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 addresses the theoretical and contextual background necessary to understand the key concepts and methodologies that form the foundation of the current research. The first section will discuss the basics of deep neural networks (NNs), upon which the technical setup of this thesis is based. The next section aims to provide a broader context for the term *quantization*, followed by a final section that explains the common NN modification techniques used in learned quantization scenarios.

2.1 Fundamentals of Deep Learning

This section introduces the fundamental concepts of deep learning, beginning with the most basic NN architecture and progressing to loss functions with regularization. The concepts of the forward pass and backpropagation will be explained in the last subsection.

2.1.1 Dense Layers in Neural Networks

NNs can be considered a mathematical abstraction of the human decision-making process. Consider a scenario where, given an image, you need to say aloud what you see. The two eyes can be regarded as input nodes that receive the initial data, the brain can be seen as the hidden layer that processes this data, and your mouth — the output node that provides the final answer.

The hidden layer, which typically consists of many neurons, is where the magic — or the transformation of data — happens. In its simplest form, within the classic *Multilayer Perceptron* (MLP) model, each hidden layer neuron performs a weighted operation:

$$output = f(w \cdot input + b)$$

where:

- *input* refers to the outputs from the previous layer (or the initial data from input nodes in our case) that are fed into a specific neuron in the hidden layer.
- w (weights) is a vector of parameters associated with that specific neuron, defining the importance of each input received by this neuron.
- b (bias) is an additional scalar parameter specific to the neuron, which shifts the result of the weighted sum, allowing for more flexibility.

- *f* is the *activation function*, a nonlinear function applied to the weighted sum of inputs and bias in that specific neuron, allowing for more complexity.
- *output* is the result produced by the neuron, which will then be passed on to the next hidden layer or to the final output layer [4].

Hidden layers where each neuron is connected to every neuron in the previous layer and every neuron in the next layer are called *dense layers*.

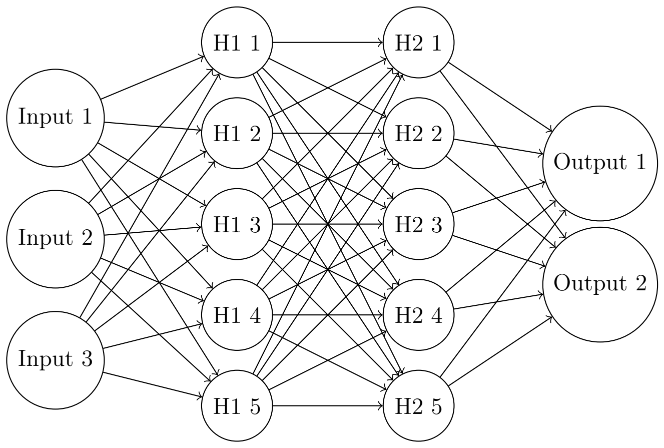

Figure 2.1: Multilayer Perceptron with Two Dense Hidden Layers

This interconnectedness of dense layers introduces the inherent redundancy of NNs. It is particularly true in models with a large number of neurons, where the *weight matrix* W — made up of the weight vectors w for each neuron in the layer — results in a vast number of parameters, many of which have little influence on the final output. Thus, customizing these dense layers will be one of the key focus points in the current work.

2.1.2 Loss Functions & Regularization

The weights and biases are usually *learnable parameters* that the model adjusts during *training*. The training process of NNs is similar to how our brains learn from mistakes. Given the ground truth, a NN adjusts its learnable parameters using a specific function that compares the ground truth with the output generated by the network, essentially measuring the magnitude of the network's errors.

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 a NN to perform well. This is why a *regularization term* that penalizes unwanted behaviours is added to the loss function.

A typical regularization term is L_2, which penalizes large weights by adding the sum of the squared weights to the loss function. The modified loss function is expressed as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \sum_i w_i^2$$

where:

- $\mathcal{L}_{\text{data}}$ is the original loss function (in our case, the SCCE loss function).
- λ is a scalar parameter that controls the strength of the regularization.
- w_i represents each individual weight value in the model.

The current work employs multiple custom regularization terms that encourage specific behaviors in the models while discouraging others. These terms will be discussed in detail in the Experimental Setup section.

THIS PART WILL GO TO THE NEXT SECTION:

An example approach is to define a regularization term that directly penalizes large differences between full-precision values and their quantized counterparts [13]. 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.

2.1.3 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.

2.2 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.

2.2.1 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 [3], 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 [2]. 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.

2.2.2 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 [7], 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 [8].
- In **learned quantization**, quantization parameters are learned as part of the model training process. This corresponds to Quantization Aware Training (QAT) [6], 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.

2.3 Common Approaches in 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 TODO

2.3.2 TODO

3 Learned Quantization

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

3.1 Custom Quantization Layers

3.2 Custom Loss Functions

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 Hyperparamter Tuning

4.3 Dataset-Specific Results

5 Related Work

A significant amount of scientific work has been done on QAT. This research can be categorized based on different characteristics, which are covered separately in the following paragraphs.
Model architecture.
RNN - [10] CNN - [12]
Quantization target parameters.
weights and activations - [9]
weights and activations - [5]
weights - [11] weigths - [10] weights and inputs - [12]
Granularity of quantization.
Handling of differentiability.
Quantization precision.
binary weights - [1] binary weights and activations - [5] ternary weights - [10]
Integration with pruning & other techniques.
pruning and Huffman Codign - [4]
distillation - [11]
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] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. “BinaryConnect: Training Deep Neural Networks with binary weights during propagations”. In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes et al. 2015, pp. 3123–3131. URL: `https://proceedings.neurips.cc/paper/2015/hash/3e15cc11f979ed25912dff5b0669f2cd-Abstract.html`.

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

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

[4] Song Han, Huizi Mao, and William J. Dally. “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: `http://arxiv.org/abs/1510.00149`.

[5] Itay Hubara et al. “Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations”. In: *CoRR* abs/1609.07061 (2016). arXiv: `1609.07061`. URL: `http://arxiv.org/abs/1609.07061`.

[6] 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.

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

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

[9] Raghuraman Krishnamoorthi. “Quantizing deep convolutional networks for efficient inference: A whitepaper”. In: *arXiv preprint arXiv:1806.08342* (2018).

[10] Joachim Ott et al. “Recurrent Neural Networks With Limited Numerical Precision”. In: *CoRR* abs/1611.07065 (2016). arXiv: `1611.07065`. URL: `http://arxiv.org/abs/1611.07065`.

[11] Antonio Polino, Razvan Pascanu, and Dan Alistarh. "Model compression via distillation and quantization". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: `https://openreview.net/forum?id=S1XolQbRW`.

[12] Mohammad Rastegari et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". In: *CoRR* abs/1603.05279 (2016). arXiv: `1603.05279`. URL: `http://arxiv.org/abs/1603.05279`.

[13] 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.