Technische Universität Berlin

Big Data Engineering (DAMS)
Fakultät IV
Ernst-Reuter-Platz 7
10587 Berlin
https://www.tu.berlin/dams

Thesis

[Choose yours: Bachelor or Master's]

Learned Quantization Schemes for Data-centric ML Pipelines

Anuun Chinbat

Matriculation Number: 0463111

20.01.2025

Supervised by
Prof. Dr. Matthias Boehm
M.Sc. Sebastian Baunsgaard

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 01.01.2024

. . . .

(Signature) [your name]

Abstract

This page is a placeholder for the abstract which should follow this structure:

1. State the problem
2. Say why it's an interesting problem
3. Say what your solution achieves
4. Say what follows from your solution

Additional information on how to structure the abstarct can be found here: `https://mboehm7.github.io/teaching/ws2122_isw/01_Introduction.pdf`, slide 20.

Zusammenfassung

This is a placeholder for the german abstract (Kurzfassung) which should follow the same structure as the abstract.

Contents

1 Introduction

This chapter is a placeholder for the introduction of your thesis.

The first paragraph of the introduction should describe the *context*, followed by 1-3 paragraphs stating the *problems* that are solved in this thesis. The next paragraph should mention *existing work* before introducing the *idea* on how to solve the mentioned problems.

Contributions: In the last paragraph list your contributions and outline the thesis as a list of bullet points containing a short introduction into the chapters.

Additional information can be found here: `https://mboehm7.github.io/teaching/ws2122_isw/01_Introduction.pdf`, slide 21.

2 Background

This chapter covers the theoretical building blocks related to learned quantization, a process whereby the model learns its quantization parameters during training instead of receiving static quantization requirements as input or undergoing discretization post training. Although the primary focus of this thesis exclusively lies on learned quantization, a general explanation of the term quantization and its types will be provided in the first section of this chapter to deliver a bigger picture of this work's relevance in the broader context. The following sections will detail the specific terms and methods that are directly associated to the technical setup employed in the experiments, starting from the basics of deep neural networks (NNs) and ending with their common modification techniques used in learned quantization scenarios.

2.1 Quantization

This section aims to answer the *why* question with respect to quantization and further provides a broader understanding of the term regarding its types.

2.1.1 Purpose & Definition

With humans becoming increasingly dependent on deep learning models disguised as everyday tools — such as facial recognition filters, document scanners, self-driving cars, and more — the need for these models to function in a resource- and time-efficient manner is more imperative than ever. Among the many ways to meet this need, quantization has become one of the most common techniques used to reduce the computational and memory costs of deep NNs, given the sheer amount of parameters they possess and the inherent redundancy this introduces.

While the term *quantization* has its roots in the first half of the 20th century [4], in the context of NNs, it has gained renewed importance — the over-parameterization of NNs introduces a certain type of flexibility that allows quantization to be implemented in many different forms [3]. Despite the abundance of these forms and approaches, quantization of NNs generally refers to the process of reducing the bit precision of their parameters, all while maintaining accuracy within an acceptable range.

2.1.2 Quantization Types

Although there is a multitude of ways to classify NN quantization methods, the most general classification is the division between *static, dynamic*, and *learned quantization.*

- In **static quantization**, quantization parameters are fixed before model inference, based on the data observed during training or calibration. This corresponds to Post-Training Quantization (PTQ), which directly quantizes the trained floating-point model [6], using various discretization approaches based on the range and distribution of the parameters being quantized, as well as the level of granularity at which the quantization values are applied.

- In **dynamic quantization**, the quantization parameters are calculated dynamically during inference. This typically applies to activations, as their range changes depending on the input data and, unlike that of the weights, cannot be precomputed with static quantization [7].

- In **learned quantization**, quantization parameters are learned as part of the model training process. This corresponds to Quantization Aware Training (QAT) [5], where quantization is integrated directly into training rather than applied afterwards. Since learned quantization is central to this work, a detailed review of QAT and its applications is provided in the Related Work chapter.

2.2 Fundamentals of Deep Learning

With the definition of quantization clarified, this section introduces the fundamental concepts of deep learning necessary for understanding it, beginning with simple neural network architectures and progressing to the concepts of forward-pass and back-propagation.

2.2.1 Dense Layers in Neural Networks

NNs are a mathematical abstraction of the human brain, where nodes pass signals to one another, similar to how neurons in the brain communicate. These nodes take an input and generate an output, much like answering a question based on given information. Any information can be represented as a set of numeric values — for example, an image is a set of pixel values, each representing a specific color. Using this set of values as input, a NN can generate an output, possibly identifying what's in the image.

To generate an output from the input, the classical form of NNs, called the Multilayer Perceptron (MLP), processes the input through multiple layers of nodes, where each layer applies a transformation to the data. These transformations are determined by the **weights** (W) and **biases** (b) associated with the nodes, which are learned during training. At each layer, the operation is defined as:

$$\text{output} = f(W \cdot \text{input} + b)$$

where f is an activation function applied to the weighted sum. The process repeats across layers until the final output is produced, such as identifying the content of an image.

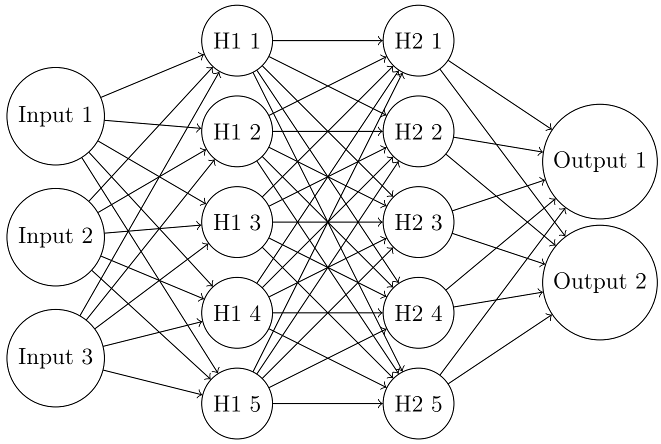

Figure 2.1: Multilayer Perceptron with Two Hidden Layers

The layers in between, where this operation is applied, are called hidden layers. They are typically dense layers, meaning each node in one layer is connected to every node in the next. Dense layers introduce the inherent redundancy of NNs due to the large number of numeric values they contain, particularly in W and b. Thus, modifying the operations these layers perform and the parameters they learn will be a key focus of this work.

2.2.2 Loss Functions & Regularization

The training process of NNs is similar to how our brains learn from mistakes. Given the ground truth, an MLP, for example, adjusts its learnable parameters in W and b using a specific function that compares the ground truth with the output generated by the MLP.

This function is called a *loss function*, and depending on the type of question the network aims to answer, it can take many different forms. For example, for the MLP described in Figure 2.1 that generates a binary classification, we would use the *log loss* function. Since the datasets used in this thesis involve multi-class classification, the *sparse categorical cross-entropy* (SCCE) loss function will be used, which measures the difference between the predicted class probabilities and the true labels for each class in the dataset.

Often the loss function alone is not enough for the NN to perform well. This is why a regularization term that penalizes unwanted behaviours is added to the loss function.

An example approach is to define a regularization term that directly penalizes large differences between full-precision values and their quantized counterparts [8]. This can encourage the model to learn parameters that are easily quantizable without significant performance loss. If the values that are being quantized are W, then this regularization term could look like:

$$\mathcal{L}_{\text{quant}} = \lambda \sum_i \|W_i - q(W_i)\|^2$$

Where:

- λ is a scalar that controls the importance of the quantization penalty.
- W_i represents the full-precision weight value before quantization at index i.
- $q(W_i)$ represents the quantized version of W_i.

The current work uses multiple custom regularization terms that trigger the quantization process during training. These will be discussed in detail in the Experimental Setup section.

2.2.3 Forward-Pass & Back-Propagation

The repetition of the operation described earlier in the Dense Layers in Neural Networks subsection during model training is essentially the forward pass. It is the process where input data is passed through the network layer by layer, with each layer applying its learned weights and biases to produce a final output.

As mentioned in the previous subsection, this output is then compared with the ground truth by the loss function that produces an error. This error is used to update the learnable parameters — in case of MLPs, the values in W and b — during a process called back-propagation.

In other words, back-propagation is the method by which the network adjusts its parameters to minimize the error. It calculates the gradient of the loss function with respect to each parameter using the chain rule. W and b are typically updated as follows:

$$W = W - \eta \frac{\partial L}{\partial W}, \quad b = b - \eta \frac{\partial L}{\partial b}$$

where L is the loss function, and η is the learning rate.

For example, consider the weight $W_{I1,H1_1}$ represented as the line between *Input 1* and the hidden layer node $H1_1$ in Figure 2.1. The gradient of this weight with respect to the loss is calculated using the chain rule:

$$\frac{\partial L}{\partial W_{I1,H1_1}} = \frac{\partial L}{\partial \text{Output 1}} \cdot \frac{\partial \text{Output 1}}{\partial H1_1} \cdot \frac{\partial H1_1}{\partial W_{I1,H1_1}}$$

Where:

- $\frac{\partial L}{\partial \text{Output 1}}$ is the gradient of the loss with respect to *Output 1*.

- $\frac{\partial \text{Output 1}}{\partial H1_1}$ is the gradient of *Output 1* with respect to the output of $H1_1$.
- $\frac{\partial H1_1}{\partial W_{I1,H11}}$ is the value of *Input 1*, since $H1_1$ is a weighted sum of the inputs.

This shows how each weight contributes to the final error during back-propagation.

2.3 Common Concepts of QAT

Now that the fundamentals of deep learning have been covered, this section introduces concepts commonly encountered in learned quantization, including the modified forward-pass technique and the challenges it creates for back-propagation.

2.3.1 Low-Precision Forward-Pass

In QAT, the operation performed during the forward-pass is often modified. Instead of propagating the full precision values, the output of a layer is first quantized and then dequantized before being passed to the next layer [5]. This simulates the effect of quantization during inference, helping the network adjust to the reduced precision parameters.

The *quantization* process can be generalized as follows:

$$q(x) = \text{round}\left(\frac{x}{P}\right) - Z$$

where $q(x)$ is the quantized value, x is the full precision value of the parameter that is being quantized, P is a quantization parameter, such as a scaling factor, and Z represents the zero point. This method is the most commonly used *uniform quantization*, which can be either *asymmetric* or *symmetric*, depending on the value of Z, with $Z = 0$ representing *symmetric quantization*.

For *dequantization*, we apply:

$$x_{\text{dequant}} = P \cdot (q(x) + Z)$$

Note that the dequantized value does not necessarily match the initial value. For example, if $x = 7.8$ and $P = 2$, then quantization gives $q(x) = 4$, and dequantization results in $x_{\text{dequant}} = 8$. This demonstrates how the *quantization* and dequantization process approximates the original value x by rounding it to the nearest representable value based on the quantization parameter P.

2.3.2 Full-Precision Back-Propagation

In QAT, although the forward-pass operates on quantized values, back-propagation typically uses full-precision values. This is because the rounding operation in the forward-pass is non-differentiable, which makes it impossible to compute gradients directly.

Consider the example of computing the gradient of the weight $W_{I1,H1_1}$ in Figure 2.1:

$$\frac{\partial L}{\partial W_{I1,H1_1}} = \frac{\partial L}{\partial \text{Output 1}} \cdot \frac{\partial \text{Output 1}}{\partial H1_1} \cdot \frac{\partial H1_1}{\partial W_{I1,H1_1}}$$

If the output of $H1_1$ was quantized using a rounding operation, the gradient

$$\frac{\partial H1_1}{\partial W_{I1,H1_1}}$$

would be undefined due to the non-differentiability of rounding. This would break the gradient flow, making it impossible to update $W_{I1,H1_1}$ during back-propagation.

To work around this, techniques such as the Straight-Through Estimator (STE) are commonly used [1] [2] [8]. The STE approximates the gradient by treating the rounding function as if it were differentiable during the back-propagation. This approach ensures that the model can simulate low-precision inference during the forward pass, while still benefiting from the accuracy of full-precision gradient updates during back-propagation.

The Experimental Setup section will explain a slighly different version of the typical STE that was used in the current work.

3 Design

This chapter elaborates on the problem that this thesis tries to solve and explains the individual methods used for solving the problem.

4 Experiments

This chapter provides details about the experiments conducted within the context of this thesis.

4.1 Experimental Setup

All experiments are caried out on machine XYZ.

4.2 Results for Experiment A

Figure 4.1 illustrates the situation between Alice and Bob. (sequence diagram from www.websequencediagrams.com)

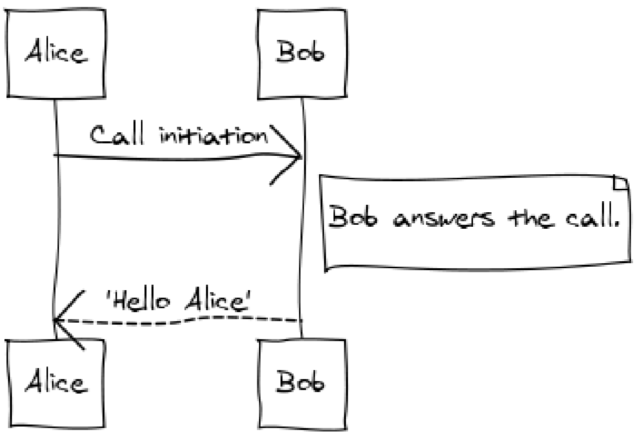

Figure 4.1: Alice and Bob

4.3 Results for Experiment B

In Table 4.1 the statistics for...

Dateset	Minimum y	Maximum y	Average y
DS 1	-68.57	506.78	86.05
DS 2	-0.18	537.67	102.51

Table 4.1: This table shows the statistics (minimum, maximum, and average) for the different datasets.

4.4 Discussion of Results

...

5 Related Work

There is a large amount of scientific work that has been done on QAT, and it can be categorized based on different characteristics, separately covered in the sections of this chapter.

5.1 Quantization Target Parameters

In QAT, one of the key decisions relates to which parameters of the NN to quantize.

5.2 Granularity of Quantization

Granularity of quantization refers to the level at which quantization is applied within a neural network.

5.3 Handling of Differentiability

5.4 Precision in Quantization

5.5 Integration with Pruning & Other Techniques

5.6 Modifications to Loss Functions

6 Conclusions

This chapter summarizes the contriubtions of the thesis and provides an outlook into future work.

List of Acronyms

ML	Machine Learning

Bibliography

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". In: *arXiv preprint arXiv:1308.3432* (2013).

[2] Angela Fan et al. "Training with Quantization Noise for Extreme Model Compression". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. Facebook AI Research, LORIA, Inria. 2021.

[3] Amir Gholami et al. "A Survey of Quantization Methods for Efficient Neural Network Inference". In: *arXiv preprint arXiv:2103.13630* (2021).

[4] Robert M. Gray and David L. Neuhoff. "Quantization". In: *IEEE Transactions on Information Theory* 44.6 (1998), pp. 2325–2383.

[5] Benoit Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2704–2713.

[6] Chutian Jiang. "Efficient Quantization Techniques for Deep Neural Networks". In: *Proceedings of the 2021 International Conference on Signal Processing and Machine Learning*. 2021.

[7] Sehoon Kim et al. "I-BERT: Integer-only BERT Quantization". In: *Proceedings of the 38th International Conference on Machine Learning*. 2021, pp. 5506–5518.

[8] Bohan Zhuang et al. "Towards Effective Low-bitwidth Convolutional Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 7920–7928.

Appendix

Add additional experimental results that do not need to be directly included in the thesis body.