the initialized weights in each layer. This method has demonstrated a minimum mean

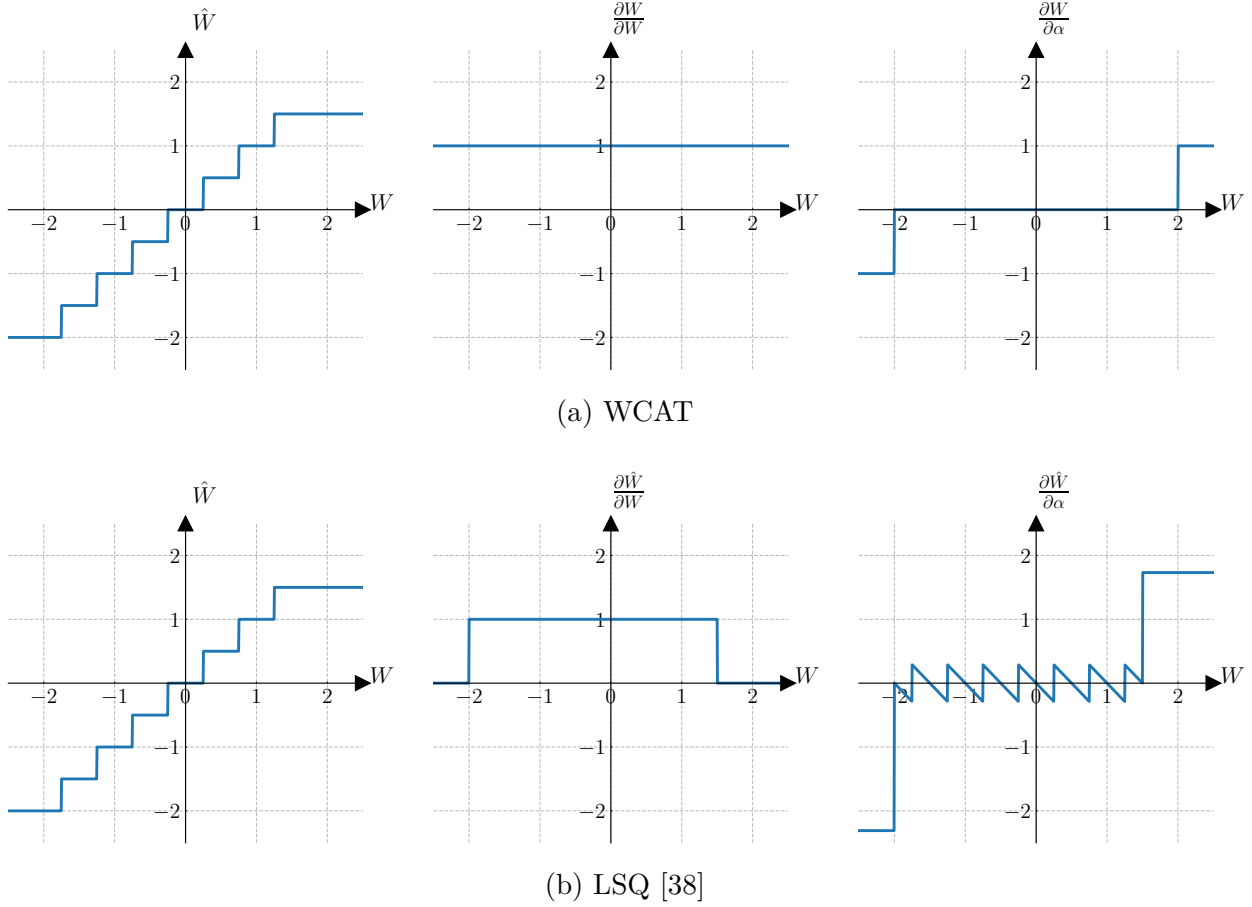


Figure 3.4 Comparison of the forward pass and gradient of the quantizer with respect to the quantization range and real-valued weights for WCAT and LSQ

squared error (MSE) between real and recovered weights. We define the gradient of α as follows:

$$\frac{\partial \hat{w}}{\partial \alpha} := \begin{cases} -1, & w < -\alpha, \\ 0, & -\alpha \leq w \leq \alpha, \\ 1, & w > \alpha, \end{cases} \quad (3.12)$$

with a different α being learned for each layer.

Importantly, since the gradients from the weights outside the quantization range get accumulated, we introduce a scaling factor applied to the gradient of the quantization range α during the learning process, similar to [38]. More specifically, we multiply the accumulated gradient of alpha, ∇_{α} , with $1/\sqrt{|W|}$, where $|W|$ represents the cardinality of the weight matrix, *i.e.* the number of weights in a given layer.

This scaling promotes a more controlled and slower update of the quantization range, while still allowing the network to learn the optimal α values. Figure 3.4 concludes the discussion in this section and illustrates the forward pass and gradient of the quantizer with respect to the quantization range and real-valued weights for our proposed WCAT method and LSQ [38]. As shown in the figure, LSQ provides a more complicated gradient and takes into account the impact of transitions between quantized states. On the other hand, our WCAT method, by setting the gradient to 1 for all weight values during backpropagation, allows for more aggressive weight clipping and promotes DNN robustness.

3.2.5 Weight Decay

To further promote a small α , we add an L2 regularization term to the α update at each iteration. The strength of the regularization is controlled by λ , the regularization coefficient, and the added loss term is expressed by $\frac{\lambda}{2}\alpha^2$. The trade-off between robustness and performance can be controlled by adjusting the value of λ . A large value of λ results in an increasingly smaller α during training and thus a more robust network since the quantization range will be reduced. On the other hand, a smaller value of λ will result in a larger α and a less robust network but with better noiseless performance.

3.2.6 Bit Flip Noise (BFN) Injections

To further improve DNN robustness at inference time, a popular approach is to simulate bit-flip errors during training [11, 64]. As discussed in Section 3.1.1, the desired objective is not differentiable and, therefore, needs to be approximated. A common approach for approximating the objective in Equation (3.2) is to use STE for the noise-injecting model, which passes the output gradients through the input gradients. Hence, by sampling ϵ from the noise distribution and using one noise realization per minibatch, assuming stochastic minibatch optimization, we can complete the forward pass by injecting the bit flip noise to the weights to compute the perturbed weights \tilde{w} and then compute the backward by setting $\frac{\partial \tilde{w}}{\partial w} = 1$.

CHAPTER 4 EXPERIMENTAL RESULTS ON WCAT

In this chapter, we present the experimental results of the proposed WCAT method. We begin by outlining our experimental setup in Section 4.1. Then, we investigate the impact of regularization on the weight distribution and robustness of the model in Section 4.2. Next, we examine the effect of gradient scaling (Section 4.3) and quantization range initialization (Section 4.4) on network resilience. We also compare various quantization methods in terms of their bit-error robustness against random (Section 4.5) and adversarial attacks (Section 4.6). Finally, we demonstrate how our proposed method can significantly reduce energy consumption at inference time while maintaining the desired level of reliability in Section 4.7.

4.1 Experimental Settings

Network Architectures and Datasets: Our study involves the analysis of object classification results obtained from three widely-used visual datasets, namely CIFAR10, CIFAR100 [65], and ImageNet [66]. Both CIFAR10 and CIFAR100 comprise 60,000 RGB images with dimensions of $32 \times 32 \times 3$. As per the standard approach, 50,000 examples were utilized for training and the remaining 10,000 for testing, with images evenly drawn from 10 and 100 classes, respectively. On the other hand, ImageNet is a more extensive dataset, consisting of 1.2 million training images across 1,000 classes, with input dimensions of $224 \times 224 \times 3$. Our work employs the same data augmentation techniques as those in [67]. For our experiments, we trained residual networks [67], specifically ResNet-20, ResNet-56, and ResNet-18, on CIFAR10 and CIFAR100, and ImageNet, respectively.

Training Setups: Each model on CIFAR10 and CIFAR100 was trained using stochastic gradient descent (SGD) [68] with a Nesterov momentum of 0.9, a weight decay of 5×10^{-4} , and a batch size of 128 for 300 epochs for 4 and 8 bit weight quantization. We used a cosine annealing learning rate scheduler, with an initial rate of 0.1.

To fine-tune our 8-bit network on ImageNet, we began with pre-trained weights and conducted a single epoch of additional training, using SGD with momentum and a learning rate of 10^{-4} . As for our 4-bit networks, we initialized the pre-trained weights and fine-tuned them for 110 epochs using SGD with momentum, with a batch size of 1024. We implemented exponential learning rate decay, starting from an initial rate of 0.0015 and decaying it to a final value of 10^{-6} .

Table 4.1 Robustness of ResNet-20 on CIFAR10 for Normal 4-bit Quantization with Different Levels of Bit-Flip Noise Injection During Training

Model	BFN Bit Error Rate	Test accuracy (%)		
		$p_o = 0\%$	$p_o = 0.1\%$	$p_o = 1\%$
ResNet-20 on CIFAR10	0.01%	92.41	91.93 \pm 2	83.42 \pm 3.4
	0.1%	91.97	91.6 \pm 2	82.89 \pm 3.2
	0.5%	91.91	91.4 \pm 2	84.81 \pm 2.3
	1%	Did not converge	-	-

Noise Injection Training: We conducted experiments applying various levels of bit-flip noise during training, as detailed in Section 3.2.6, in addition to the previous methods. Our results showed that a low bit error rate during training did not improve the model’s robustness during inference. Conversely, a high amount of noise completely destroyed the network’s performance during training and prevented it from converging. After testing different bit-flip probabilities p_o ranging from 10^{-4} to 10^{-2} , we found that a value of $p_o = 5 \times 10^{-3}$ increased the model’s robustness, with only a slight degradation in performance in the absence of noise. Table 4.1 illustrates the robustness of ResNet-20 on CIFAR10 for normal 4-bit quantization across different levels of noise injection during training. Furthermore, in the case of ImageNet, the network proved highly sensitive to minor perturbations, making it challenging to converge, even with a small amount of noise during training.

Evaluation Metrics: In order to assess the robustness of DNNs, we performed 50 iterations of bit-flip noise for each test example, and the classification result was obtained by averaging the outcomes. For evaluating the resilience of the DNNs against adversarial attacks, we modified the code from [4] and used the number of bit flips (N_{flip}) as the metric to measure BFA resistance, which corresponds to a reduction of top-1 accuracy to below 11%, 1.1%, and 0.11% for CIFAR10, CIFAR100, and ImageNet respectively. Since BFA requires a set of data to perform the attack, we selected 64 sample images from the test set as the default configuration, and we report the average N_{flip} over 10 different seeds.

Baselines: We conducted a comparison between our proposed method, WCAT, and three baselines, including linear quantization (Normal), per-layer aggressive weight clipping (PLClip) - which is one of the state-of-the-art methods for dealing with bit-flip errors - and LSQ, a state-of-the-art quantization method with learnable step size, using both 4-bit and 8-bit weight quantization. To determine the optimal standard clipping value for PLClip, we conducted a hyperparameter search within the range of 0.1 to 0.8 and we found the best standard clipping value for PLClip to be 0.3 for ResNet-20 on CIFAR10, 0.4 for ResNet-56 on CIFAR100, and 0.7 for ResNet-18 on ImageNet. It is worth noting that to generate a different clipping range

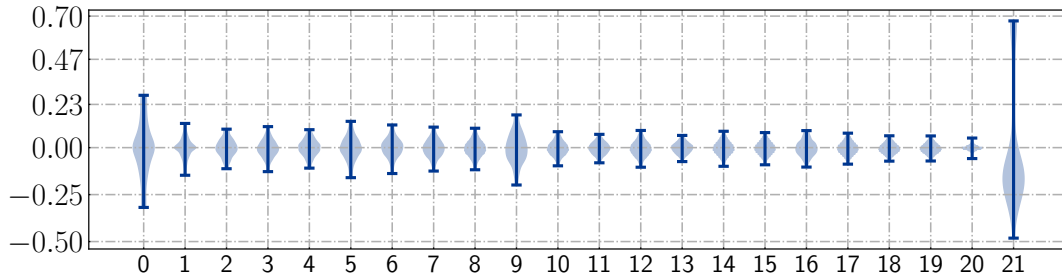
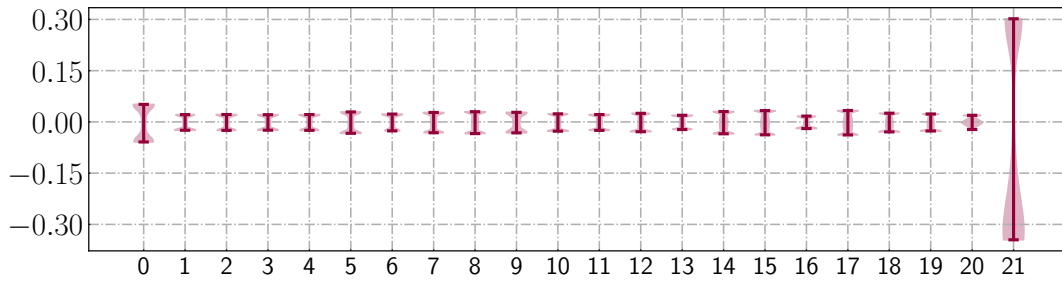
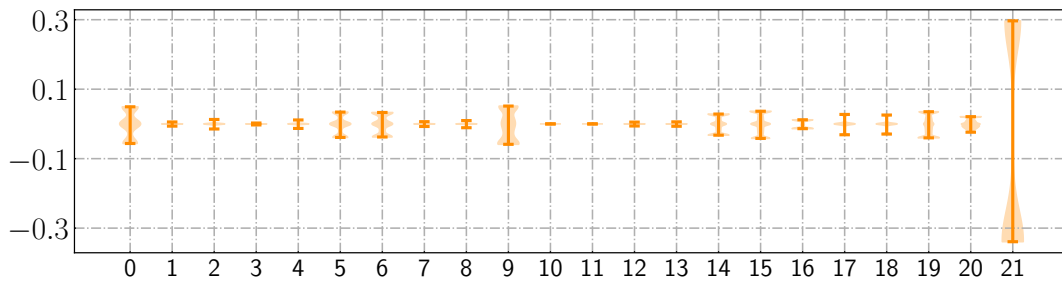
(a) Reg. coeff. $\lambda = 0.001$ (b) Reg. coeff. $\lambda = 0.01$ (c) Reg. coeff. $\lambda = 0.025$

Figure 4.1 Weight distribution of each layer of ResNet-20 trained with different regularization coefficients

Table 4.2 Robustness of ResNet-20 on CIFAR10 and ResNet-18 on Imagenet with 4 bit quantization for different gradient scales

Model	Grad scale	Reg. coeff. λ	Test accuracy (%)		
			$p_o = 0\%$	$p_o = 0.1\%$	$p_o = 1\%$
ResNet-20 on CIFAR10	1	0.0005	90.77	83.06 \pm 3.6	17.8 \pm 4.9
		0.001	91.18	86.49 \pm 2.3	23.02 \pm 6.4
		0.01	92.36	91.37 \pm .3	75.76 \pm 5.1
	$1/\sqrt{N}$	0.0005	92.48	91.55 \pm .35	77.72 \pm 5.4
		0.001	92.41	91.93\pm.2	83.42 \pm 3.4
		0.01	91.88	91.69 \pm .1	88.62\pm1.2
ResNet-18 on IMAGENET	1	0.001	68.59	31.46 \pm 5.9	0.1 \pm .01
		0.01	69.87	51.73 \pm 4.4	0.2 \pm .07
		0.5	69.91	66.36 \pm 1.1	22.84 \pm 5.2
	$1/\sqrt{N}$	0.001	70.07	63.26 \pm 2.4	9.1 \pm 4.9
		0.01	69.28	66.96\pm.6	36.83 \pm 6.4
		0.5	66.56	65.11 \pm .5	47.26\pm3.4

per layer, we multiplied the clipping value by the ratio between the maximum absolute weight of the given layer and the maximum absolute weight across the entire model. However, it’s crucial to mention that PLClip is highly sensitive to the clipping hyperparameter selection, and finding the best value is a time-consuming process. Additionally, computing the clipping range for each layer using PLClip after each iteration significantly increases the training time.

4.2 Regularization

In this study, we investigate the impact of L2 regularization of the quantization ranges on the robustness and performance of different models in the presence of noise. To this end, we vary the L2 regularization coefficient and evaluate the resulting models under different noise realizations. First, we observed that increasing the L2 regularization reduces the range of weights as illustrated in Figure 4.1, where the index of each layer is shown along the x-axis. Also, the results presented in Table 4.2 show that increasing the L2 regularization tends to improve robustness in noisy environments, at the expense of decreased performance in the noiseless setting.

Specifically, we find that increasing the regularization strength to force the model to have a lower quantization range can improve robustness, as evidenced by the results for $\lambda = 0.5$ for the ImageNet dataset. However, this comes at a significant cost in terms of performance in the absence of noise. In these scenarios, it may be more beneficial to train the model with bit-flip noise to improve its robustness against noise, as we showed in Figure 4.2.

Table 4.3 Comparison of robustness for ResNet-18 on ImageNet using different quantization range initialization

Model Bit Width	Initialization Method	Test Accuracy	
		$p_o = 0\%$	$p_o = 0.05\%$
4-bit	Max_Abs	69.55	66.88
	SAWB	69.28	68.33
8-bit	Max_Abs	70.09	42.23
	SAWB	70.04	52.49

4.3 Gradient Scaling

Table 4.2, shows the result of an experiment to investigate the impact of scaling gradient on the performance and robustness of the models. Our findings indicate that with low regularization, the scaling gradient not only improves the performance of the noiseless model but also enhances the robustness of the noisy regime for both models. Interestingly, the addition of an L2 regularizer further improves the performance of the models with a gradient scale of 1. As discussed in section 3.2.4, updating the quantization range involves accumulating all the gradients of weights at each layer with respect to α , which can lead to high gradients and instability without gradient scaling. The addition of a regularizer helps to restrict the rapid change when we don't use gradient scaling. Ultimately, we observe that the models trained with scaled gradients achieve the best performance for different levels of noise. These results suggest that scaling gradient besides increasing regularization can significantly improve the robustness of the model against bit errors, especially in noisy environments while enhancing the overall performance of the model.

4.4 Quantization Range Initialization

In Section 3.2.4, we discussed our approach to initializing the quantization range. We use the SAWB method, which approximates the optimal quantization range by analyzing the statistics of the weights on each layer. In contrast, a more simplistic method is to initialize the range based solely on the maximum absolute value of the weights, which we refer to as the Max_Abs method.

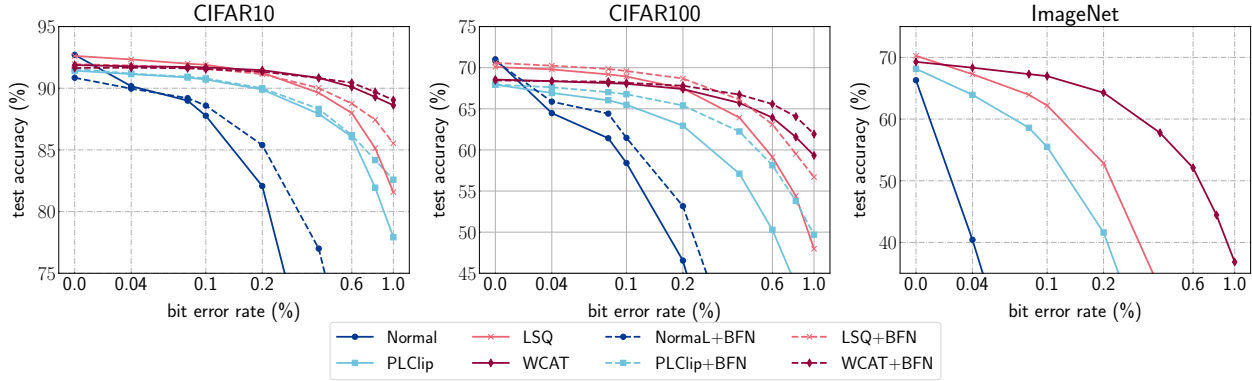


Figure 4.2 Robustness of different quantization methods on ResNets with 4-bit weight quantization

To evaluate the effect of these initialization methods, we conducted experiments on ResNet-18 finetuned on ImageNet and presented the results in Table 4.3. As demonstrated, the Max_Abs method produces better clean performance for both 8-bit and 4-bit quantization. However, this comes at a cost to the robustness of the network after finetuning, as the larger initial range of the weights can lead to increased sensitivity to perturbations.

4.5 Random Attack

In Figure 4.2, we compare the bit-error robustness of different methods for 4-bit weight quantization on ResNet-20 trained on CIFAR10, ResNet-56 trained on CIFAR100, and ResNet-18 on ImageNet. Overall, WCAT appears to be superior to other methods, as it maintains high performance across all tested noisy regimes on all models and datasets.

Our observations demonstrate how a carefully designed quantization scheme can improve robustness while maintaining better accuracy on noiseless models. Across all datasets, LSQ outperforms PLClip, with LSQ achieving the best accuracy on noiseless models. In CIFAR10, LSQ and WCAT perform similarly under low noise conditions, but WCAT shows significantly better performance under high noisy regimes.

We observe significant improvement in ImageNet results. Even with a small amount of noise (e.g., 10^{-4}), the accuracy of normal quantization drops dramatically. Although LSQ and PLClip show better robustness than normal quantization, WCAT outperforms all methods under highly noisy regimes. For instance, WCAT can tolerate a 0.2% bit error rate while keeping test accuracy above 65% whereas LSQ and PLClip degrade accuracy to 53% and 41%, respectively, for this amount of noise.

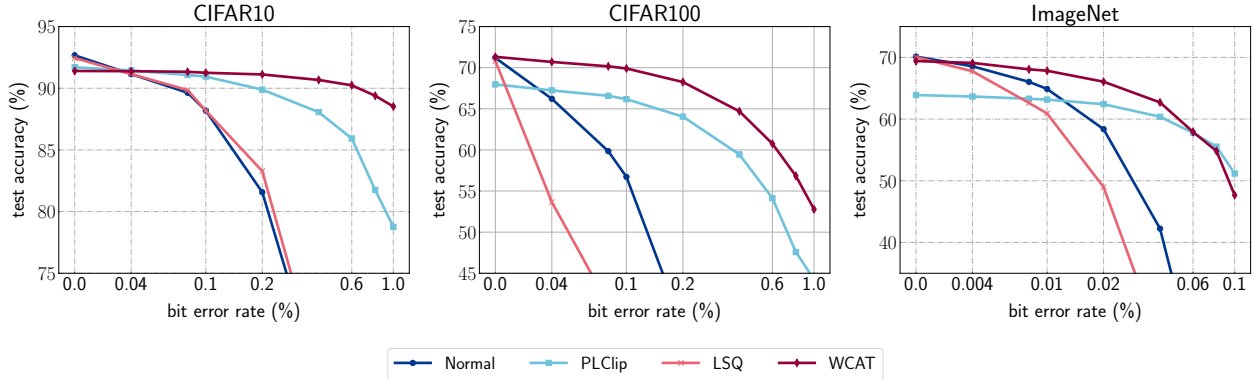


Figure 4.3 Robustness of different quantization methods on ResNets with 8-bit weight quantization

In all methods, bit-flip noise (BFN) injection training tends to improve robustness, especially under highly noisy regimes. Although BFN significantly sacrifices test accuracy for normal quantization under noiseless conditions, this performance loss is negligible in PLClip and WCAT.

The results of our experiments on 8-bit quantization are presented in Figure 4.3. LSQ exhibits the best accuracy on clean weights for all models. However, when faced with even a minor bit-flip noise, LSQ’s accuracy drops significantly. In contrast, WCAT demonstrates remarkable robustness across all datasets. Specifically, on ImageNet, the PLClip method prioritizes accuracy in highly noisy regimes at the expense of performance on clean weights. Our findings emphasize the robustness of the proposed method on various models and datasets for 4-bit and 8-bit quantization. Additionally, we observed that ImageNet on 8-bit is more sensitive to bit-flip noise than in 4-bit, and could not tolerate high levels of noise as well. Furthermore, our results suggest that this method can generalize well across different conditions, enhancing robustness while maintaining high performance on noiseless models.

4.6 Adversarial Attack

Figure 4.4 showcases the robustness of different methods against adversarial bit flip attacks for all models and for both 4-bit and 8-bit quantization. For each dataset, we show the average number of bit flips (N_{flip}) required to destroy network performance and reduce accuracy to random guess over 10 different seeds.

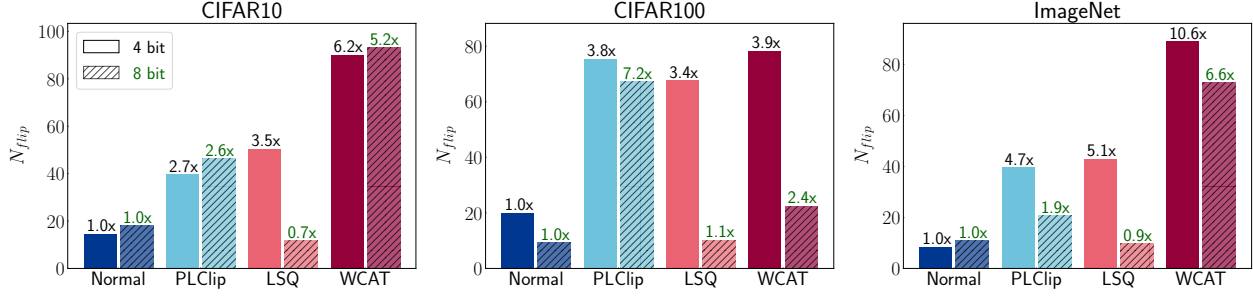


Figure 4.4 Impact of adversarial bit flip attacks on network performance using ResNets with 4 and 8-bit weight quantization

Notably, WCAT not only enhances robustness against random noise but also outperforms all methods in all models against adversarial attacks under 4-bit quantization. This indicates that the benefits of our proposed method extend beyond improving network robustness in the presence of random noise.

Moreover, WCAT’s superiority is apparent in ImageNet results, where it requires $10.6\times$ more bit flips to cause network performance degradation when compared to other methods. This demonstrates the effectiveness of WCAT in protecting networks from adversarial attacks in addition to random noise.

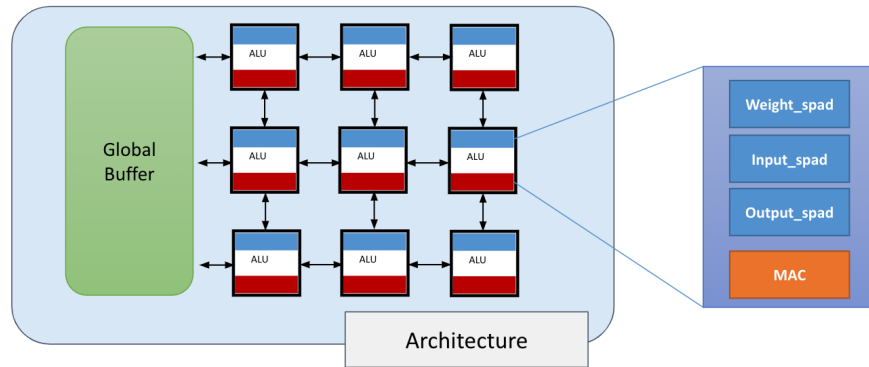


Figure 4.5 The modified Eyeriss-like architecture

4.7 Energy Consumption

We model the energy consumption of unreliable SRAMs by applying quadratic regression on the data from [51, Fig. 11], which is based on measurements on fabricated chips. For each model, we use Accelergy [46] to estimate the energy consumption on the Eyeriss accelerator [18], thus including both memory-access and operation energy. We modified the Eyeriss

architecture to have only on-chip memory, implemented with SRAM, for all data types as shown in Figure 4.5. We assume only the DNN weights to be stored unreliably, with the activations and partial sums being stored in a reliable SRAM. We also assume that batch normalization layers are fused with the convolutional layers. The energy consumption at nominal voltage (0.8 V) for one image (batch size of 1) is $118.37 \mu\text{J}$, $285.96 \mu\text{J}$, $390.54 \mu\text{J}$, and $495.12 \mu\text{J}$, for ResNet-20, ResNet-32, ResNet-44, and ResNet-56, respectively.

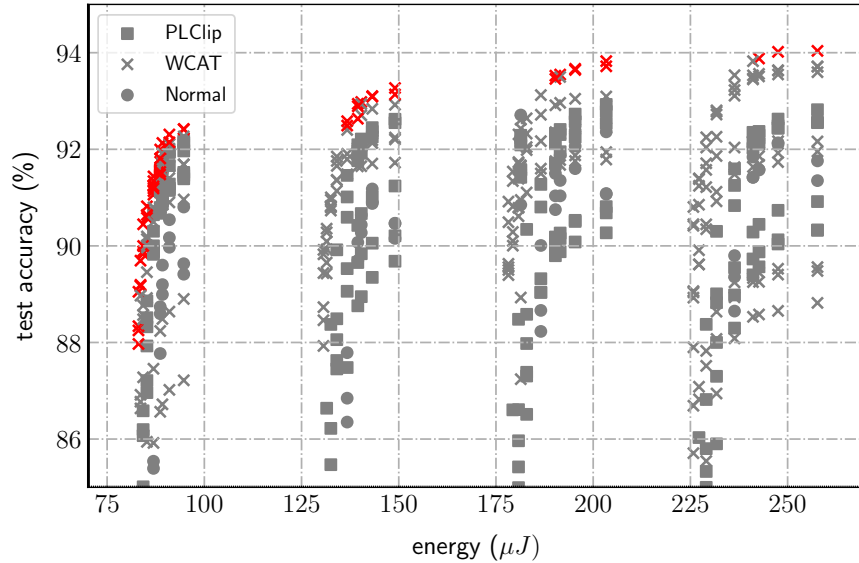


Figure 4.6 Classification accuracy in terms of energy per inference on CIFAR-10

In Figure 4.6, we illustrate the accuracy versus energy consumption of the aforementioned ResNet models on CIFAR-10 using the different methods and levels of noise. All methods are repeated with and without BFN. We show the points in the Pareto frontier in red color, which represents the best trade-off between energy consumption and accuracy. We observe that WCAT always achieves the lowest energy consumption for a given accuracy level. Specifically, for an accuracy level of 93%, our proposed method achieves an energy consumption of $140.36 \mu\text{J}$, while the closest competitor, PLClip+BFN, consumes $203.35 \mu\text{J}$.

CHAPTER 5 ROBUST QUANTIZATION WITH BINARY GUMBEL

As discussed in Section 3.1.1, backpropagation through XOR operation is one of the challenges in training a robust neural network against bit-flip error. Even STE-based gradient estimation, introduced in Section 3.2.6, is not a precise way to approximate the gradient of the XOR operation. In fact, it has been theoretically demonstrated that only for proper choice of STE, such as thresholding function (e.g. quantizer), the negative expected gradients of STE correspond to descent directions for minimizing the loss function [69].

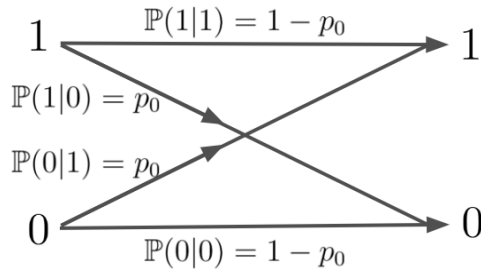


Figure 5.1 Bit flip noise model

A possible solution to accurately estimate the gradient of the XOR operation is by modeling each bit in the neural network as a random variable that follows a Bernoulli distribution with a specific parameter. By incorporating the noise model shown in Figure 5.1, we can accurately determine the distribution after injecting bit-flip noise. This approach provides a valuable opportunity to estimate the gradient of the XOR operation precisely by sampling from the perturbed probability distribution in the forward pass and updating these probabilities through the backward pass.

5.1 Background

We can model each bit in the network as a random variable drawn from a Bernoulli distribution with a specific parameter π_i , where i denotes the i -th bit. The probability mass function of the Bernoulli distribution is given by $q_{\pi_i}(b_i) = \pi_i^{b_i}(1 - \pi_i)^{1-b_i}$, where b_i is the value of the i -th bit. Using this probability distribution, we can define the objective as:

$$l = \mathbb{E}_{B \sim \text{Bernoulli}(\pi)} [\mathcal{L}(f_B(x), y)], \quad (5.1)$$

where $f_B(x)$ is the neural network with the set of bits $B = \{b_0, b_1, \dots, b_n\}$, and (x, y) is

a training input and its corresponding label from the training set. We can compute the gradient of the loss function with respect to π :

$$\begin{aligned}\nabla_{\pi} &= \frac{\partial}{\partial \pi} \mathbb{E}_{B \sim \text{Bernoulli}(\pi)} [\mathcal{L}(f_B(x), y)] \\ &= \sum_{b_1=0}^1 \cdots \sum_{b_n=0}^1 \mathcal{L}(f_B(x), y) \frac{\partial}{\partial \pi} \prod_{j=0}^n q_{\pi_j}(b_j),\end{aligned}\tag{5.2}$$

but the number of terms in the summation grows exponentially in the number of bits, making the exact gradient computationally intractable. Furthermore, the summation is not in the form of an expectation, which means we cannot estimate it using Monte Carlo sampling. This phenomenon is referred to as *distributional dependency* in the literature.

5.1.1 Gumbel-Max Trick

One way to approximate the exact gradient is by using the reparameterization trick. This involves changing the source of randomness in the expectation to allow for Monte Carlo gradient estimation. We can accomplish this by defining a function $g(\pi, \epsilon)$ that generates samples similar to B by sampling from $\epsilon \sim \eta$. Then, we can write:

$$\begin{aligned}\nabla_{\pi} &= \frac{\partial}{\partial \pi} \mathbb{E}_{B \sim \text{Bernoulli}(\pi)} [\mathcal{L}(f_B(x), y)] \\ &= \frac{\partial}{\partial \pi} \mathbb{E}_{\epsilon \sim \eta} [\mathcal{L}(f_{g(\pi, \epsilon)}(x), y)] \\ &\simeq \frac{1}{M} \sum_i^M \frac{\partial}{\partial \pi} \mathcal{L}(f_{g(\pi, \epsilon_i)}(x), y),\end{aligned}\tag{5.3}$$

where $\epsilon_i \sim \eta$ and M is the number of Monte Carlo samples used to estimate the gradient. To obtain samples similar to b_i , we used the Gumbel-Max trick [70] which provides a straightforward way to draw samples from a categorical distribution. So we can define the auxiliary function g as follow:

$$g(\pi, \epsilon) = \text{onehot} \left(\arg \max_k [\epsilon_k + \log \pi_k] \right) \quad \text{where } k = 1, \dots, K.\tag{5.4}$$

In 5.4 ϵ_k are i.i.d samples drawn from $Gumbel(0, 1)$ and π is the class probabilities with K categories. The Gumbel-Softmax method uses the softmax function as a continuous, differentiable approximation to argmax for the backward, known as the straight-through

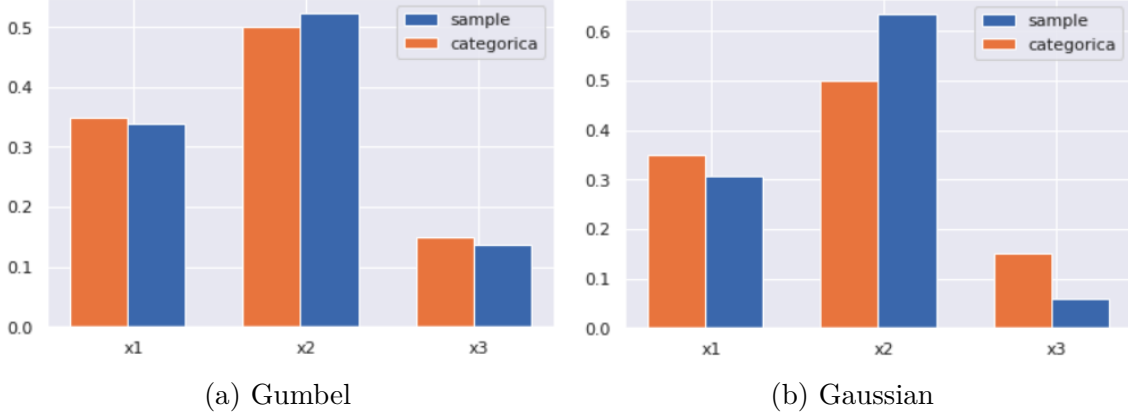


Figure 5.2 Comparison of Gumbel and Normal noise for generating samples close to categorical distributions

Gumbel-Softmax, as follows:

$$z_k = \frac{\exp((\log(\pi_k) + \epsilon_k) / \tau)}{\sum_{j=1}^K \exp((\log(\pi_j) + \epsilon_j) / \tau)}, \quad (5.5)$$

where τ is a temperature parameter that controls the smoothness of the approximation. The Gumbel noise is a carefully chosen noise for the max operation that can generate samples close to categorical distributions. We demonstrate this in Figure 5.2, where we show the average occurrence of samples for Gumbel and Gaussian noise. As we can see, the samples generated using normal noise are not close to the true class probability, highlighting the importance of using the Gumbel distribution.

5.2 Method

In this section, we present a detailed overview of our proposed method. We begin by introducing the binary Gumbel algorithm and provide a simple implementation for generating binary samples from this distribution. We then extend this approach by using perturbed logits to approximate the gradients of bit flip noise and train a more robust network. Finally, we employ the Rao-Blackwellization technique to further reduce the variance of gradient approximation and improve training efficiency.

Algorithm 2 Binay Gumbel Pseudocode, PyTorch-like

```

def binary_gumbel(logits, tau=1.):
    U = torch.rand(logits.shape)
    L = torch.log(U) - torch.log(1-U)
    hard_sample = torch.heaviside((logits+L), torch.tensor(0.)).short()
    soft_sample = torch.sigmoid((logits+L) /tau)
    sample = hard_sample + soft_sample - soft_sample.detach()
    return sample

```

5.2.1 Binary Gumbel

In our case, where we have only two categories, we can simplify the sampling process as follow. Let us set the probability vector to $\boldsymbol{\pi} = [\pi_i, 1 - \pi_i]$, then the probability of selecting the first category can be written as $\mathbb{P}(z[0] = 1) = \pi_i$. By using the Gumbel-Max trick, we have:

$$\begin{aligned}
 \mathbb{P}(z[0] = 1) &= \mathbb{P}(\epsilon_1 + \log \pi_i > \epsilon_2 + \log(1 - \pi_i)) \\
 &= \mathbb{P}\left(\epsilon_1 - \epsilon_2 + \log\left(\frac{\pi_i}{1 - \pi_i}\right) > 0\right)
 \end{aligned}
 \tag{5.6}$$

where, ϵ_k are i.i.d samples drawn from $Gumbel(0, 1)$. We know the difference between two Gumbels follows a Logistic distribution $L := \epsilon_1 - \epsilon_2 \sim Logistic$, which can be sampled in the following way,

$$L := \log U - \log(1 - U) \quad \text{where } U \sim Uniform(0, 1) \tag{5.7}$$

and by setting the logit ζ_i as log odd ratio, $\zeta_i := \log\left(\frac{\pi_i}{1 - \pi_i}\right)$, we can generate samples as follow:

$$b_i = \mathbb{H}(\zeta_i + L) \quad \text{where } L \sim Logistic \tag{5.8}$$

where the \mathbb{H} is the unit step function. For the backward path, we can consider the continuous relaxation of the unit step function as follow:

$$\hat{b}_i = \sigma\left(\frac{\zeta_i + L}{\tau}\right) \quad \text{where } L \sim Logistic \tag{5.9}$$

where $\hat{b}_i \in (0, 1)$, and the $\tau \in (0, \infty)$ is the temperature of the sigmoid function. The pseudo-code of straight-through Gumbel-Softmax for the binary case is in Algorithm 2.

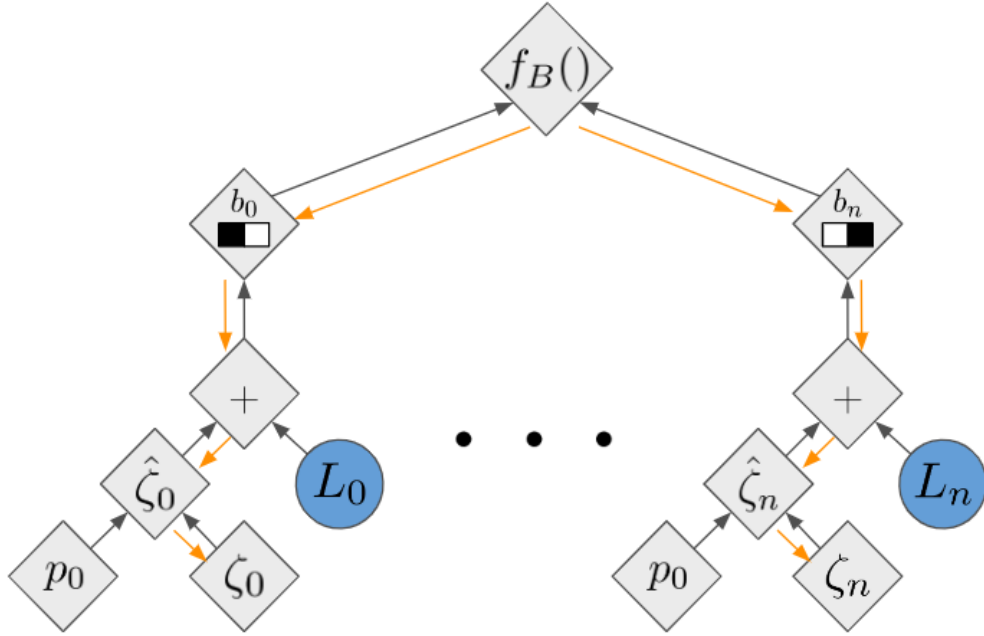


Figure 5.3 Stochastic computation graph for the proposed method

5.2.2 Using Perturbed Logits to Imitate Bit-Flip Noise Injection

We illustrate our method using a stochastic computation graph, as shown in Figure 5.3. The black arrows demonstrate the forward pass and the orange arrows show the backpropagation. To incorporate noise during training, we introduce a bit error rate p_0 and perturb the logits ζ_i with this probability as follow:

$$\hat{\zeta}_i = \log \left(\frac{\hat{\pi}_i}{1 - \hat{\pi}_i} \right) \quad (5.10)$$

where, $\hat{\pi}_i = \sigma(\zeta_i)(1 - p_0) + (1 - \sigma(\zeta_i))p_0$. We then employ the binary Gumbel trick to sample binary values from the perturbed logits. These binary samples are then de-quantized and used to compute the rest of the neural network.

5.2.3 Bias-variance tradeoff in gradient estimation

The Gumbel-Softmax estimator can be biased, and decreasing the temperature τ can reduce this bias by generating samples closer to the true categorical distribution. However, reducing the temperature also increases the variance of the gradient estimation. This tradeoff is illustrated in Figure 5.4 with a simple example where we want to compute the gradient

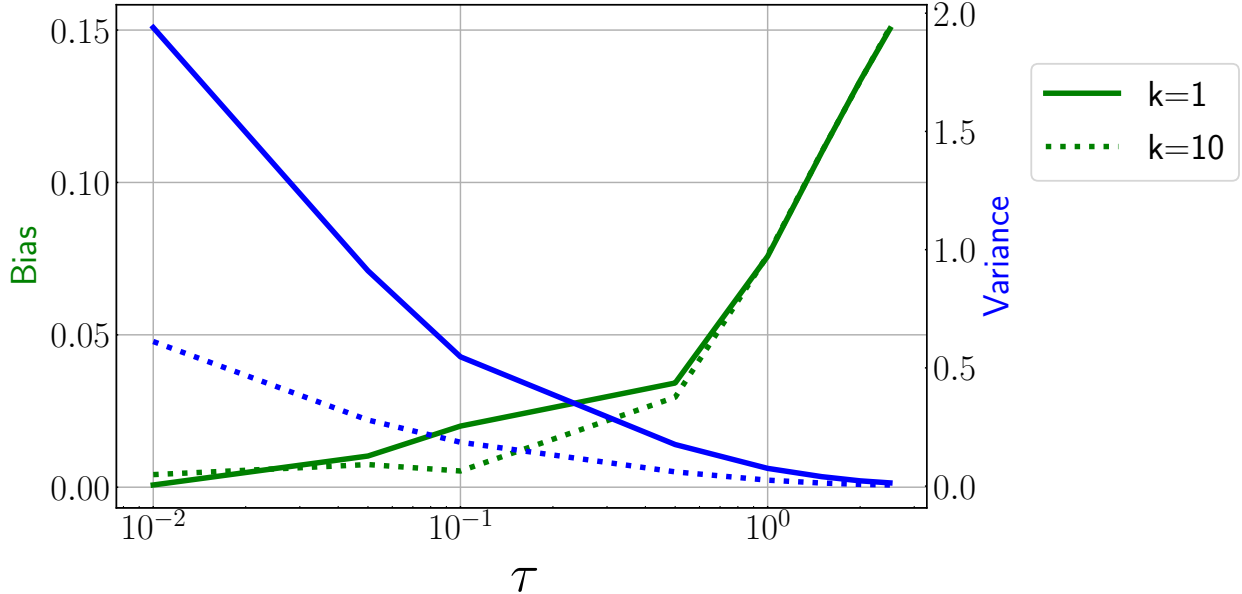


Figure 5.4 Tradeoff between bias and variance of Gumbel-Softmax estimator with respect to temperature τ

$\nabla_{\zeta} := \mathbb{E}_x[f(x)]$, where $f(\cdot)$ is the identity function. High variance in gradient estimation can make optimization more difficult. To further reduce the variance of gradient estimation, we employ the Rao-Blackwellization technique with the Gumbel-Softmax estimator [71]. This method marginalizes the Straight-Through Gumbel-Softmax estimator (ST-GS) and as a result, reduces overall variance. Let $z \sim \text{Categorical}(\zeta)$, then the Gumbel-Rao estimator (GR) is defined as:

$$\nabla_{GR} = \mathbb{E}[\nabla_{STGS}|z] \simeq \frac{1}{K} \sum_{i=1}^K \nabla_{STGS}(z_i) \quad (5.11)$$

where K is the number of Monte Carlo samples. It has been proven that increasing K can decrease the variance, as shown in Figure 5.4. The GR estimator can be implemented efficiently without increasing the number of function evaluations.

5.3 Experimental results

We evaluated our method on the MNIST dataset using Lenet5 with 2-bit weight quantization, setting the temperature to $\tau = 1$. To study the impact of noise and the number of Monte-Carlo samples on the Gumbel-Rao estimator, we conducted experiments whose results are shown in Figure 5.5. As seen in Figure 5.5a, increasing p_0 led to a slight decrease in accuracy

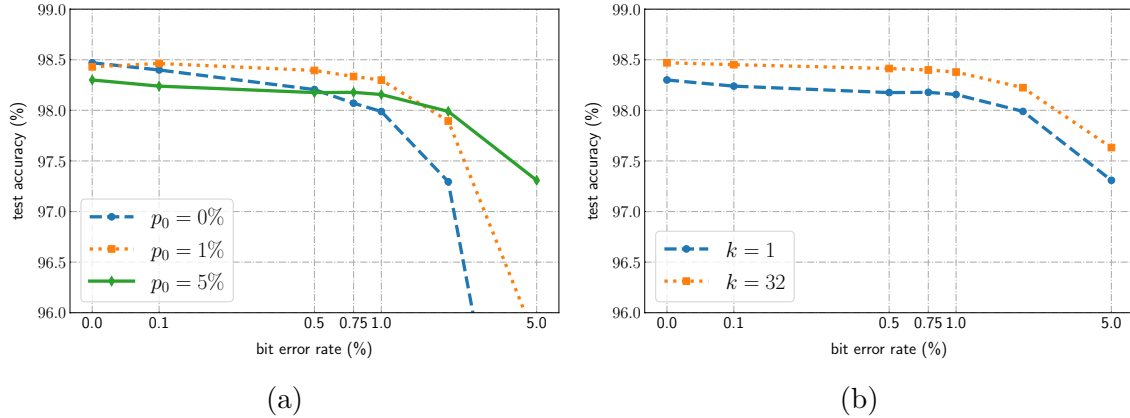


Figure 5.5 Comparison of the accuracy and robustness of Lenet5 with 2-bit weight quantization

for the noiseless network but provided greater robustness against bit-flip noise. Moreover, Figure 5.5b shows that increasing the number of Monte-Carlo samples improves the accuracy of the noiseless model and enhanced robustness against bit-flip noise.

We compared our proposed method with different quantization methods and our experimental results, as shown in Figure 5.6, demonstrate that our method has higher accuracy in both noiseless and highly noisy regimes.

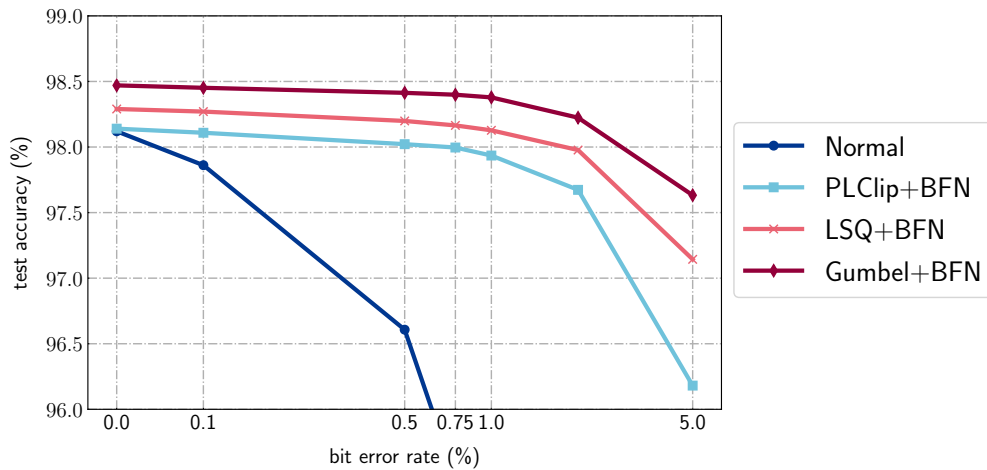


Figure 5.6 Robustness of different quantization methods on Lenet5 with 2-bit weight quantization

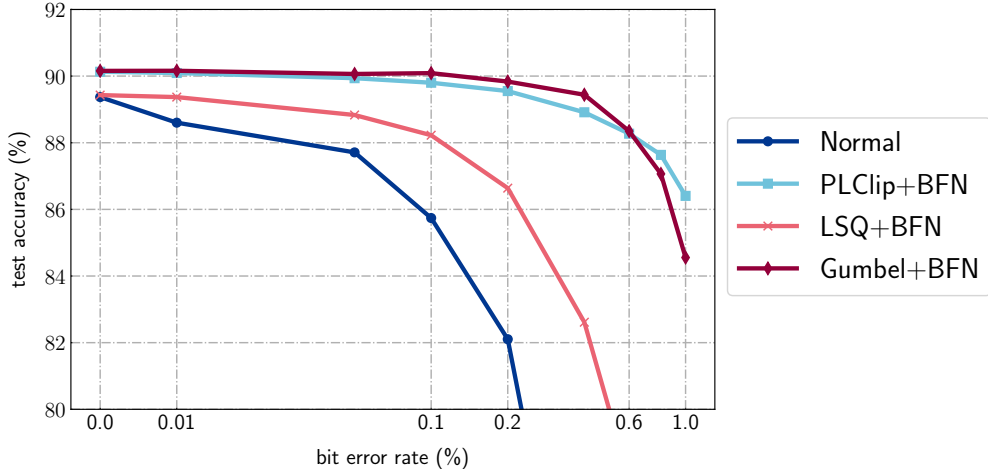


Figure 5.7 Comparison of robustness for different quantization methods on CIFAR10 using ResNet20 with 2-bit quantization

The proposed method was also evaluated on the CIFAR10 dataset, using ResNet20 with 2-bit quantization. However, training with the proposed method presented certain limitations and convergence issues. As a solution, we utilized a pre-trained model with full precision weights and initialized the logits in a way that the expected value of sampled weights is close to the full precision weights. To reduce the variance estimation, we excluded the first and last layers of the model.

To optimize the proposed method, we conducted a hyperparameter search and determined the optimal temperature to be $\tau = 3$. Additionally, we employed a linear scheduler to anneal the temperature to 0.5 during the training process. We fine-tuned the model for 100 epochs, utilizing an Adam optimizer with a learning rate of 0.1 for the logits and 0.01 for the other model parameters. We also used a multi-step learning scheduler and decreased the learning rate by a factor of 0.1 at epochs 60 and 90. Furthermore, we inject noise during training using perturbed logits with $p_0 = 0.5\%$.

Figure 5.7 illustrates that the proposed Gumbel-binary estimation method showed only a marginal improvement in robustness compared to the PLClip method, and was found to be inferior in the high-noise regime. This suggests that there are limitations when scaling up our method and it may encounter convergence issues when dealing with larger models. Further research is needed to address these limitations and develop a lower variance estimator that can handle more complex datasets.

CHAPTER 6 CONCLUSION

In this work, we proposed two novel approaches to enhance the robustness of DNNs against bit-flip errors during inference. Our first method incorporates a learning mechanism to optimize the quantization range of different layers during training, achieving a delicate balance between error resiliency and performance in noiseless conditions. The second method provides a novel probabilistic solution for training in presence of bit-flip noise.

Our experimental results indicate that the proposed methods outperform existing robust quantization techniques. They exhibit higher accuracy levels and better defense against both random and adversarial attacks. WCAT consistently provides a practical solution to improve the resilience of DNNs against bit-flip errors across different architectures and datasets. Additionally, it significantly reduces energy consumption at inference time while remaining reliable.

While our proposed methods have shown promising results, there are still limitations to be addressed. First, in the WCAT method, we did not propose a constraint on learning the quantization range during training to be positive. Therefore, when we increase the regularization, it may lead to a negative quantization range during the optimization step. This limitation could be addressed by introducing a constraint on the quantization range to be positive, which could improve the optimization process stability.

Also, the effectiveness of the Gumbel-binary estimation method may be limited when scaling up to larger models. Additionally, training with this method may present convergence issues. Hyperparameter choices such as temperature during estimation greatly influence performance. To overcome these limitations, future research could focus on developing lower variance estimators or exploring other probabilistic methods to improve DNN robustness against bit-flip errors.

In future work, it would be worthwhile to evaluate the WCAT method on other applications such as natural language processing tasks, which are commonly implemented using transformer-based models. Considering the large size and complexity of these networks, robustness against bit-flip errors may present significant challenges. Therefore, exploring the potential of our methods on such models would be a promising avenue for future research.

REFERENCES

- [1] P. Bose *et al.*, “Secure and resilient SoCs for autonomous vehicles,” *Proceedings of the 3rd International Workshop on Domain Specific System Architecture (DOSSA)*, pp. 1–6, 2021.
- [2] C. Gongye *et al.*, “New passive and active attacks on deep neural networks in medical applications,” in *Proceedings of the 39th international conference on computer-aided design*, 2020, pp. 1–9.
- [3] C. Qian *et al.*, “A survey of bit-flip attacks on deep neural network and corresponding defense methods,” *Electronics*, 2023.
- [4] A. S. Rakin, Z. He, and D. Fan, “Bit-flip attack: Crushing neural network with progressive bit search,” in *IEEE/CVF International Conference on Computer Vision*, 2019.
- [5] Y. Kim *et al.*, “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors,” in *ACM/IEEE International Symposium on Computer Architecture*, 2014.
- [6] V. Sze *et al.*, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
- [7] L. Liu *et al.*, “Generating robust dnn with resistance to bit-flip based adversarial weight attack,” *IEEE Transactions on Computers*, vol. 72, no. 2, pp. 401–413, 2023.
- [8] Y. Guo *et al.*, “ModelShield: A generic and portable framework extension for defending bit-flip based adversarial weight attacks,” in *IEEE International Conference on Computer Design*, 2021.
- [9] Z. He *et al.*, “Defending and harnessing the bit-flip based adversarial weight attack,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.
- [10] A. M. Buldu *et al.*, “Mbet: Resilience improvement method for dnns,” *2022 IEEE International Conference On Artificial Intelligence Testing (AITest)*, 2022.
- [11] D. Stutz *et al.*, “Bit error robustness for energy-efficient DNN accelerators,” *Machine Learning and Systems*, 2021.
- [12] J. Choi *et al.*, “Accurate and efficient 2-bit quantized neural networks,” *Machine Learning and Systems*, 2019.

- [13] M. Mathieu, M. Henaff, and Y. LeCun, “Fast training of convolutional networks through FFTs,” *arXiv preprint arXiv:1312.5851*, 2013.
- [14] A. Lavin and S. Gray, “Fast algorithms for convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4013–4021.
- [15] R. Krashinsky *et al.*, “Nvidia ampere architecture in-depth,” 2020. [Online]. Available: <https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/>
- [16] M. Sabbagh, Y. Fei, and D. Kaeli, “A novel gpu overdrive fault attack,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.
- [17] A. Ardakani *et al.*, “An architecture to accelerate convolution in deep neural networks,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 4, pp. 1349–1362, 2017.
- [18] Y.-H. Chen *et al.*, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.
- [19] S. Han *et al.*, “Learning both weights and connections for efficient neural network,” *Advances in neural information processing systems*, vol. 28, 2015.
- [20] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *International Conference on Learning Representations*, 2016.
- [21] P. Molchanov *et al.*, “Importance estimation for neural network pruning,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 264–11 272.
- [22] A. N. Gomez *et al.*, “Learning sparse networks using targeted dropout,” *arXiv preprint arXiv:1905.13678*, 2019.
- [23] Z. Li *et al.*, “Can pruning improve certified robustness of neural networks?” *Transactions on Machine Learning Research*, 2023.
- [24] S. Ye *et al.*, “Adversarial robustness vs. model compression, or both?” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 111–120.

- [25] Y. Guo *et al.*, “Sparse DNNs with improved adversarial robustness,” *Advances in neural information processing systems*, vol. 31, 2018.
- [26] A. S. Rakin *et al.*, “Robust sparse regularization: Defending adversarial attacks via regularized sparse network,” in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 125–130.
- [27] “Nvidia A100 GPUs power the modern data center.” [Online]. Available: <https://www.nvidia.com/en-us/data-center/a100/>
- [28] E. Frantar *et al.*, “Gptq: Accurate post-training quantization for generative pre-trained transformers,” *arXiv preprint arXiv:2210.17323*, 2022.
- [29] Z. Tu *et al.*, “Adabin: Improving binary neural networks with adaptive binary sets,” in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI*. Springer, 2022, pp. 379–395.
- [30] F. Li, B. Zhang, and B. Liu, “Ternary weight networks,” *arXiv preprint arXiv:1605.04711*, 2016.
- [31] M. Rastegari *et al.*, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV*. Springer, 2016, pp. 525–542.
- [32] R. Banner, Y. Nahshan, and D. Soudry, “Post training 4-bit quantization of convolutional networks for rapid-deployment,” *Advances in Neural Information Processing Systems*, 2019.
- [33] R. Zhao *et al.*, “Improving neural network quantization without retraining using outlier channel splitting,” in *International Conference on Machine Learning*, 2019.
- [34] J. Choi *et al.*, “PACT: Parameterized clipping activation for quantized neural networks,” *arXiv preprint arXiv:1805.06085*, 2018.
- [35] C. Baskin *et al.*, “Nice: Noise injection and clamping estimation for neural network quantization,” *Mathematics*, 2021.
- [36] A. Goncharenko *et al.*, “Trainable thresholds for neural network quantization,” in *International Work-Conference on Artificial Neural Networks*, 2019.

- [37] K. Yamamoto, “Learnable companding quantization for accurate low-bit neural networks,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 5029–5038.
- [38] S. K. Esser *et al.*, “Learned step size quantization,” in *International Conference on Learning Representations*, 2020.
- [39] S. Jung *et al.*, “Learning to quantize deep networks by optimizing quantization intervals with task loss,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.
- [40] S. Jain *et al.*, “Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks,” *Machine Learning and Systems*, 2020.
- [41] S. K. Esser *et al.*, “Learned step size quantization,” in *International Conference on Learning Representations*, 2020.
- [42] L. Jiang *et al.*, “Xnor-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 drams,” in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.
- [43] P. Chi *et al.*, “Prime: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.
- [44] A. Chen and M.-R. Lin, “Variability of resistive switching memories and its impact on crossbar array performance,” in *2011 International Reliability Physics Symposium*. IEEE, 2011, pp. MY–7.
- [45] Y. S. Shao *et al.*, “Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures,” in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 97–108.
- [46] Y. N. Wu, J. S. Emer, and V. Sze, “Accelergy: An architecture-level energy estimation methodology for accelerator designs,” in *IEEE/ACM International Conference on Computer-Aided Design*, 2019.
- [47] S. C. Smithson *et al.*, “Neural networks designing neural networks: multi-objective hyper-parameter optimization,” in *Proceedings of the 35th International Conference on Computer-Aided Design*, 2016, pp. 1–8.

- [48] B. Reagen *et al.*, “A case for efficient accelerator design space exploration via bayesian optimization,” in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.
- [49] A. Jones *et al.*, “Apollo: Transferable architecture exploration,” in *ML for Systems Workshop at NeurIPS 2020*, 2020.
- [50] A. S. Rakin *et al.*, “RA-BNN: Constructing robust & accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy,” *arXiv preprint arXiv:2103.13813*, 2021.
- [51] A. Di Mauro *et al.*, “Always-on 674μ W@ 4GOP/s error resilient binary neural networks with aggressive SRAM voltage scaling on a 22-nm IoT end-node,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2020.
- [52] D. G. Mahmoud, V. Lenders, and M. Stojilović, “Electrical-level attacks on cpus, fpgas, and gpus: Survey and implications in the heterogeneous era,” *ACM Computing Surveys (CSUR)*, vol. 55, no. 3, pp. 1–40, 2022.
- [53] N. Chandramoorthy *et al.*, “Resilient low voltage accelerators for high energy efficiency,” in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 147–158.
- [54] B. Ghavami *et al.*, “Blind data adversarial bit-flip attack against deep neural networks,” in *Euromicro Conference on Digital System Design*, 2022.
- [55] A. S. Rakin, Z. He, and D. Fan, “TBT: Targeted neural network attack with bit Trojan,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.
- [56] S. M. Jahinuzzaman *et al.*, “Design and analysis of a 5.3-pJ 64-kb gated ground SRAM with multiword ECC,” *IEEE Journal of Solid-State Circuits*, 2009.
- [57] A. M. Buldu *et al.*, “MBET: Resilience improvement method for DNNs,” in *IEEE International Conference On Artificial Intelligence Testing*, 2022.
- [58] U. Zahid *et al.*, “Fat: Training neural networks for reliable inference under hardware faults,” in *2020 IEEE International Test Conference (ITC)*. IEEE, 2020, pp. 1–10.
- [59] D. Stutz *et al.*, “Random and adversarial bit error robustness: Energy-efficient and secure DNN accelerators,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

- [60] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” *arXiv preprint arXiv:1308.3432*, 2013.
- [61] M. Nagel *et al.*, “A white paper on neural network quantization,” *arXiv preprint arXiv:2106.08295*, 2021.
- [62] G. Mordido, S. Chandar, and F. Leduc-Primeau, “Sharpness-aware training for accurate inference on noisy DNN accelerators,” *arXiv preprint arXiv:2211.11561*, 2022.
- [63] G. Mordido, M. Van Keirsbilck, and A. Keller, “Instant quantization of neural networks using monte carlo methods,” in *Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition*, 2019.
- [64] S. Henwood, F. Leduc-Primeau, and Y. Savaria, “Layerwise noise maximisation to train low-energy deep neural networks,” in *International Conference on Artificial Intelligence Circuits and Systems*, 2020.
- [65] A. Krizhevsky, V. Nair, and G. Hinton, “Cifar-10 (canadian institute for advanced research),” 2009. [Online]. Available: <http://www.cs.toronto.edu/~kriz/cifar.html>
- [66] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, 2017.
- [67] K. He *et al.*, “Deep residual learning for image recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
- [68] H. Robbins and S. Monro, “A stochastic approximation method,” *The Annals of Mathematical Statistics*, 1951.
- [69] P. Yin *et al.*, “Understanding straight-through estimator in training activation quantized neural nets,” *International Conference on Learning Representations*, 2019.
- [70] E. Jang, S. Gu, and B. Poole, “Categorical reparameterization with Gumbel-softmax,” *International Conference on Learning Representations*, 2017.
- [71] M. B. Paulus, C. J. Maddison, and A. Krause, “Rao-blackwellizing the straight-through gumbel-softmax gradient estimator,” *International Conference on Learning Representations*, 2021.