DFQ – Through Weight Equalization and Bias Correction*

Nidhal Saïdani[1]

Sorbonne University Pierre and Marie Curie Campus
Tour Zamansky, 4 Pl. Jussieu, 75005 Paris, France
ns@proton.me
http://nidhalsaidani.com

Abstract. Quantizing deep neural networks (DNNs) in a data-free context presents a practical challenge, often due to constraints such as data privacy, proprietary concerns, or transmission limitations. Conventional approaches equate data-free with training-free, relying on manual weight analysis for quantization. However, a more sophisticated strategy is at hand. This novel method leverages the scale-equivariance property of activation functions to equalize weight ranges within the network, while also addressing biases introduced during quantization. The result is an efficient solution for deploying DNNs on 8-bit fixed-point hardware. In this report, we delve into the principles and applications of this innovative quantization method, evaluating its performance on various computer vision architectures and tasks, including semantic segmentation and object detection, to assess its potential impact on real-world applications.

Keywords: Data-free quantization · Bias correction · 8-bit fixed-point quantization · Deep neural networks

1 Introduction

In recent years, artificial intelligence (AI) has garnered significant attention, permeating diverse publications beyond the tech realm. The buzz surrounding machine learning, deep learning, and AI often conjures visions of a future replete with intelligent chatbots, autonomous vehicles, and seamless virtual assistants. Yet, this outlook oscillates between dystopian prophecies of a job-scarce society dominated by machines and utopian dreams of unparalleled efficiency. As practitioners in the field of machine learning, distinguishing genuine breakthroughs from hyperbolic headlines becomes paramount.

Within the domain of machine learning projects, achieving a harmonious balance between power, efficiency, and cost-effectiveness has remained an ongoing challenge. Frequently, practitioners grapple with the trade-offs, compelled to prioritize certain attributes at the expense of others. However, the imperatives of

* Notes for the paper Data-Free Quantization – Through Weight Equalization and Bias Correction

crafting neural networks tailored for mobile applications permit little latitude for such compromises. In this milieu, practitioners are pressed to not only optimize models for speed but also for compactness and minimal power consumption—an intricate trifecta of requisites that calls for innovative solutions.

It is within this context that the paper under review, "Data-Free Quantization – Through Weight Equalization and Bias Correction," emerges as a pivotal contribution, offering novel insights and techniques to address the pressing demands of model optimization in mobile applications. This report delves into the key methodologies and findings presented in the paper, shedding light on their implications for the evolving landscape of machine learning.

2 Efficient Neural Network Operation with Reduced Precision

2.1 Quantization

Quantization stands as a pivotal technique in reducing the size of a model. It involves the conversion of model weights, initially represented in high-precision floating-point format, into lower-precision formats, including both floating-point (FP) and integer (INT) representations like 16-bit or 8-bit. This transformation yields notable enhancements in both model size and inference speed, while maintaining a commendable level of accuracy. Moreover, quantization serves to optimize model performance by curbing memory bandwidth demands and augmenting cache utilization.

In the domain of deep neural networks, the term "quantized" is commonly associated with the use of INT8 representation. Nonetheless, alternative formats like UINT8 (unsigned version) or INT16 (applicable on x86 processors) also find utility.

While the benefits of quantization are significant, it's essential to recognize that the approach necessitates tailored strategies for diverse models. Achieving successful quantization often hinges on a foundation of prior expertise and rigorous fine-tuning efforts. Furthermore, it introduces fresh considerations and trade-offs, particularly when employing low-precision integer formats like INT8.

2.2 Levels of Quantization Solutions

In literature, the practical application of proposed quantization methods is rarely discussed. To distinguish between the differences in applicability of quantization methods, the authors introduce four levels of quantization solutions, in decreasing order of practical applicability. The axes for comparison are whether or not a method requires data, whether or not a method requires error backpropagation

on the quantized model, and whether or not a method is generally applicable for any architecture or requires significant model reworking. The definitions for each level are as follows:

- **Level 1:** No data and no backpropagation required. Method works for any model. As simple as an API call that only looks at the model definition and weights.
- **Level 2:** Requires data but no backpropagation. Works for any model. The data is used e.g. to re-calibrate batch normalization statistics or to compute layer-wise loss functions to improve quantization performance. However, no fine-tuning pipeline is required.
- **Level 3:** Requires data and backpropagation. Works for any model. Models can be quantized but need fine-tuning to reach acceptable performance. Often requires hyperparameter tuning for optimal performance. These methods require a full training pipeline.
- **Level 4:** Requires data and backpropagation. Only works for specific models. In this case, the network architecture needs non-trivial reworking, and/or the architecture needs to be trained from scratch with quantization in mind. Takes significant extra training-time and hyperparameter tuning to work.

These four levels provide a framework for categorizing quantization methods based on their practical applicability, considering factors such as data requirements, need for backpropagation, and adaptability to different model architectures.

2.3 Comparison: Floating-point vs Fixed-point Representation

To delve further into the comparison between floating-point and fixed-point representation, it's crucial to explore their underlying mechanics. Floating-point representation relies on a mantissa and an exponent to delineate real values, affording flexibility to both components. The exponent empowers the system to span a vast spectrum of numerical magnitudes, while the mantissa governs the level of precision. This dynamic interplay between exponent and mantissa allows for a wide range of values to be accurately represented. Furthermore, the decimal point in this system enjoys a degree of mobility, enabling it to shift relative to the digits, a feature critical for accommodating numbers of vastly differing scales.

On the other hand, fixed-point representation introduces a paradigm shift by replacing the exponent with a constant scaling factor. This strategic alteration paves the way for the application of integers to convey a number's value, expressed as an integer multiple of the established constant. This adjustment, however, firmly anchors the position of the decimal point, which is now rigidly determined by the chosen scaling factor. Visualizing this on a number line, the selected scaling factor governs the minimum separation between two adjacent ticks, and the bit depth allocated to represent the integer dictates the total number of ticks available (as in the case of 8-bit fixed point, allowing for 256

distinct values). This innate characteristic empowers practitioners to make deliberate trade-offs between range and precision, tailoring the representation to suit specific computational needs. Nevertheless, it is important to bear in mind that values not perfectly aligned with the constant multiple will be rounded to the nearest point on the scale, a consideration that becomes paramount in precision-critical applications. This elucidates the nuanced dynamics between range, precision, and computational efficiency inherent to fixed-point representation.

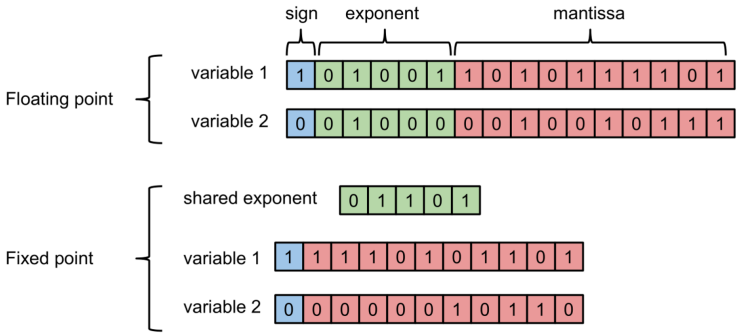

Fig. 1. Floating-point vs Fixed-point

3 Introduction to Quantization

Quantization in neural networks is a widely adopted technique for model compression and acceleration. It involves the approximation of continuous values, such as weights and activations, with discrete counterparts. While quantization offers significant benefits in reducing model size and computational complexity, it introduces quantization errors, often leading to bias in the model's predictions. This report delves into the concept of quantization bias, its impact on neural network performance, and strategies for correction.

4 Quantization in Neural Networks

4.1 Basics of Quantization

Quantization is the process of mapping input values from a large set to output values in a smaller set. In the context of neural networks, it typically involves reducing the precision of the network's parameters and activations from floating-point representation to fixed-point or integer representation.

4.2 Visualizing Quantization Bias

The following figure illustrates the difference between original and quantized distributions, highlighting how quantization can alter the data distribution and introduce bias.

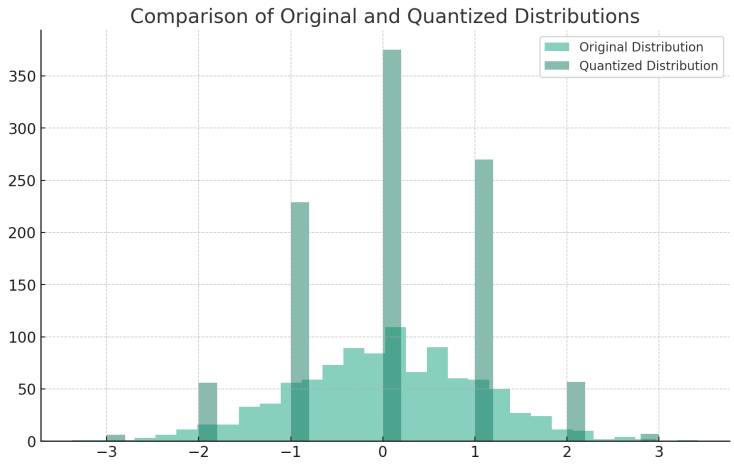

Fig. 2. Comparison of Original and Quantized Distributions

4.3 Types of Quantization Errors

The process of quantization introduces two main types of errors:

- Rounding Error: Occurs due to the approximation of continuous values to the nearest discrete levels.
- Clipping Error: Arises when values are truncated to fit within the representable range of the quantized format.

5 Quantization Bias Correction

Quantization can introduce a biased error in the activations of a neural network. This bias can significantly alter the distribution of the network's outputs, affecting subsequent layers and the overall performance.

5.1 Mathematical Formulation

The core challenge in quantization bias correction lies in accurately representing the continuous values of neural network parameters (weights and activations) with a limited set of discrete values. Let's formulate this mathematically for a fully connected layer.

Quantization Process Consider a fully connected layer with a weight tensor W, input activations x, and the corresponding output $y = Wx$. In a quantized model, the weights are approximated as W_q, leading to a quantized output $y_q = W_q x$.

The quantization of weights can be represented as:

$$W_q = s \cdot round\left(\frac{W}{s}\right) \tag{1}$$

where s is the scale factor determined during the quantization process.

Quantization Error The quantization error, ϵ, is the difference between the original and quantized weights:

$$\epsilon = W_q - W \tag{2}$$

This error propagates through the network, affecting the output as:

$$y_e = W_q x = (W + \epsilon)x = y + \epsilon x \tag{3}$$

where y_e is the output of the quantized layer.

Correcting Quantization Bias To correct the quantization bias, we aim to adjust the output distribution of the quantized layer to match the distribution of the original layer. This involves modifying the bias term of the layer to counteract the shift introduced by ϵ.

The bias correction term, δ, is computed as the expected value of the error:

$$\delta = E[\epsilon x] \tag{4}$$

The corrected output of the quantized layer becomes:

$$y_c = y_e - \delta = (W + \epsilon)x - E[\epsilon x] \tag{5}$$

where y_c represents the bias-corrected output.

Implementation Considerations In practice, the calculation of δ and its incorporation into the network require careful consideration. This includes determining the scale factor s and handling the non-linearity of activation functions, especially in deep networks with multiple layers.

5.2 Quantization Bias Correction in Data-Free Quantization

The Data-Free Quantization (DFQ) method introduces a novel approach to correct the bias introduced by quantization in neural networks. This method is especially useful when access to training data is limited or unavailable. The main idea is to correct for the bias in the layer's output by adjusting the network's batch normalization parameters.

Error Formulation Consider a fully connected layer with weights W, quantized weights W_q, and input activations x. The output error due to quantization is given by:

$$y_e = W_q x = y + \epsilon_x \tag{6}$$

where $\epsilon = W_q - W$ is the quantization error, and y is the pre-activation output of the layer in the original model.

Bias Correction Using Batch Normalization The expectation of the error for output i, $E[\epsilon_x]_i$, if non-zero, indicates a change in the mean of the output distribution. This shift can be corrected by adjusting the bias term in the batch normalization layer. The corrected output is obtained by:

$$E[y] = E[y_e] - E[\epsilon_x] \tag{7}$$

This ensures that the mean of each output unit is preserved post-quantization.

Computing Expected Error To compute the expected error $E[\epsilon_x]$, the DFQ method utilizes the statistics captured by the batch normalization layers. Assuming the activation functions are of the ReLU family, the expected input to the layer $E[x]$ can be estimated from the batch normalization parameters (scale γ and shift β).

The expected value for a channel c in x following a batch normalization layer and a ReLU activation is:

$$E[x_c] = \gamma_c \cdot N\left(\frac{-\beta_c}{\gamma_c}\right) + \beta_c \cdot \left(1 - \Phi\left(\frac{-\beta_c}{\gamma_c}\right)\right) \tag{8}$$

where $N(\cdot)$ and $\Phi(\cdot)$ denote the normal PDF and CDF, respectively. This computation takes into account the clipping effect of the ReLU function.