Lattice Quantization

Clément Metz
CEA-List
University Paris-Saclay
Palaiseau, France
clement.metz@cea.fr

Thibault Allenet
CEA-List
Palaiseau, France
thibault.allenet@cea.fr

Antoine Dupret
CEA-Leti
Palaiseau, France
antoine.dupret@cea.fr

Johannes Thiele
CEA-List / Axelera.ai
Zurich, Switzerland
johannes.c.thiele@gmail.com

Olivier Bichler
CEA-List
Palaiseau, France
olivier.bichler@cea.fr

Abstract

Post-training quantization of neural networks consists in quantizing a model without retraining, which is user-friendly, fast and data frugal. In this paper, we propose LatticeQ, a novel post-training weight quantization method designed for deep convolutional neural networks. Contrary to scalar rounding widely used in state-of-the-art quantization methods, LatticeQ uses a quantizer based on lattices – discrete algebraic structures. LatticeQ exploits the inner correlations between the model parameters to the benefit of minimizing quantization error. This allows to achieve state-of-the-art results in post-training quantization. In particular, we demonstrate ImageNet classification results close to full precision on the popular Resnet-18/50, with little to no accuracy drop for 4-bit models. Our code is available here.

1 Introduction

Quantization is an effective way to make neural networks viable for low power and/or low memory applications. Most prominent breakthroughs [Esser et al., 2020, Jin et al., 2019] to quantize Deep Convolutional Neural Networks rely on retraining the model from scratch. Those lead to best performance but they are data-expensive and computationaly intensive. Thus, one may not have access to sufficient resources to use them. To overcome these constraints, post-training quantization methods [Krishnamoorthi, 2018] have been suggested that do not rely on retraining a network from scratch. While it leads to inferior task performance, these methods do solve the aforementioned deployment problems, that are critical in many real world applications.

In this paper, we introduce a post-training quantization technique for deep convolutional neural networks, which achieves state-of-the-art classification performance on ImageNet down to 4-bit weights on Resnet architectures. Our method relies on a novel quantizer which exploits linear correlations between the parameters of convolution layers to minimize its error. This paper is organized as follows: section 2 presents previous work on post-training quantization. Section 3 analyzes the parameter correlations within DCNNs and introduces the intuition behind our approach. In section 4, we detail our approach and in section 5, we compare it to existing state-of-the art methods under various post-training hypotheses. Finally, in section 6, we provide analysis of the characteristics of our quantizer, including its quantization error, its distribution, and its memory overhead.

2 Related work

The performance of DCNN quantization schemes heavily depends on hypotheses: availability of training data, computation time and hardware capacities. Post-training quantization meets stricter specifications than quantization-aware training. Nagel et al. [2019] assumes that no data is available at all for quantization. Banner et al. [2019] introduces bias correction and per-channel bit allocation. Choukroun et al. [2019] specifically designes a quantizer to minimize the MSE loss of the quantization operation. Other approaches that use a few samples of data for calibration have been proposed. Nagel et al. [2020] uses data in order to learn whether to round the weights up or down. Wang et al. [2020] proceeds by bit optimization, and Liu et al. [2021] exploits multipoint quantization to approximate full precision weight vectors with a linear combination of low bit vectors. Li et al. [2021] notices that blockwise optimization gave better results than usual layerwise optimization.

3 Preliminary observations

Figure 1 (left) shows the correlation matrix of the 9 distributions of filter parameters in a 3×3 convolution layer. Formally, we plot in row i and column j the points (w_i, w_j) for each filter $f = (w_1, ..., w_9)$ in a chosen layer (blue dots). On the long diagonal, we plot the histogram of w_i. We notice that filters tend to have linearly correlated coordinates. The same observation can be made in other 3×3 layers. From this observation, we justify the main assumption of our method: a quantizer “shaped as a parallelogram" reduces quantization error more than a scalar uniform quantizer – hence “shaped as a square" (see Figure 1, right).

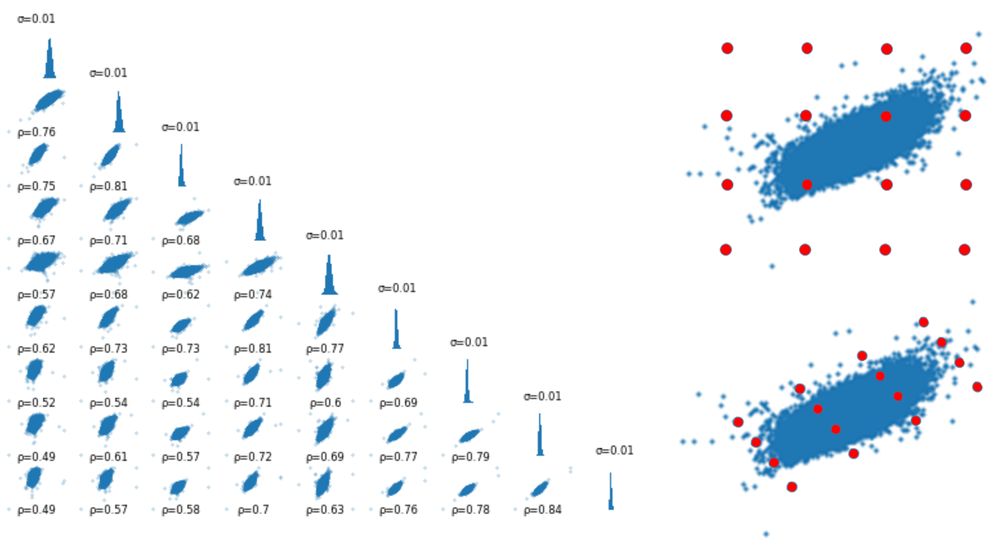

Figure 1: Left: Correlation diagram of filters in layer 4.0 conv2 (Conv2d 3×3). Right: Uniform quantization (up) and lattice quantization (down). Red dots are quantization points.

4 Methods

Many state-of-the-art quantization methods rely on scalar uniform quantizers, which require only a few additional parameters, namely, step size, bitwidth, and/or threshold. Other approaches use vector quantization, which may either be scalar or multidimensional, leading to more flexible quantization sets Han et al. [2016], Stock et al. [2020]. However, these methods are often limited by the need of codebooks. LatticeQ takes the best of both approaches. It adds a limited number of parameters to encode quantization bases, and offers a broader variety of possible quantization sets than scalar uniform quantizers. In this section, we explain how LatticeQ works. See Appendix D for a numerical example.

4.1 Quantization

In order to quantize the weights, LatticeQ uses lattices, which are algebraic structures that discretize the notion of vector space. Each lattice has a basis, meaning that each point of the lattice can be written as an integer linear combination of the vectors of this basis. This integer linear combination is the encoding of LatticeQ. A lattice has infinite cardinality. In order to use this structure as our quantizer, we need a finite number of quantization points. We truncate our lattice in the following fashion:

Let Λ be a lattice, and $\mathcal{B} = (\mathbf{b_i})_{1 \leq i \leq n} \in \mathbb{R}^n$ a basis of Λ. Given b the bitwidth, our quantization set is $Q = \{q \in \mathbb{R}^n, q = \sum_{i=1}^{n} f_i \mathbf{b_i},\ \forall i \in \{1, ..., n\},\ -2^{b-1} \leq f_i \leq 2^{b-1} - 1\}$.

The quantization process for a 3×3 layer is the following: we flatten the weights and group them by blocks of 3. Then, using the quantization basis, we search for the nearest quantization point to this block. The vector found is the quantization point for this block. In the scalar model, the quantization operation relies on a simple round function, but quantizing on any lattice requires a more complex algorithm. The problem of finding the closest lattice point to a real vector is known as Closest Vector Problem (CVP):

"Given $x \in \mathbb{R}^n$ and Λ a lattice of $\mathbb{R}^n$, find $\lambda \in \Lambda$ such that: $d(x, \lambda) = min\{d(x, l),\ l \in \Lambda\}$."

In order to solve the closest vector problem, we use the nearest plane algorithm Babai [1986]. It is fairly easy to implement, computationally light, and still provides good approximations in low dimension. We detail its implementation in Appendix B.

4.2 Dequantization

From an implementation standpoint, in memory, each quantized block is represented by its coordinates in the quantization basis. Let us suppose $\mathcal{B} = (\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}) = ((b_{1,1}, b_{1,2}, b_{1,3}), (b_{2,1}, b_{2,2}, b_{2,3}), (b_{3,1}, b_{3,2}, b_{3,3}))$ is our quantization basis (with each $b_{i,j}$ a scalar, possibly quantized with a uniform min/max quantizer), and a 3×3 quantized filter F^q:

$$F^q = \begin{pmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{pmatrix} \tag{1}$$

It consists in 3 quantized blocks: (f_1, f_2, f_3), (f_4, f_5, f_6) and (f_7, f_8, f_9). The convolution represented by F^q is the concatenation of three 1×3 vectors:

$$\hat{F} = Dequant(F^q) = \begin{pmatrix} (f_1\mathbf{b_1} + f_2\mathbf{b_2} + f_3\mathbf{b_3}) \\ (f_4\mathbf{b_1} + f_5\mathbf{b_2} + f_6\mathbf{b_3}) \\ (f_7\mathbf{b_1} + f_8\mathbf{b_2} + f_9\mathbf{b_3}) \end{pmatrix} \tag{2}$$

Note that if we choose $\mathcal{B}$ as a uniform scaling of an orthonormal basis ($\mathcal{B} = \lambda \times I_n$), the quantization set is exactly the one of classic scalar quantization (we refer to this simplified version as "Cubic LatticeQ"). In practice, we can use $Dequant$ to compute the convolution in the quantized network. However, for per-layer quantization, we propose an optimized way of dealing with the inference in a LatticeQ-quantized network with quantized activations in Appendix A.

4.3 Basis search

Once the problem of quantizing on a lattice is adressed, there remains to find the most relevant lattice. For the data free approach, we opt for a simple random search with restarts, where restarts simply consist in running the algorithm several times in a row and keeping the best result of all the runs. We look for a lattice that reduces the mean cube error loss (MCE) between the full precision weights of the layer (or channel), and their quantized version. The pseudocode of this algorithm is detailed in Appendix C.

5 Experiments

We evaluate our method on the ImageNet [Russakovsky et al., 2015] classification task. All experiments are made using PyTorch [Paszke et al., 2019], and the pretrained models used all come from the torchvision.models library. Based on experiments, we set bases dimension to 3 for 3×3 convolution layers, 2 for 1×1 layers and linear layers, and 1 for the first layer (since we did not notice any advantage in increasing the dimension for this particular layer). We report bitwidths settings and top-1 accuracy for each tested model, and we also provide the results from Banner et al. [2019], Choukroun et al. [2019] for comparison.

5.1 Zero-shot LatticeQ

In this section, we suppose that no data sample is available at all for quantization.

Table 1: LatticeQ with bias correction on ImageNet. We use per-channel quantization and bias correction. We compare our results with results that we generated from the source code of Banner et al. [2019], using per-channel quantization and bias correction. We also compare with paper results from OMSE [Choukroun et al., 2019].

		Top-1 accuracy		
Network	Method	W4A32	W3A32	W2A32
Resnet-18 (69.8)	LatticeQ (Ours)	**69.0**	**66.7**	**41.7**
	Banner et al.	67.5	43.2	1.2
	OMSE+opt	67.1		
Resnet-50 (76.0)	LatticeQ (Ours)	**75.6**	**73.6**	**47.3**
	Banner et al.	74.8	67.4	0.4
	OMSE+opt	74.7		
VGG16-bn (73.4)	LatticeQ (Ours)	**72.9**	**70.9**	**40.7**
	Banner et al.	71.6	65.9	0.1
Densenet-121 (74.4)	LatticeQ (Ours)	**73.3**	**68.9**	**10.4**
	Banner et al.	69.8	54.2	0.5
	OMSE+opt	71.7		
Mobilenet-v2 (71.9)	LatticeQ (Ours)	**68.2**	**48.9**	**0.3**
	Banner et al.	62.8	13.9	0.1

The results in Table 1 show the substantial improvement brought by LatticeQ over state-of-the-art approaches. Our method outperforms Banner et al. [2019] with similar experimental hypotheses, and also the OMSE+opt method, in almost all settings for all presented models. We find that LatticeQ manages to stay within 1% of full precision model accuracy on Resnets for 4-bit weights, and within 3% for 3-bit weights, whereas Banner et al. [2019] collapses. As expected, compact models like Densenet [Huang et al., 2018] and Mobilenet-V2 [Sandler et al., 2018] are less resilient to quantization than Resnets [He et al., 2016a].

5.2 LatticeQ with a few data samples

It has lately become mainstream to use a few samples of calibration data to finetune quantized weights [Nagel et al., 2020, Wang et al., 2020, Li et al., 2021], and it has substantially improved the performance of post-training quantization on a variety of networks. We also wanted to demonstrate the potential of our method under slightly different hypotheses. Therefore, using the insight from Li et al. [2021], we designed a simple optimization mechanism. For some network sub-block function B_i with full precision network input x_i, quantized network input x_i^q and $\mathcal{L}$ some loss, the optimization problem is given by

$$\arg\min_{B_i^q} \mathcal{L}(B_i(x_i), B_i^q(x_i^q)) \tag{3}$$

First, we initialize per-layer quantization bases with Algorithm 2. We select a sample of 512 calibration images, and split them into 4 batches of 128. Using this sample, we perform 2000 training epochs for each block and learn its weights and bases with Adam [Kingma and Ba, 2017] optimizer. We use the MSE loss and STE [Bengio et al., 2013] for gradient propagation through the round function. The learning rate is set to 1.10^{-6} for 4-bit and 2.10^{-6} for 3-bit. Quantization is performed per layer as it is common practice in level 2 state of the art. For this setting, we compare our results with paper results from Adaround [Nagel et al., 2020], Bitsplit [Wang et al., 2020] and results from the source code of Brecq [Li et al., 2021] in Table 2. It is important to note that Li et al. [2021] experimented with preresnet [He et al., 2016b] architecture, which is not the same as He et al. [2016a]. Also, for the sake of fairness, we only consider per-layer quantization results. Table 2 shows that LatticeQ also has potential when it comes to retraining. We find that LatticeQ maintains the accuracy drop at 1.2% for 3-bit on standard Resnet-18, and even slightly improves on Brecq [Li et al., 2021], by reusing the same blockwise optimization method.

Table 2: LatticeQ with data samples

	Top-1 accuracy		
Network	Method	W4A32	W3A32
Resnet-18 (69.8)	LatticeQ (Ours)	**69.3**	**68.6**
	Adaround	68.6	
	Bitsplit	69.1	66.8
Preresnet-18 (71.0)	LatticeQ (Ours)	**70.5**	**69.3**
	Brecq	70.3	69.0

6 Analysis

6.1 Lattice quantization compared with scalar quantization

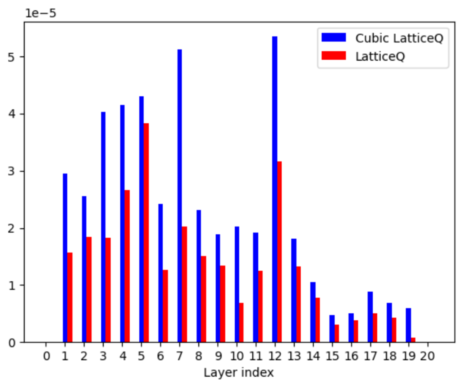

Figure 2: Resnet18 per-layer quantization error comparison between LatticeQ and Cubic LatticeQ (scalar quantization). Vertical axis is MCE.

Table 3: Comparison between baseline per-channel LatticeQ and baseline per-channel Cubic LatticeQ (scalar quantization).

Network	Method	FP32	W4A8
Resnet-18	LatticeQ	69.8	**67.2**
	Cubic LatticeQ	69.8	57.6

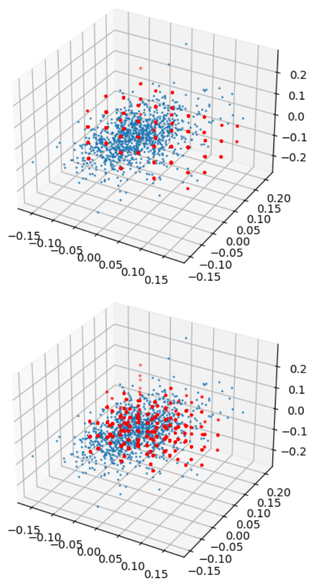

Figure 3: Up: 2-bit Cubic LatticeQ quantization points (red) and 1×3 filter blocks (blue). Down: Same with 2-bit baseline LatticeQ quantization points.

We experimentally demonstrate the advantages, both in terms of quantization error (Figure 2) and task loss (Table 3), in using a deformable lattice for quantization rather than a cubic lattice (which is equivalent to uniform scalar quantization). This confirms our hypothesis that the inner correlations of the parameters of a neural network can be exploited for the purpose of quantization. On Figure 3, each full precision filter block is represented by a blue dot in the 3D space, and each quantization point of our method is represented by a red dot in the 3D space. The comparison between both distributions clearly shows that LatticeQ increases the concentration of quantization points in the most critical areas of the filters' multidimensional distribution, as intended.

6.2 Memory overhead

Table 4: Baseline LatticeQ memory cost given in bit/weight, taking into account the extra memory required for the bases.

Type	Network	W4A8 memory storage	Avg. bit/weight
Per layer	Resnet-18	6.100 MB	4.00
	Resnet-50	13.78 MB	4.00
	Densenet-121	4.465 MB	4.00
	Mobilenet-v2	2.376 MB	4.00
Per channel	Resnet-18	6.158 MB	4.02
	Resnet-50	14.01 MB	4.04
	Densenet-121	4.555 MB	4.04
	Mobilenet-v2	2.547 MB	4.17

We compute the memory overhead resulting from our method, both in per-channel and per-layer quantization settings. See Table 4 for a few examples among the networks we experimented with in this paper. We report the bit-equivalent memory cost of 4-bit compressed models. The compression ratio penalty due to quantization bases is negligible in the per-layer setting, and less than 1% in the per-channel setting for all architectures but Mobilenet (because of group convolutions). Compressions that can be achieved by descending to lower bitwidths therefore largely offset the memory overhead of LatticeQ.

7 Conclusion

In this paper, we introduced LatticeQ, a novel post-training quantization method which exploits the flexibility of lattice quantizers for the purpose of DCNN quantization. LatticeQ is particularly useful to deploy models trained in floating point precision on lightweight architectures without requiring a single training sample (which could happen for confidentiality, safety reasons, or for the sake of simplicity). We showed that our quantizer significantly outperforms state-of-the-art methods based on the scalar quantizer for 3 and 4-bit quantization on several well-known architectures. Furthermore, we believe that lattice quantization has potential beyond post-training and that LatticeQ could inspire other works outside of the realm of scalar quantization.

References

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned Step Size Quantization. *arXiv:1902.08153*, 2020.

Qing Jin, Linjie Yang, and Zhenyu Liao. Towards Efficient Training for Neural Network Quantization. *arXiv:1912.10207*, 2019.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv:1806.08342*, 2018.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-Free Quantization Through Weight Equalization and Bias Correction. *ICCV*, 2019.

Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *NeurIPS*, 2019.

Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit Quantization of Neural Networks for Efficient Inference. *arXiv:1902.06822*, 2019.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? Adaptive rounding for post-training quantization. 2020.

Peisong Wang, Qiang Chen, Xiangyu He, and Jian Cheng. Towards Accurate Post-training Network Quantization via Bit-Split and Stitching. *ICML*, 2020.

Xingchao Liu, Mao Ye, Dengyong Zhou, and Qiang Liu. Post-training Quantization with Multiple Points: Mixed Precision without Mixed Precision. *AAAI*, 2021.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction. *ICLR*, 2021.

Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149*, 2016.

Pierre Stock, Armand Joulin, Remi Gribonval, Benjamin Graham, and Herve Jegou. And the bit goes down: Revisiting the quantization of neural networks. *ICLR*, 2020.

L Babai. Nearest lattice point problem. 1986.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *arXiv:1409.0575*, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPS*, 2019.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993*, 2018.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. IEEE, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CVPR*, 2016a.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks, 2016b.

A Optimized inference for LatticeQ networks

In this section, we suppose that we have a 3×3 convolution layer Λ quantized using our per-layer method, with in input channels and out output channels. We want to compute a forward pass through this layer. Let $(C_i^{in})_{1 \leq i \leq in}$ the set of input channels and $(C_j^{out})_{1 \leq j \leq out}$ the set of output channels. We want to compute C_j^{out} for each j. Let W^q the tensor of quantized weights. Let the quantization basis $\mathcal{B} = (\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}, \mathbf{b_4}, \mathbf{b_5}, \mathbf{b_6}, \mathbf{b_7}, \mathbf{b_8}, \mathbf{b_9})$. Note that in this notation, our basis has dimension 9, which is not the choice we made in the paper. This is not a big deal, since we can choose to fill coordinates with zeros: $\mathbf{b_1} = (b_{1,1}, b_{1,2}, b_{1,3}, 0, 0, 0, 0, 0, 0)$, $\mathbf{b_4} = (0, 0, 0, b_{1,1}, b_{1,2}, b_{1,3}, 0, 0, 0)$, $\mathbf{b_7} = (0, 0, 0, 0, 0, 0, b_{1,1}, b_{1,2}, b_{1,3})$. In this manner, our 3D bases can be expanded in 9D bases.

Usually, in a full precision network, C_j^{out} is computed as :

$$C_j^{out} = \sum_{i=1}^{in} Conv(W_{i,j}; C_i^{in}) \tag{4}$$

In our case, we change the order of operations :

$$C_j^{out} = \sum_{k=1}^{9} \sum_{i=1}^{in} W_{i,j,k}^q Conv(\mathbf{b_k}; C_i^{in}) \tag{5}$$

Since $Conv(\mathbf{b_k}; C_i^{in})$ does not depend on the output channel, we can start by computing $Conv(\mathbf{b_1}; C_i^{in})$ for each i (in convolutions). $\mathbf{b_1}$ can be uniformly quantized using a simple min/max quantizer, therefore we can use low-bit operators to compute these convolutions. Then, we only need to multiply the result by $W_{i,j,1}^q$ (which is an integer) for each j, and store the result in the corresponding output channel. Then, we reiterate the process with $\mathbf{b_2}$, $\mathbf{b_3}$, etc.

With this method, the complexity of the forward pass through Λ is $9 \times in$ low-bit convolutions and $9 \times in \times out$ scalar multiplications and additions.

B Babai algorithm

Babai's nearest plane algorithm is at the core of our quantization mechanism. It relies on a process called "Gram-Schmidt orthogonalization", which is very standard.

Algorithm 1 Nearest plane algorithm

```
1: function BABAI(Basis $\mathcal{B}$, Vector $t$)
2:     $\mathcal{B}^* = GramSchmidt(\mathcal{B})$
3:     $b \leftarrow t$
4:     for $j \in \{n, ..., 1\}$ do
5:         $u_j \leftarrow \frac{<b, b_j^*>}{<b_j^*, b_j^*>}$
6:         $b \leftarrow b - \lfloor u_j \rceil b_j$
7:     end for
8:     return $(\lfloor u_j \rceil)_{1 \leq j \leq n}$ or $x = \sum_{j=1}^{n} \lfloor u_j \rceil b_j = t - b$
9: end function
```

C Random search

The following algorithm 2 is the one we use to search a relevant quanization basis.

Algorithm 2 FindBasis

```
1:  function FINDBASIS(Weight tensor $W$, Basis dimension $dim$, Bitwidth $bits$, Temperatures $T$)
2:      $\mathcal{B} \leftarrow I_{dim}$
3:      for $0 \leq s \leq |T|$ do
4:          $\mathcal{B}' \leftarrow \mathcal{B}' + Sample(\mathcal{G}(0, \sigma = T(s), |\mathcal{B}|))$
5:          if $MCE(W, W^q_{\Lambda_{\mathcal{B}'}}) < MCE(W, W^q_{\Lambda_{\mathcal{B}}})$ then
6:              $\mathcal{B} \leftarrow \mathcal{B}'$
7:          end if
8:      end for
9:      return $\mathcal{B}$
10: end function
```

D Numerical example

In this appendix, we show the LatticeQ quantization and dequantization of an example convolution kernel.

Let us suppose that we have the following quantization basis :

$$\mathcal{B} = (b_1, b_2, b_3) = ((1,1,2), (2,3,1), (1,3,1)) \tag{6}$$

And the following convolution kernel to quantize:

$$F = \begin{pmatrix} 0.2 & 0.8 & 2.1 \\ 1.7 & -0.9 & 3.0 \\ 3.0 & 2.1 & -1.3 \end{pmatrix} \tag{7}$$

F is split into 1×3 vectors: $(0.2, 0.8, 2.1)$, $(1.7, -0.9, 3.0)$ and $(3.0, 2.1, -1.3)$. Each of these vectors goes through Babai's nearest plane algorithm, which yields :

$$\begin{cases} Quant((0.2, 0.8, 2.1)) = (1, -1, 1) \\ Quant((1.7, -0.9, 3.0)) = (2, 1, -2) \\ Quant((3.0, 2.1, -1.3)) = (-1, 3, -2) \end{cases} \tag{8}$$

Therefore:

$$F^q = Quant(F) = \begin{pmatrix} 1 & -1 & 1 \\ 2 & 1 & -2 \\ -1 & 3 & -2 \end{pmatrix} \tag{9}$$

And the result of dequantization is:

$$\hat{F} = Dequant(F^q) = \begin{pmatrix} (b_1 - b_2 + b_3) \\ (2b_1 + b_2 - 2b_3) \\ (-b_1 + 3b_2 - 2b_3) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 2 \\ 2 & -1 & 3 \\ 3 & 2 & -1 \end{pmatrix} \tag{10}$$

E Note on limitations

The main interest of LatticeQ lies in the exploitation of inner correlations between the parameters of neural networks. Correlations have been clearly identified in the convolution kernels of the popular architectures we worked with. Nothing prevents *a priori* from using LatticeQ on RNNs or Transformers, since LatticeQ is nothing but a generalization of a scalar quantization technique. Further study should evaluate whether LatticeQ (or a similar idea) presents, or not, a particular interest at quantizing a wider variety of models.