Differentiable Model Compression via Pseudo Quantization Noise

Alexandre Défossez\textsuperscript{[1]}  
Meta AI, FAIR Team, Paris, France  
defossez@fb.com

Yossi Adi\textsuperscript{*}  
Meta AI, FAIR Team, Tel-Aviv, Israel  
adiyoss@fb.com

Gabriel Synnaeve  
Meta AI, FAIR Team, Paris, France  
gab@fb.com

Reviewed on OpenReview: https://openreview.net/forum?id=DijnKziche

Abstract

We propose DiffQ a differentiable method for model compression for quantizing model parameters without gradient approximations (e.g., Straight Through Estimator). We suggest adding independent pseudo quantization noise to model parameters during training to approximate the effect of a quantization operator. DiffQ is differentiable both with respect to the unquantized weights and the number of bits used. Given a single hyper-parameter balancing between the quantized model size and accuracy, DiffQ optimizes the number of bits used per individual weight or groups of weights, in end-to-end training. We experimentally verify that our method is competitive with STE based quantization techniques on several benchmarks and architectures for image classification, language modeling, and audio source separation. For instance, on the ImageNet dataset, DiffQ compresses a 12 layers transformer-based model by more than a factor of 8, (lower than 4 bits precision per weight on average), with a loss of 0.3\% in model accuracy. Code is available at github.com/facebookresearch/diffq.

1 Introduction

An important factor in the adoption of a deep learning model for real-world applications is how easily it can be pushed to remote devices. It has been observed that larger models usually lead to better performance, for instance with larger ResNets (He et al., 2016) achieving higher accuracies than smaller ones. In response, the community has worked toward smaller, and more efficient models (Tan & Le, 2019). Yet an EfficientNet-B3 is still almost 50MB, a considerable amount if the model is to be included in online applications, or updated with limited network capabilities. For other applications, such as language modeling (Vaswani et al., 2017) or source separation (Défossez et al., 2019), the typical model size is closer to 1GB, ruling out any kind of mobile usage. Efficient model compression is thus important for on device adoption of deep learning models. Thus, we focus in the present work on reducing model size, rather than achieving computational gains.

The simplest method to reduce model size consists in decreasing the number of bits used to encode individual weights. For instance, using 16 bits floating point numbers halves the model size, while retaining a sufficient approximation of the set of real numbers, $\mathbb{R}$, to train with first-order optimization methods (Mickeyevius et al., 2018). When considering lower precision, for instance, 8 or 4 bits, the set of possible values is no longer a good approximation of $\mathbb{R}$, hence preventing the use of first-order optimization methods. Specifically, uniform quantization requires using the \texttt{round} function, which has zero gradients wherever it is differentiable.

Quantization can be done as a post-processing step to regular training. However, errors accumulate in a multiplicative fashion across layers, with a possibly uncontrolled decrease in the model accuracy. Courbariaux et al.
in order to provide a non-zero gradient to the original weights. This allows the model to adapt to quantization during training and reduces the final degradation of performance. However, Fan et al. (2021) noticed instability and bias in the learned weights, as STE is not the true gradient to the function.

The nature of quantization noise has been extensively studied as part of Analog-to-Digital Converters (ADC). In particular, a useful assumption to facilitate the design of post-processing filters for ADC is the independence of the input value and the “Pseudo Quantization Noise” (PQN), as formalized by Widrow et al. (1996). In this work, we show that it also applies to deep learning model quantization, and provides a simple framework in which the output and the quantized model size are both differentiable, without any use of STE. This allows to optimally set the number of bits used per individual weight (or group of weights) to achieve a trade-off between size and accuracy, in a single training and at almost no extra cost. Even when the number of bits to use is fixed, we show that unlike STE, using independent pseudo quantization noise does not introduce bias in the gradient and achieves higher performance. Although PQN has been proposed before for quantization (Baskin et al., 2018a;b), it has never been used on its own without any need for STE or other quantization methods, while achieving state-of-the-art performance.

Our Contribution: (i) With DIFFQ, we propose to use pseudo quantization noise only to approximate quantization at train time, as a differentiable alternative to STE, both with respect to the unquantized weights and number of bits used. (ii) We provide a differentiable model size estimate, so that given a single penalty level $\lambda$, DIFFQ optimizes the number of bits per weight or group of weights to achieve a given trade-off between model size and accuracy. (iii) We provide extensive experimental validation using various models (ConvNets and Transformers) and domains (image classification, language modeling, audio source separation). We demonstrate the efficiency of DIFFQ both in providing small footprint models with comparable performance to the uncompressed ones, together with easy and stable optimization, using only one sensitive hyper-parameter.

2 Related Work

Early network quantization methods focused on low-bitwidth networks such as BinaryNet (Courbariaux et al., 2015; 2016), XOR-Nets (Rastegari et al., 2016), or Ternary networks (Li et al., 2016; Wu et al., 2018). Although these methods produce highly quantized models, their performance is not on par with uncompressed ones. To improve accuracies, higher bitwidth quantization methods were studied (Jung et al., 2019; Zhang et al., 2018a; Mishra et al., 2017). These methods followed the STE approach (Bengio et al., 2013). STE allows the gradients to be backpropagated through the quantizers and, thus, the network weights can be adapted with gradient descent (Courbariaux et al., 2016).

Variational approaches were used to make the categorical distribution over quantized weights differentiable. Louizos et al. (2019) uses a Gumbel-softmax (Jang et al., 2017) but requires 2 hyper-parameters and has no bitwidth tuning. DIFFQ has a single hyper-parameter and supports automatic bitwidth tuning. Shayer et al. (2018) relies on a Central Limit Theorem (CLT) application, however this prevents weights from converging to a deterministic value, which would break the assumptions of the CLT. With DIFFQ, weights are free to converge to any optimal value. Finally, Ullrich et al. (2017) uses a gaussian mixture model trained on top of the weights, adding significant complexity both in terms of code, and computation. In contrast, DIFFQ adds only one penalty term to the loss, optimized along the rest of the model in an end-to-end fashion.

An alternative is to use a smoothed version of the quantization operator, possibly with a trained meta-network (Chen et al., 2019), however as the smoothed operator converges to the true one, gradients will eventually be zero almost everywhere. Gong et al. (2019) use a meta-network to provide gradients despite quantization. However, their implementation for training the meta-network still relies on STE.

Additive noise injection has been studied by Baskin et al. (2018a), although only during the first few epochs, after which STE based approximation is used. This work was extended to non uniform quantization (Baskin et al., 2018b). In contrast, DIFFQ uses only noise injection, and as demonstrated in Results Section, achieves a better accuracy for an equivalent compression level than both methods. Non uniform quantization was also studied by Polino et al. (2018), but without differentiability with respect to the weights, with worse
performance than DiffQ. Additive noise was also studied in the context of image compression (Ballé et al., 2017; Choi et al., 2019) in order to provide a differentiable pseudo-quantization operator. However, those work rely on an explicit estimation of the quantized values entropy, in particular with respect to a distribution of images. This formalism breaks down when having to quantize a single model, not a distribution, and DiffQ uses a simpler approach where the bitwidth is directly tuned. More recently, Park et al. (2022) extended our method for activation quantization.

An important contribution from DiffQ is the automatic tuning of the bitwidth using mixed-precision. Other mixed-precision quantization methods are based on Reinforcement Learning (Wang et al., 2019; Elthakeb et al., 2020; Liu et al., 2021), second-order optimization (Dong et al., 2019; 2020; Yao et al., 2021), and differentiable quantization methods (Uhlích et al., 2020; Wang et al., 2020). Comparing to DiffQ, such methods are more complex (e.g., require plenty of parameter tuning), more computationally heavy, and most importantly based on STE approximations. Wang et al. (2019); Elthakeb et al. (2019) suggested learning a bitwidth assignment policy using reinforcement learning methods. In contrast, our method select bitwidth along training, using only first order optimization. Jain et al. (2019); Esser et al. (2020), and Bhalgat et al. (2020) proposed learning the quantizer step-size or dynamic-range using STE, but do not allow to select the bitwidth. Our experiments show that DiffQ outperforms (Esser et al., 2020) (LSQ) both on most vision and natural language tasks. Uhlích et al. (2020) proposed a re-parametrization that allows to select the bitwidth for each layer through first order optimization, while also relying on STE. The re-parametrization is more complex than the additive noise used in DiffQ, and suffers from the biased gradient of STE. Results suggest that DiffQ achieves similar or better trade-offs between model size and accuracy. Besides, in the present work we experiment setting a bitwidth for individual groups of weights within each layer, rather than layer-wise.

The limitations of STE methods for quantization were first noticed by Liu & Mattina (2019). They recommend using a linear combination of the unquantized and quantized weight, with the gradient flowing only through the unquantized contribution. In a similar spirit, Fan et al. (2021) sample for each layer and iteration whether to use the quantized or unquantized weight. Both methods reduce the bias from STE, but also remove some of the quantization noise during training. In contrast our method allows to keep a full pseudo quantization noise without the STE bias. Liu et al. (2022) proposed the Generalized STE method to deal with gradient instabilities by calculating the expectation of the stochastic quantization during the backward phase. Finally, Nagel et al. (2022) extend the analysis we present in Section 3.3 on the oscillations of weights when using STE and suggest tracking the weight oscillations in order to freeze them when needed, as an ad-hoc solution.

A last line of related work is Product Quantization (PQ) Stock et al. (2019), where code words are being learned to quantize blocks of weights rather than single weights. This method achieves a higher compression level than per-weight quantization but also requires carefully choosing the size of the codebooks for each layer. In contrast, our method requires only choosing a single hyper-parameter to balance between model size and accuracy. Besides, as noted by Fan et al. (2021), per-weight quantization and PQ can be combined. We compare with PQ on vision and language tasks: while PQ can reach smaller model size than DiffQ, it can also suffer from unacceptable accuracy loss, in particular for language modeling.

3 Background

Let us consider a weight vector \( w \in \mathbb{R}^d \), where \( d \in \mathbb{N} \), typically the weights of convolution or linear layer. Each entry of the vector is typically coded over 32 bits with floating-point precision. We aim to reduce the number of possible states to \( 2^B \), where \( B \ll 32 \) is the number of bits of precision. First, we assume \( w_i \in [0, 1] \) for all \( 1 \leq i \leq d \). In practice, one would first normalize \( w \) as

\[
\hat{w} = \frac{w - \min(w)}{\max(w) - \min(w)}.
\]

and provide the tuple \((\min(w), \max(w))\) separately as a 32 bits IEEE float. Given that for typical deep learning models \( d \gg 1 \), storing this range has a negligible cost. For readability, we describe the method for scalar values \( w \in [0, 1] \), however, this can be easily extended to vectors \( w \in \mathbb{R}^d \).
3.1 Uniform quantization

The simplest quantization methods consist of taking $2^B$ points evenly spaced in the range $[0, 1]$ and round each entry of $w$ to the nearest point. One can then store the rounded value by its index, which requires only $B$ bits. Formally, we quantize a number $w \in [0, 1]$ over $B$ bits as

$$
\forall w \in [0, 1], B \in \mathbb{N}_+, Q(w, B) = \text{round} \left( \frac{w \cdot (2^B - 1)}{2^B - 1} \right). 
$$

(1)

While the intuitive definition of quantization is for an integer number of bits, we can extend the previous definitions to a real-valued number of bits $B \in \mathbb{R}_+$. Note that variants of this scheme exist, for instance, symmetric uniform quantization, which enforces that 0 is always exactly represented [Krishnamoorthi, 2018].

3.2 Optimization of the quantized weights

The weight vector $w$ is typically obtained through the process of training a predictor function parameterized by $w$, denoted as $f_w$, to minimize a loss function $L$,

$$
\min_{w \in \mathbb{R}^d} L(f_w),
$$

(2)

where $L(f_w)$ is the empirical risk over a given dataset. The process of quantizing a vector $w$ over $B$ bits introduces a quantization noise $N(w, B) = Q(w, B) - w$, which is unaware of the training objective $L$. Even if $w$ is close to the optimum, $Q(w, B)$ might deteriorate arbitrarily the performance of the predictor.

Thus, given a fixed budget of bits $B$, one would ideally like to minimize the empirical risk when considering the quantization process,

$$
\min_{w \in \mathbb{R}^d} L(f_{Q(w, B)}),
$$

(3)

where $f_{Q(w, B)}$ is the predictor function using the quantized model parameters.

Unfortunately, the gradients of $Q(w, B)$ are zero over its definition domain because of the rounding operation, and as a result, it cannot be optimized using first-order optimization methods such as SGD or Adam [Kingma & Ba, 2015]. One possible solution is to replace the Jacobian of $Q(\cdot, B)$ with the identity matrix during the backward phase, as suggested in the STE method [Bengio et al., 2013]. The STE method was popularized for quantization as the Quantization Aware Training (QAT) technique by Krishnamoorthi (2018).

3.3 The instability and bias in STE

As described by Fan et al. (2021), following the STE approach can cause instability during training and bias in the models’ gradients and weights. As a result, optimization will fail to converge to the optimal value even on simple cases. To demonstrate that, consider the following 1D least-mean-square problem, where $B \in \mathbb{N}_+$, the optimal weight $w_* \in [0, 1]$ such that $Q(w_*, B) \neq w_*$, and $Q(w_*, B) \in (0, 1)$. Given a random variable $X \in \mathbb{R}$ with $\sigma^2 = \mathbb{E}[X^2]$ such that $0 < \sigma^2 < \infty$, we would like to minimize the following using STE based QAT:

$$
\min_{w \in [0,1]} L(w) := \mathbb{E} \left[ \frac{1}{2} (XQ(w, B) - Xw_*)^2 \right].
$$

(4)

We immediately have that the optimum is achieved for $Q(w, B) = Q(w_*, B)$. Let us try to optimize equation 4 using SGD with STE starting from $w_0 = w_*$, with $w_n$ the sequence of iterates. We call $w_-$ and $w_+$ the quantized values just under and above $w_*$, and we assume without loss of generality that $Q(w_*, B) = w_+$. The expected gradient with STE at iteration $n$ is given by

$$
G_n = \sigma^2 (Q(w_n, B) - w_*).
$$

(5)

In particular, $G_0 = \sigma^2 (w_+ - w_*) > 0$, and $G_n$ will stay positive until $Q(w_n, B) = w_-$. At this point, we will have $G_n < 0$, and will stay so until $Q(w_n, B) = w_+$. Thus, we observe that using STE, $Q(w_n, B)$ will oscillate between $w_-$ and $w_+$, while the optimal value is $w_*$. The pattern of oscillation will depend on the learning
rate and relative position of \( w_\ast \) within the segment \([w_-, w_+]\). Taking a smaller step size will reduce the amplitude of the oscillations of \( w_n \), but not of \( Q(w_n, B) \), which is what interests us. Indeed, \( w_n \) oscillations are centered at the boundary \((w_+ + w_-)/2\). We provide one example of those oscillations on Figure 1 with \( w_\ast = 0.11 \), \( B = 4 \), \( X = 1 \) a.s. and a step size of 0.5.

Extrapolating to a model with millions of parameters, at any point in time, a significant fraction of the weights could be quantized to a suboptimal value due to the oscillations implied by the STE method. We conjecture that this behavior explains the oscillations of the accuracy observed when training an EfficientNet-b3 with QAT using 4 bits per weight on ImageNet (see Figure 1(b)). In the following section, we introduce DiffQ, a method based on independent additive pseudo quantization noise, that does not suffer from such a bias, while approximating well enough quantization noise to perform efficient quantization aware training.

4 Method

Pseudo quantization noise. A classical assumption in digital signal processing when working with quantized signals is that the quantization noise is approximated by independents uniform variables over \([-\Delta/2, \Delta/2]\) with \( \Delta = \frac{1}{2^{B-1}} \) the quantization step. This approximation was studied in depth by [Widrow et al. 1996] as Pseudo Quantization Noise (PQN). Following this assumption, we define the pseudo quantization function \( Q \) for all \( x \in \mathbb{R} \) and \( B \in \mathbb{R}_{+}^* \) as

\[
Q(x, B) = x + \frac{\Delta}{2} \cdot U[-1, 1],
\]

with \( U[-1, 1] \) an independent sample from the uniform distribution over \([-1, 1]\). This pseudo quantization function is differentiable with respect to \( x \) and \( B \). Unlike QAT, this differentiability does not require an STE. It also provides a meaningful gradient with respect to the number of bits used \( B \) (extended to be real-valued).

If we look back at the example from Figure 1 using now equation 6 instead of STE, the expected gradients for SGD become

\[
G_n = \mathbb{E} \left[ x \cdot \left( \left( w_n + \frac{\Delta}{2} \cdot U[-1, 1] \right) x - w_\ast x \right) \right] = \sigma^2 (w_n - w_\ast),
\]
which cancels out for \( w_n = w_s \), so that at convergence we indeed have \( \mathbf{Q}(w_n, B) = \mathbf{Q}(w_s, B) \), i.e. the gradient estimate is unbiased and converges to the right solution.

**Mixed precision.** We used a common precision \( B \) for all the entries of the weight vector \( w \). One can instead use different values for different entries. Formally, the entries in \( w \) are grouped by considering \( w \in \mathbb{R}^{g \times d/g} \) with \( g \) the group size and \( d/g \) the number of groups. We can then extend the definition of \( \mathbf{Q}(w, B) \) given by equation \ref{eq:quantization} and equation \ref{eq:quantization_mixed} to use a number of bits \( b_s \) for the group \( s \), with \( b \in \mathbb{R}^{+d/g} \).

**Training objective.** Given \( w \in \mathbb{R}^{g \times d/g} \) with \( g \) groups of \( d/g \) entries, and a number of bits \( b \in \mathbb{N}^2 \), we define the model size, expressed in MegaBytes (1MB = 8 \cdot 2^{20} \) bits

\[
M(b) = \frac{g}{2^{23}} \sum_{s=1}^{d/g} b_s. \tag{8}
\]

A typical objective of quantization is to achieve the best possible performance within a given model size budget or to achieve the smallest model size that reaches a given performance, i.e. we want to minimize with \( b \in \mathbb{N}^{d/g} \), and \( w \in \mathbb{R}^{g \times d/g} \) either,

\[
\min_{w,b} L(f_{\mathbf{Q}(w,b)}), \quad \text{or} \quad \min_{w,b} M(b), \quad \text{s.t.} \quad M(b) \leq m. \tag{9}
\]

We can relax \( b \) to be real valued, and replace \( \mathbf{Q} \) by our differentiable pseudo quantization function \( \mathbf{Q} \). Then, following the exact penalty method \cite{Bertsekas1997}, Section 4.2, \cite{Bertsekas2014}, Chapter 4), there is \( \lambda(m) > 0 \) (or \( \lambda(l) \) for the right hand side problem), such that the left hand size problem is equivalent to

\[
\min_{w,b} L(f_{\mathbf{Q}(w,b)}) + \lambda(m) M(b), \tag{10}
\]

which is fully differentiable with respect to \( w \) and \( b \) and can be optimized with first order optimization.

**Parametrization.** In practice, the number of bits used for each group \( b \in \mathbb{R}^g_{+} \) is obtained from a logit parameter \( l \in \mathbb{R} \), so that we have

\[
b = b_{\min} + \sigma(l)(b_{\max} - b_{\min}), \tag{11}
\]

with \( \sigma \) is the sigmoid function, and \( b_{\min} \) and \( b_{\max} \) the minimal and maximal number of bits to use. The trainable parameter \( l \) is initialized so that \( b = b_{\text{init}} \). We set \( b_{\text{init}} = 8 \).

**Evaluation and noise distribution.** At evaluation time, we round the value \( b \) obtained from equation \ref{eq:quantization_mixed} as \( \hat{b} = \text{round}(b) \) and quantize \( w \) as \( \mathbf{Q}(w, \hat{b}) \). Thus, the amount of quantization noise at evaluation can be larger than the amount of noise injected at train time. We observed that using a noise distribution with larger support, such as Gaussian noise with unit variance (i.e. 3 times the variance of \( \mathcal{U}([-1, 1]) \)), makes the model more robust to this operation. An empirical comparison between uniform and Gaussian noise can be found in Table \ref{table:noise} in the Appendix. Thus in the rest of the paper, we always use Gaussian noise at train time.

**True model size.** The mode size given by equation \ref{eq:quantization_mixed} is used at train time but does not account for part of the true model size. At evaluation time, we represent each weight by the integer obtained from the rounding operation in equation \ref{eq:quantization_mixed}. For each layer in the network, we also store two 32 bits float numbers for the minimum and maximum scale. Finally, the actual value of \( \hat{b} \) must be coded, as it is no longer a fixed constant. For each layer, we compute the maximum value of \( C_s = \log_2(1 + \hat{b}_s - b_{\min}) \) over all groups \( s \in \{1, \ldots, d/g\} \). We encode once the value \( \max(C) \) as an 8-bit integer, and for each group, we encode \( b_s - b_{\min} \) over \( \max(C) \) bits. The true size for one layer, expressed in MegaBytes, is thus given by

\[
\tilde{M}(b) = \frac{1}{2^{23}} \left( 2 \cdot 32 + \frac{d}{g} \max(C) + g \sum_{s=1}^{d/g} b_s \right). \tag{12}
\]
Table 1: Comparison of DiffQ against baselines presented in the Related Work section. Sizes marked with † are reported after Huffman coding, following Polino et al. (2018). Accuracies marked with * are the best rather than last one to match previous practices.

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>Top-1 Acc. (%)</th>
<th>M.S. (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-18</td>
<td>Uncompressed</td>
<td>95.3</td>
<td>42.7</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>UNIQ Baskin et al. (2018b)</td>
<td>89.1</td>
<td>2.7</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>NICE Baskin et al. (2018a)</td>
<td>92.7</td>
<td>2.7</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>DiffQ (Ours)</td>
<td>93.9</td>
<td>2.7</td>
</tr>
<tr>
<td>ResNet-20</td>
<td>Uncompressed</td>
<td>92.7*</td>
<td>1.48</td>
</tr>
<tr>
<td>ResNet-20</td>
<td>DQ Uhlich et al. (2020)</td>
<td>91.4*</td>
<td>0.07</td>
</tr>
<tr>
<td>ResNet-20</td>
<td>DiffQ (Ours)</td>
<td>91.6*</td>
<td>0.06</td>
</tr>
</tbody>
</table>

| CIFAR100 |                          |                |           |
| Wide-ResNet| Uncompressed            | 76.2           | 139.4     |
| Wide-ResNet| DiffQUANT Polino et al. (2018) | 49.3           | 7.9       |
| Wide-ResNet| DiffQ (Ours)            | 75.6           | 4.7       |

| IMAGE NET |                          |                |           |
| ResNet-18 | Uncompressed             | 70.9*          | 44.6      |
| ResNet-18 | META-QUANT Chen et al. (2019) | 60.3           | 1.3       |
| ResNet-18 | DQ Uhlich et al. (2020)  | 70.1*          | 5.4       |
| ResNet-18 | LSQ 4 bits Esser et al. (2020) | 70.7*          | 5.6       |
| ResNet-18 | DiffQ (Ours)             | 71.1*          | 5.3       |
| ResNet-50 | Uncompressed             | 77.1*          | 97.5      |
| ResNet-50 | LSQ 4 bits Esser et al. (2020) | 76.2*          | 12.3      |
| ResNet-50 | LSQ 3 bits Esser et al. (2020) | 75.6*          | 9.3       |
| ResNet-50 | DiffQ (Ours)             | 76.6*          | 10.5      |
| ResNet-50 | DiffQ (Ours)             | 76.3*          | 8.8       |

5 Results

We present experimental results for language modeling, audio source separation, and image classification. We show that DiffQ can often provide a model with comparable performance to the uncompressed one while producing a model with a smaller footprint than the baseline methods (STE based). We provide a finer analysis of different aspects of DiffQ hyper-parameters and their impact on quantized models in next Section. Finally, we discuss limitations of DiffQ in the Limitation Section. Both experimental code, and a generic framework usable with any architecture in just a few lines, is available on our Github [github.com/facebookresearch/diffq](https://github.com/facebookresearch/diffq). All hyper-parameters for optimization and model definition are detailed in the Appendix. In all tables, ↑ (resp. ↓) indicates that highest is best (resp. lowest is best). All results referred to as “QAT” are obtained using the formula given by equation 1 with a layer-wise min-max scaling of the weights. When using DiffQ, we use the same per layer min-max scaling. When also doing activation quantization, we use per-channel min-max scaling of the activations. All DiffQ experiments use Gaussian noise as explained in Section 4.

5.1 Comparison to related work

On Table 1, we compare DiffQ to some of the related work presented in Section 2. Compared with the NICE (Baskin et al. 2018a) and UNIQ (Baskin et al. 2018b) methods, which also rely on additive noise, DiffQ achieves significantly better accuracy for the same model size. We then compare to the differentiable
Table 2: Language modeling results for a 16 layer Transformer trained on Wikitext-103. We also test combining weight and activation quantization. We compared DiffQ to QAT and Quant-Noise (QN) method proposed by Fan et al. (2021) (models with † were trained with a layer-drop of 0.2 [Fan et al. (2019)]. Activations are quantized over 8 bits, with a per-channel scaling.

<table>
<thead>
<tr>
<th>Weights</th>
<th>Activation</th>
<th>PPL ↓</th>
<th>M. S. (MB) ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNCOMPRESSED</td>
<td>-</td>
<td>18.1</td>
<td>942</td>
</tr>
<tr>
<td>8 BITS</td>
<td>8 BITS</td>
<td>18.3</td>
<td>236</td>
</tr>
<tr>
<td>QAT 8BITS</td>
<td>8 BITS</td>
<td>19.7</td>
<td>236</td>
</tr>
<tr>
<td>QAT 4BITS</td>
<td>8 BITS</td>
<td>29.9</td>
<td>118</td>
</tr>
<tr>
<td>LSQ 4 Bits (Esser et al., 2020)</td>
<td>8 BITS</td>
<td>18.9</td>
<td>118</td>
</tr>
<tr>
<td>DiffQ (λ=5, g=16)</td>
<td>8 BITS</td>
<td>18.1</td>
<td>130</td>
</tr>
<tr>
<td>DiffQ (λ=10, g=16)</td>
<td>8 BITS</td>
<td>18.6</td>
<td>113</td>
</tr>
<tr>
<td>UNCOMPRESSED †</td>
<td>-</td>
<td>18.3</td>
<td>942</td>
</tr>
<tr>
<td>QN 8 Bits† Fan et al. (2021)</td>
<td>QN 8 BITS</td>
<td>18.7</td>
<td>236</td>
</tr>
<tr>
<td>QN 4 Bits† Fan et al. (2021)</td>
<td>QN 8 BITS</td>
<td>19.5</td>
<td>118</td>
</tr>
<tr>
<td>PQ† Fan et al. (2021)</td>
<td>-</td>
<td>20.7</td>
<td>38</td>
</tr>
</tbody>
</table>

quantization method by [Polino et al., 2018], which only optimizes the non uniform quantization points, not the pre-quantization weights. Following their practice, we report numbers after Huffman coding. We achieve a model almost half as small, with a gap of 25% in accuracy, proving that optimizing pre-quantization weights is more important than tuning a non uniform quantization grid. Meta-Quant (Chen et al., 2019) achieves smaller model size than DiffQ, with 1 bit per weight, a regime where the PQN assumption breaks down, at the price of losing nearly 10% of accuracy. Finally, compared with two quantization methods: DQ by Uhlich et al. (2020) and LSQ by Esser et al. (2020). When considering DQ, DiffQ achieves slightly smaller model size and better accuracy on ImageNet using ResNet-18, and a 15% smaller model with slightly better accuracy for a Resnet-20 trained on CIFAR-10. Comparing to LSQ, DiffQ achieves better accuracy with smaller model size on ImageNet using both ResNet-18 and ResNet-50. Additional comparison between DiffQ and LSQ for higher compression rates can be on Table B.1 in the Appendix.

5.2 Language Modeling

We trained a 16 layers transformer (Vaswani et al., 2017) based language model on the Wikitext-103 text corpus (Merity et al., 2016), following Baevski & Auli (2019), and using the Fairseq framework (Ott et al., 2019). Results are presented in Table 2. We compare to the Quant-Noise method by Fan et al. (2021), but use a reduced layer-drop (Fan et al., 2019) of 0.1 instead of 0.2. This both improves the baseline, as well as the performance of DiffQ models. For DiffQ, we explicitly set the gradient for the number of bits parameters to zero for all layers that have been dropped. In order to test the compatibility of DiffQ with efficient int8 kernels, we further quantize the activations to 8 bits using PyTorch native support (Paszke et al., 2019).

The transformer model has some tied parameters (e.g. word embedding in the first and pre-softmax layer). It is important to detect such tied parameters with DiffQ. We use a single shared bits parameter when a parameter tensor is reused multiple times, and for each forward, we sample a single pseudo quantization noise per group of shared weights and reuse it appropriately. Failure to do so led to a significant worsening of the performance at validation time.

While QAT breaks down when trying to get to 4 bits precision (perplexity of 29.9), using DiffQ allows to achieve a lower model size (113MB vs. 118 MB for QAT 4 bits) with a perplexity closer to the uncompressed one (18.6, vs. 18.1 uncompressed). We also tried fine-tuning a pre-trained model with LSQ (Esser et al., 2020). While this works better than QAT, LSQ reaches a worst perplexity for a slightly larger model size.

1We used our own LSQ implementation, with only weight quantization, since no official code is available. Comparison with the results reported in Esser et al. (2020) can be found on Table B.1.
Table 3: Music source separation results for the Demucs model \cite{defossez2019demucs}. We report Signal-to-Distortion Ratio (SDR) together with Model Size (M.S.).

<table>
<thead>
<tr>
<th></th>
<th>SDR (dB) ↑</th>
<th>M. S. (MB) ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
<td>6.31</td>
<td>1014</td>
</tr>
<tr>
<td>QAT 4bits</td>
<td>5.99</td>
<td>130</td>
</tr>
<tr>
<td>QAT 5bits</td>
<td>6.27</td>
<td>162</td>
</tr>
<tr>
<td>DiffQ ($\lambda=3e-4$)</td>
<td>6.28</td>
<td>120</td>
</tr>
</tbody>
</table>

than DiffQ (18.9 perplexity for 118 MB). Similarly, Quant-Noise \cite{fan2021quant} improves on QAT but performs worse than DiffQ, even when using more than twice as many bits. With just 4.4 bits per weight on average, DiffQ achieve the same perplexity as the baseline. We also compare to PQ \cite{stock2019pruning}, as reported by \cite{fan2021quant}. While PQ achieves higher compression levels, with just 38MB, its perplexity is the worst of all methods.

5.3 Music Source Separation

We use the Demucs architecture by \cite{defossez2019demucs} with 64 initial hidden channels. The model is trained on the standard MusDB benchmark \cite{rafii2017musdb}, for 180 epochs, and evaluated with the Signal-To-Distortion Ratio (SDR) metric \cite{vincent2006performance}. The unquantized model is 1GB. We compare DiffQ with QAT training with either 5 or 4 bits, with the results presented in Table 3. With 5 bits, QAT is able to replicate almost the same performance as the uncompressed model. When trying to further compress the model to 4 bits per weight, QAT leads to a sharp decrease of the SDR, losing 0.3dB, for a 130MB model. DiffQ achieves a model size of 120MB, with only a drop of 0.03dB of SDR compared to the uncompressed baseline.

5.4 Image Classification

Next, we evaluated three image classification benchmarks: ImageNet \cite{deng2009imagenet}, CIFAR-10 and CIFAR-100 \cite{krizhevsky2009cifar}. For CIFAR-10 and CIFAR-100 results are reported for MobileNet-v1 \cite{howard2017mobilenets}, ResNet-18 \cite{he2016resnets}, and Wide-ResNet with 28x10, depth and width levels respectively \cite{zagoruyko2016wideresnets}. ImageNet results are reported using EfficientNet-B3 \cite{tan2019efficientnet} and DeiT-B \cite{touvron2020deit} models. More details regarding hyper-parameters and augmentations used can be found in the Appendix.

CIFAR10 & CIFAR-100. Results for CIFAR10 and CIFAR100 are depicted in Figures 2(a) and 2(b). We compare DiffQ, QAT and LSQ (without activation quantization) using 2, 3, and 4 bits quantization. Performance of the uncompressed model is additionally presented as an upper-bound. To better understand the effect of the penalty level $\lambda$ on both model size and accuracy, we train models with DiffQ using different penalty levels. Exact results are presented in Table B.2 in the Appendix, together with a detailed analysis.

Results suggest DiffQ models reach comparable performance to the LSQ and outperforms QAT models while producing models with a smaller footprint. When considering 2 bits quantization, QAT is always worse than both LSQ and DiffQ. While LSQ works well for Resnet18, it suffers from large drops in accuracies for MobileNet and WideResNet, failing entirely to train for MobileNet on CIFAR10, despite initialization from a pre-trained model.

ImageNet - DeiT. Results for ImageNet using DeiT-B model are presented in Table 4. We compared DiffQ to QAT when training with 4 and 8 bits. Both QAT with 8 bits and DiffQ reach comparable performance to the uncompressed model, while DiffQ yields a model almost half of the size as QAT, however still bigger than QAT with 4 bits. When we increase $\lambda$, we get a smaller model-size than QAT with 4 bits but with better accuracy levels.
Figure 2: Model accuracy and size on CIFAR10 (a) and CIFAR100 (b) using MobileNet, ResNet-18, and WideResNet (WRN) models for various penalty levels using DiffQ, QAT, LSQ, and the baseline.

Table 4: Image classification results for the ImageNet benchmark. Results are presented for DiffQ and QAT using 4 and 8 bits using the DeiT model \textsuperscript{(Touvron et al., 2020)}. We report Top-1 Accuracy (Acc.) together with Model Size (M.S.).

<table>
<thead>
<tr>
<th></th>
<th>Top-1 Acc. (%) ↑</th>
<th>M.S. (MB) ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
<td>81.8</td>
<td>371.4</td>
</tr>
<tr>
<td>QAT 4bits</td>
<td>79.2</td>
<td>41.7</td>
</tr>
<tr>
<td>QAT 8bits</td>
<td>81.6</td>
<td>82.9</td>
</tr>
<tr>
<td>DiffQ ($\lambda=1e^{-2}$)</td>
<td>82.0</td>
<td>45.7</td>
</tr>
<tr>
<td>DiffQ ($\lambda=0.1$)</td>
<td>81.5</td>
<td>33.02</td>
</tr>
</tbody>
</table>

\textbf{ImageNet - EfficientNet.} We evaluate the performance of DiffQ on the memory-efficient EfficientNet-B3 model. Results are depicted on Figure B.1 (c) as well as in Table B.5 both in the Appendix. Both QAT 8 bits and DiffQ achieves similar accuracy (QAT 81.3 %, DiffQ 81.5%) but with a smaller model size for DiffQ (8.5MB vs. 12MB for QAT). When considering QAT 4 bits, DiffQ produces a smaller model with a significantly better accuracy level (80.8%). For QAT 4, we noticed considerable instability close to the end of the training, see Figure B.1 (b) in the Appendix.

5.5 Analysis

\textbf{Bits Histogram.} Figure 3 presents the weight bitwidth assignment over layer groups for the EfficientNet-B3 \textsuperscript{(Tan & Le, 2019)} and DeiT \textsuperscript{(Touvron et al., 2020)} models trained on ImageNet. The capacity distribution over depth for ConvNets (EfficientNet-B3) and Transformers (DeiT) are different (fp32 shows uncompressed capacity). Notice, that the quantization trends are different too: for the ConvNet, smaller bitwidths are used for deeper layers of the model while large bitwidth is more common in the first layers (except for the last linear layer which seems to need some precision). For the Transformer, this effect of varying quantization by layer is similar but less pronounced, due to the more symmetric nature of the architecture.

\textbf{Fixed bitwidth.} On Table B.4 in the Appendix, we compare QAT to DiffQ using a fixed number of bits, i.e. comparing strictly PQN to STE. On MobileNet, ResNet-18, and WideResNet for both CIFAR10 and
CIFAR100, DiffQ outperforms QAT, with a gap especially noticeable for 2 bits models, a regime where QAT becomes unstable, as we described in previous section.

**Group size.** We additionally evaluate the affect of the group-size, \( g \), on model size and accuracy, by optimizing DiffQ models using \( g \in \{1, 4, 8, \infty \} \). When \( g=\infty \), we use a single group for the entire layer. Results for ResNet-18 using CIFAR-100 are depicted in Figure 1(a) in the Appendix. Interestingly, we observed that increasing \( g \), yields in a smaller model size on the expense of a minor decrease in performance. However, when setting \( g=\infty \) model performance (model size and accuracy) is comparable to \( g=8 \) for this task.

**Runtime overhead and loading time.** Using DiffQ usually increase the training time by some amount. On the language modeling task, the time per batch went from 115ms to 125ms. When training a ResNet18 on CIFAR-10, it increased from 120ms to 150ms. For the Demucs model, it went from 0.9s to 1.1s. However, when training the EfficientNet-b3 model, we observed that the time per batch would nearly double. Thus it seems that for most architectures the training time overhead is limited, although the worst case can be up to twice as slow. At evaluation time, decompressing the Demucs model from its variable bitwidth compact representation takes around 2.81 seconds on a MacBook Pro with 2.4 GHz 8 cores Intel i9 processor.

### 5.6 Limitations

The model size given by equation 12 is obtained with a traditional encoding of the quantized model. However, more efficient coding techniques exist when the entropy of the data is low, such as Huffman coding [Huffman, 1952]. Using the ZLib library, we obtain an estimate of the Huffman compressed model size after quantization. For instance, for the language model described in Table 2, the QAT 8 model gets further compressed from 236MB to 150MB, showing that the entropy of its quantized weight is significantly lower than the maximal one for 8 bits integers. However, the DiffQ model naive size is 113MB, and after compression by ZLib, gets to 122MB. This is a sign that the entropy is close to its maximal value, with ZLib adding only overhead for no gain. In equation 10 we only penalize the naive number of bits used, while asking for the best possible accuracy. In that case, the model maximally use the entropy capabilities for a given number of bits. An interesting line of research would be to replace the model size equation 8 to account for the actual entropy of the data, for instance with differentiable kernel density estimation. We leave that for further research.

Another limitation of DiffQ is that it can make training up to twice as slow, due to the extra parameters to optimize for and the more complex gradient calculation graph. Besides, in order to achieve a specific model size or accuracy, one has to tune the \( \lambda \) penalty parameter.

### 6 Discussion

We presented DiffQ, a novel and simple differentiable method for model quantization via pseudo quantization noise addition to models' parameters. Given a single hyper-parameter that quantifies the desired trade-off between model size and accuracy, DiffQ can optimize the number of bits used for each trainable parameter or group of parameters during model training. We conduct expensive experimental evaluations on various domains using different model architectures. Results suggest that DiffQ is superior to the baseline methods on several benchmarks from various domains. On ImageNet, Wikitext-103, and MusDB, we achieve a model size that is smaller than a 4 bits quantized model, while retaining the same performance as the unquantized baseline. For future work, we consider adapting the model size penalty to account for Huffman encoding, which could allow to further reduce the model size when it is gzipped. Another line of work would be using PQN to improve activation quantization, enabling 4-bits kernels for a larger number of tasks.

### References


Figure 3: We group layers of a given architecture into 11 groups (group 0 being closest to the input, and 10 closest to the output), and report for each group its contribution to the model size. We compare the baseline EfficientNet-B3 (above) and DeiT (below) models (floating point 32 bits) and the quantized models with DIFFQ ($\lambda=5e^{-3}$ for EfficientNet-B3, $\lambda=1e^{-2}$ for DeiT). For quantized model, we also report the distribution over each bitwidth within each group of layers. Scale is logarithmic across layers, and linear inside each one. Finally, “overhead” shows the capacity needed to encode the bitwidth used for each group of weights.


