Not All Bits have Equal Value:
Heterogeneous Precisions via Trainable Noise

Pedro Savarese
TTI-Chicago
savarese@ttic.edu

Xin Yuan
University of Chicago
yuanx@uchicago.edu

Yanjing Li
University of Chicago
yanjingl@uchicago.edu

Michael Maire
University of Chicago
mmaire@uchicago.edu

Abstract

We study the problem of training deep networks while quantizing parameters and activations into low-precision numeric representations, a setting central to reducing energy consumption and inference time of deployed models. We propose a method that learns different precisions, as measured by bits in numeric representations, for different weights in a neural network, yielding a heterogeneous allocation of bits across parameters. Learning precisions occurs alongside learning weight values, using a strategy derived from a novel framework wherein the intractability of optimizing discrete precisions is approximated by training per-parameter noise magnitudes. We broaden this framework to also encompass learning precisions for hidden state activations, simultaneously with weight precisions and values. Our approach exposes the objective of constructing a low-precision inference-efficient model to the entirety of the training process. Experiments show that it finds highly heterogeneous precision assignments for CNNs trained on CIFAR and ImageNet, improving upon previous state-of-the-art quantization methods. Our improvements extend to the challenging scenario of learning reduced-precision GANs.

1 Introduction

Scaling up the size of deep neural networks offers returns to accuracy in tasks across a range of application areas, including computer vision and natural language processing. Expanding network depth from tens of layers [23, 44] to a hundred [13, 52] revolutionized image classification and object detection [9, 14]. Following the 340M parameter BERT model [4], the size of state-of-the-art language models has increased by multiple orders of magnitude: GPT-2 with 1.5B [40], GPT-3 with 175B, and recently Megatron-Turing with 530B parameters [45] – more than a 1000-fold increase in size over BERT. Larger models increase the computational cost of inference, posing challenges to their deployment in real-world applications.

To ameliorate these costs, multiple research avenues have explored methods for constructing more compact deep networks while preserving accuracy comparable to larger models. These include, *e.g.,* architectural building blocks with sparser connectivity [17, 37, 54, 16] or efficient dimensionality reduction stages [20]. Another category of approaches shrinks larger models via pruning [11, 31, 29, 8, 24, 43], reducing precision [12, 19, 51], or binarization [18, 41, 35, 38]. Parallel efforts on neural architecture search [57, 36, 27, 32, 28, 49] optimize coarser aspects of model structure by automating exploration of hyperparameters governing network size and configuration of layers.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Figure 1: **Weight noise as a differentiable proxy for precision.** A learnable magnitude δ scales uniform random noise added to weight w during training. The width of the basin over which it is possible to perturb w without increasing task loss L drives learning of δ. After training, we reduce the bit precision of the numeric representation of w as much as possible, with the constraint of remaining in the $(w - \delta, w + \delta)$ range. ***Left:*** A random perturbation $\tilde{w}$ increases loss, driving a decrease in δ. ***Middle:*** Perturbation leaves loss unchanged, driving an increase in δ. ***Right:*** Noise level δ, and the corresponding implied precision, stabilize when matched to the size of the basin.

Yet, these methods fall short of providing a complete framework for producing efficient networks. Pruning, quantization, and binarization are often implemented as heuristic post-processing transformations that operate on a trained network [12, 25] or require special training procedures [18, 41, 53, 56]. Section 2 provides a broader survey. The lack of modular approaches to optimizing the network structure at this finest level of detail complicates potential combination with neural architecture search techniques for selecting the functional form of layers and building blocks at coarser granularity.

We introduce a principled framework for training compact neural networks, which encompasses pruning, quantization, and binarization. Each is a special case of learning a more general heterogeneous mixed-precision architecture, wherein a set of auxiliary parameters govern the bit precision of each weight in the original network architecture. Our approach learns auxiliary parameters simultaneously with learning the original parameters (weights), with training driven entirely by stochastic gradient descent. In addition to modularity, this strategy provides two crucial benefits:

- Unlike schemes which post-process a trained network, our method makes the entirety of the training process aware of the goal of producing a reduced-precision model. Learning weights in the same context as quantization avoids divergence between training and inference scenarios.

- The auxiliary parameters can freely participate in the definition of the loss being optimized during training. We can craft loss terms that approximate actual costs (*e.g.,* in computation or energy usage) of inference as determined by the current configuration of numeric precisions. Combined with the task-specific loss term, we can choose how to trade-off accuracy and efficiency.

Key to our technical approach is correspondence between a notion of noisy weights and numeric precision. Specifically, during training, we instantiate each network weight as a full-precision value plus uniform random noise of a learnable magnitude. These noise magnitudes are precisely the auxiliary parameters. The more noise that can be injected into the value of a particular weight without increasing loss, the lower the precision required to represent that weight.

Figure 1 illustrates the intuition behind this correspondence. During training, noise magnitudes serve as a differentiable proxy for discrete bit precisions. Upon completion of training, the learned noise magnitudes dictate replacement of noisy weights with fixed, reduced-precision values. Though related to Bayesian neural networks, our formulation differs substantially from prior work on Bayesian approaches to network pruning and quantization [30]. Section 3 provides details, with complete derivations in Appendix A.

Our method is able to allocate precisions under any level of granularity (*e.g.,* per-parameter, per-layer) by sharing the trainable noise magnitude variables across different parameters. Although a per-layer precision scheme is a more natural fit for current GPUs, new hardware designs can leverage more

fine-grained and heterogeneous schemes [34, 46, 21]; hence, we focus on the limit of granularity where each parameter has its own precision. This setting has the potential to achieve significant energy efficiency and run-time advantages over the per-layer setting without sacrificing accuracy, as suggested by our empirical evidence. Future commercially available accelerators, based on the latest hardware research, could make these advantages real.

Results in Section 4 show that, on standard benchmarks such as creating weight-quantized CNNs for image classification on CIFAR-10 and ImageNet, our approach outperforms competitors by achieving higher accuracy using fewer bits per parameter. Further experiments show similar improvements when training GANs. We also show empirically that, when extended to quantize hidden activations of a network, our method is able to find heterogeneous precision schemes and outperforms well-established activation quantization methods.

The source of our advantages may span both the ability to simultaneously account for precision and task objectives within the training dynamics, as well as the ability to learn heterogeneous weight precisions at the finest level of granularity.

2 Related Work

Most approaches in the quantization literature assume that a predefined precision is given prior to training [55, 53, 10, 51, 19, 33, 3, 26, 12, 6], designating the number of bits used to represent each parameter of the network. The focus of these works is typically to circumvent the obstacle posed by the non-differentiability of the quantization function, as this makes first-order methods inapt to train quantized models due to the lack of meaningful gradients.

Figure 2: Multidimensional example where different precisions should be assigned to each parameter due to their distinct perturbation limits. Note the connection between each parameter's perturbation limit, the width of the level curve in the parameter's axis, and the allocated precision for each parameter.

DoReFa [55] updates real-valued weights using gradients w.r.t. their quantized forms (*i.e.,* a straight-through estimator [1]), enabling training over quantized parameter values with gradient descent. LQ-Nets [53], PACT [3], and LSQ [6] introduce further flexibility to quantization by also optimizing the quantization step size, which we refer by *quantization scale* throughout our work, *i.e.,* the real value that each bit corresponds to is also learned.

DSQ [10] proposes a smooth approximation to the quantization mapping, which allows weights to be directly trained with SGD without straight-through estimators. SLB [51] introduces per-parameter auxiliary variables that induce distributions over values that quantized weights can take – these variables are trained with gradient descent and a final quantized model is created by approximating each weight's distribution by a point-mass.

More recently, some works on quantization have explored ways to assign different precisions to parameter groups in a neural network [48, 47, 5, 50] – such methods, however, remain a minority in the quantization literature. DNAS [48] uses neural architecture search to map candidate precisions to different parameter groups. HAQ [47] uses reinforcement learning to learn a quantization policy. HAWQ [5] uses second-order information to allocate precisions to different parameter groups.

Closest to our work is BSQ [50], which introduces new variables that are trained with gradient-based methods, which are then used to allocate precisions throughout the network. It operates by first mapping each real-valued weight to a bitstring variable whose length is chosen a-priori. These variables are then iteratively trained with gradient descent and pruned based on their magnitudes, resulting in a decrease in precision whenever a component (a bit) of the bitstring is removed. Finally, the bitstrings are mapped back to weights, and the quantized model undergoes fine-tuning by adopting straight-through estimators for the quantized weights.

Lastly, Fan et al. [7] improve the performance of quantized networks by randomly selecting a subset of weights to be quantized during training while keeping the remaining parameters in their original

---
Algorithm 1 SMOL

1: **procedure** SMOL ($w \in \mathbb{R}^d, L : \mathbb{R}^d \to \mathbb{R}, \lambda \in \mathbb{R}, p_{init} \in \mathbb{N}$)
2:     Instantiate parameters $s \in \mathbb{R}^d$ and initialize $s_{init} = -\ln\left(2^{p_{init}-1} - 1\right)$
3:     **for** $t = 1 \to T_1$ **do**
4:         Sample $\epsilon^{(t)} \sim \mathcal{U}^d(\pm 1)$
5:         Compute $\mathcal{L}(w, s) = L(w + \sigma(s) \odot \epsilon^{(t)}) + \lambda \| \log_2(1 + e^{-s}) \|_1$ and $\nabla_{w,s}\mathcal{L}(w, s)$
6:         Update w and s using $\nabla_w \mathcal{L}(w, s)$ and $\nabla_s \mathcal{L}(w, s)$
7:         Clip w to $\pm(2 - \sigma(s))$
8:     **end for**
9:     Set $p = 1 + \text{round}(\log_2(1 + e^{-s}))$
10:    ZPA (Optional): If $|w| < |w - q(w, p)|$ then set $p = 0$ (element-wise)
11:    **for** $t = T_1 \to T_2$ **do**
12:        Compute $L(w_q)$ and $\nabla_{w_q} L(w_q)$, where $w_q = q(w, p)$
13:        Update w using $\nabla_{w_q} L(w_q)$ instead of $\nabla_w L(w_q)$
14:    **end for**
15: **end procedure**
---

format. While our approach also uses randomness to improve quantization in some fashion, the form of noise samples, along with how and why they are used, are vastly different.

3 Method

3.1 Allocating Precisions by Optimizing Noise Magnitudes

We consider the problem of allocating a precision p (a positive integer which represents the number of bits) to each weight w of a network under a total precision budget (the total number of bits that can be used to represent the model). If we let $q(w, p)$ denote the quantization of w with p many bits, then our goal is to minimize $L(q(w, p))$ subject to $p \leq C$, where C is our total budget measured in bits and L captures the loss incurred by the weights – typically the cross-entropy loss over a training set.

Although our method is the result of a rigorous sequence of approximations to the objective described above, it admits an intuitive description and motivation that relies on a connection between quantization and perturbations. More specifically, if we assume that a weight w can be perturbed in any direction by at least ϵ without degrading the performance of the network, then we can safely represent w with p bits as long as the quantization error does not exceed ϵ – that is, $|w - q(w, p)| \leq \epsilon$.

In other words, if we know the largest perturbation that each weight admits (its *perturbation limit*) without yielding performance degradation, then we can easily assign a precision to each weight: it suffices to choose the smallest number of bits such that the quantization error falls below the perturbation limit of the corresponding weight.

Our method works by estimating the perturbation limit of each parameter to be quantized, which is achieved by directly optimizing the magnitude of the weight perturbations through a novel loss function. We call our method SMOL : **S**earching for **M**ixed-Precisions by **O**ptimizing **L**imits for Perturbations.

Once the perturbation limit of each weight has been estimated through optimization by our method, we assign per-weight precisions by mapping each perturbation limit to a number of bits.

More specifically, for a model with weight parameters w and training loss L, our method first introduces new parameters s of the same shape as w, and at each iteration t aims to update w, s to decrease an augmented loss function:

$$L\left(w + \sigma(s) \odot \epsilon^{(t)}\right) + \lambda \left\| \log_2(1 + e^{-s}) \right\|_1, \tag{1}$$

where $\epsilon^{(t)}$ has the same size as s and whose components are independently sampled from an uniform distribution over the interval $[-1, +1]$, and λ controls the trade-off between performance and representational cost – a larger value for λ will place more importance on using less bits to represent weights compared to achieving small loss. Note that the noise samples $\epsilon^{(t)}$ are independent across

different iterations, *i.e.,* we draw a new sample $\epsilon^{(t)}$ at each iteration t. Here, the term $\sigma(s)$ represents the perturbation limit estimates of the weights w: the distance between w and $w + \sigma(s) \odot \epsilon^{(k)}$ is at most $\sigma(s)$ since ϵ^k is between -1 and $+1$.

A key aspect of our proposed loss function lies in the fact that it is fully differentiable w.r.t. to s; hence, gradient-based methods like SGD and Adam can be applied off-the-shelf to optimize both the original weights w and the auxiliary variables s. This enables the parameters s to be seen as being part of the model itself, allowing for our method to be easily combined with higher-order optimizers, neural architecture search, and other algorithms that operate on networks. After training, we map the final values of s to precisions via some function $\lceil \cdot \rfloor$ that outputs integers *e.g.,* rounding or truncation:

$$p = 1 + \lceil \log_2(1 + e^{-s}) \rfloor. \tag{2}$$

The resulting tensor p will have the same shape as w, and its components represent the number of bits assigned to each parameter in w. Having allocated a precision to each weight parameter, we can discard the auxiliary variables s and use quantized weights $w_q = q(w, p)$ to compute the model's predictions instead of applying perturbations. This will typically lead to non-negligible changes in the model's activations, which we circumvent by fine-tuning the model by further optimizing w to minimize the training loss. Following prior work, we use straight-through estimators to optimize w since q is non-differentiable, *i.e.,* we set $\nabla_w L = \nabla_{w_q} L$ and perform gradient descent on w.

Lastly, note that inverting the $s \rightarrow p$ mapping offers a way to initialize s given a desired initial precision p_{init}, more specifically we set $s_{init} = -\ln\left(2^{p_{init}-1} - 1\right)$. For example, adopting an initial precision $p_{init} = 8$ results in $s_{init} = -\ln(2^{8-1} - 1) = -\ln(127)$, and the perturbation limit estimate will start as $\sigma(s_{init}) = 2^{1-p_{init}} = 2^{-7}$ for all weights w of the network.

Our method is also applicable if groups of parameters must share the same precision value *e.g.,* the layer-wise setting where all parameters of each layer are represented with the same number of bits. In this case, we assign each parameter group i to a single component s_i of s, and when applying perturbations to the weights in the group we scale all the noise samples by the same scalar $\sigma(s_i)$. At the end of training, we map s_i to a single integer p_i via (2) which is shared across the group.

3.2 Zero Precision Allocation

A fundamental limitation of having each i'th bit map to either $+2^{1-i}$ or -2^{1-i} is that zero weights cannot be represented: the possible values that a 1-bit weight can assume are $\{-1, +1\}$, for a 2-bit weight $\{-1.5, -0.5, +0.5, +1.5\}$, and moreover a quantized weight cannot assume the value 0 regardless of its precision. However, in our setting – where we can assign a different precision to each weight – we can directly re-define the quantization function q to map any w to 0 whenever $p = 0$, hence introducing the notion of *zero-precision weights*. This is analogous to the procedure of assigning zero precision to a filter in BSQ, which leads to the corresponding convolution to be completely skipped when computing a model's outputs.

We propose to assign zero precision to a weight w whenever $|w| \leq |w - q(w, p)|$. Since the quantization function q maps w to the closest value that is representable with p bits, whenever a weight's precision is set to zero due to our proposed strategy, the induced quantization error is guaranteed not to increase. This follows since prior to changing p the quantization error is $|w - q(w, p)|$; but once we set $p = 0$, it becomes $|w - q(w, 0)| = |w - 0| = |w|$, which cannot lead to an increase since the condition is precisely $|w| \leq |w - q(w, p)|$.

For example, $q(w = 0.2, p = 2) = 0.5$ since it is the value in $\{-1.5, -0.5, +0.5, +1.5\}$ that is closest to $w = 0.2$, while $q(w = 0.2, p = 0) = 0$ following our re-definition of q yields a quantization error of $|0.2 - 0| = 0.2$. On the other hand, for $p = 2$ we have $|0.5 - 0.2| = 0.3$. Hence, assigning zero precision in this case not only frees up 2 bits but also decreases the quantization error by 0.1.

One advantage of adopting this procedure to assign zero precisions to weights is that it only changes q and hence the quantized network, therefore not affecting the procedure described in the previous section. In other words, one can allocate precisions to weights by optimizing perturbations as described previously, and then quantize the model separately with and without zero precision allocation.

This results in two quantized models with different precision assignments, hence we can obtain two networks by training the auxiliary variables s only once. The notion of zero precisions also unifies quantization and pruning, since in practice assigning zero precision is equivalent to pruning a weight.

Figure 3: Performance of SMOL and BSQ when quantizing a ResNet-20 trained on CIFAR-10. SMOL-L denotes SMOL with layer-wise precisions.

Figure 4: Performance of SMOL , BSQ, and LQ-Nets when quantizing a ResNet-50 trained on ImageNet. SMOL* denotes SMOL with zero-precision allocation.

4 Results

We evaluate our method in the tasks of quantizing CNNs trained for image classification and generation. For all experiments, we adopt an initial precision $p_{init} = 8$, which corresponds to initializing each component of the new parameter s as $-\ln(2^7 - 1) \approx -4.84$.

Table 1: Performance of different quantization methods on ResNet-20 when trained on CIFAR-10. * denotes results with Zero Precision Allocation.

		ResNet-20	
Method	Precision Granularity	Avg. Bpp $\downarrow$ (Ratio $\uparrow$)	Test Acc. (%)
LQ-FP		32.0 (1.0)	92.1
BSQ-FP		32.0 (1.0)	92.6
LQ-Nets	Network	1.0 (32.0)	90.1
DSQ	Network	1.0 (32.0)	90.2
SLB	Network	1.0 (32.0)	90.6
LQ-Nets	Network	2.0 (16.0)	91.8
SLB	Network	2.0 (16.0)	92.0
SMOL*	Parameter	1.3 (24.6)	91.5
SMOL*	Parameter	1.7 (18.8)	92.6
LQ-Nets	Network	3.0 (10.7)	92.0
BSQ	Layer	2.7 (11.9)	91.9
SMOL-L	Layer	2.4 (13.3)	92.4
SMOL	Parameter	2.1 (15.2)	92.5
SMOL	Parameter	2.5 (12.8)	92.8
BSQ	Layer	3.2 (10.0)	92.8
SMOL-L	Layer	3.2 (10.0)	93.0
SMOL	Parameter	2.8 (11.4)	93.0

We use the floor operation to map real-valued precisions to integers: based on our preliminary experiments this more aggressive rounding operation yields more compact models and rarely results in performance degradation.

For all experiments we train the auxiliary parameters s with Adam [22], using the default learning rate of 10^{-3} and no weight decay – all its other hyperparameters are set to their default values.

We run SMOL with and without Zero Precision Allocation: allowing for zero precisions results in more compact models, and we use SMOL* to denote results with zero precision allocation. When allocating per-layer precisions, we refer to our method as SMOL-L.

To compare different methods we measure the performance and the size of the resulting network. We report the average number of bits assigned to the model's weights (the sum of all precisions divided by the number of weights), which we refer to as 'average bpp', along with the compression ratio relative to a full-precision model, which equals to 32 divided by the average bpp. Appendix D.4 provides preeliminary experiments with Transformers.

4.1 Image Classification on CIFAR-10

We first compare SMOL against different quantization methods on the small-scale CIFAR-10 dataset, adopting different networks to evaluate how aggressively each method can quantize weights without degrading the model's generalization performance. Since prior works typically rely on other methods to quantize activations (*e.g.,* using PACT [3] for activations while applying a new method to the weights), we only consider results with full-precision activations for CIFAR-10 as this isolates each method's performance to its ability to quantize weights, leading to a more direct comparison.

Table 2: Performance of SMOL and BSQ on MobileNetV2 and ShuffleNet when trained on CIFAR-10.

Method	MobileNetV2		ShuffleNet	
	Avg. Bpp $\downarrow$	Test Acc.	Avg. Bpp $\downarrow$	Test Acc.
FP	32.0	94.4	32.0	90.7
BSQ	2.8	94.1	3.4	91.8
SMOL	1.5	94.5	1.8	91.6
SMOL	1.7	94.8	2.9	92.0

We adopt the standard data augmentation procedure of applying random translations and horizontal flips to training images, and train each network for a total of 650 epochs: the precisions are trained with SMOL for the first 350 while the remaining 300 are used to fine-tune the weights while the precisions remain fixed. Note that this training budget assigned to our method is considerably smaller than BSQ's 1000 total epochs which are split between pre-training, precision allocation, and fine-tuning.

Following prior work, we keep the batch normalization parameters in full-precision as they represent a small fraction of the network's total parameters. Like in BSQ, weights of parameterized shortcut connections are also kept in full-precision – namely, shortcuts that consist of 1×1 convolutions followed by normalization in ResNet-20 and MobileNetV2.

To train the weights we use SGD with a momentum of 0.9 and an initial learning rate of 0.1, which is decayed at epochs 250, 500, and 600. We use a batch size of 128 and a weight decay of 10^{-4} for ResNet-20, $4 \cdot 10^{-5}$ for MobileNetV2, and $5 \cdot 10^{-4}$ for ShuffleNet.

ResNet-20 results are given in Table 1: SMOL comfortably outperforms BSQ and other competing methods by offering higher performance at lower precision. Comparing against BSQ's 91.9% accuracy at 2.7 bpp, SMOL provides 0.6% higher accuracy at 0.6 lower bpp (92.5% at 2.1 bpp). With zero-precision allocation (SMOL*), our method outperforms BSQ by 0.7% at 1.0 lower bpp (92.6% at 1.7 bpp), and matches the performance of the full-precision model with a $18.8\times$ compression ratio.

When allocating layer-wise precisions (SMOL-L), we observe a 0.5% higher accuracy at 0.3 lower bpp compared to BSQ, showing that although per-parameter precisions improve efficiency, our method outperforms the state-of-the-art even when constrained to the less flexible, layer-wise setting (more details in Appendix D.1). Figure 3 shows efficiency curves for BSQ and SMOL-L.

Table 2 presents results for MobileNetV2 and ShuffleNet: SMOL also comfortably outperforms BSQ, offering 0.7% higher accuracy at 1.1 lower bpp on MobileNetV2 (94.8% at 1.7 bpp, compared to 94.1% at 2.8 bpp), and 0.2% higher performance at 0.5 lower bpp on ShuffleNet (92.0% at 2.9 bpp, compared to 91.8% at 3.4 bpp), while outperforming the full-precision model in both cases. Additional empirical studies are provided in Appendix D.

4.2 ImageNet

For the large-scale ImageNet classification task, we quantize the ResNet-18 and ResNet-50 models which allow for comparisons against LQ-Nets, DSQ, SLB, and BSQ. For both networks we train the weight parameters with SGD, a momentum of 0.9, a weight decay of 10^{-4} and a batch size of 256 which is distributed across 4 GPUs. We follow LQ-Nets in terms of data augmentation.

We train ResNet-18 for a total of 180 epochs: the first 120 are used for precision training and the last 60 for fine-tuning, and SGD has an initial learning rate of 0.1 which is decayed by 10 at epochs 45, 90, 150, and 165.

For ResNet-50, we start from a pre-trained, full-precision model and train for 100 epochs: allocating the first 60 for precision training and the remaining 40 for fine-tuning. An initial learning rate of 0.01 is decayed by 10 at epoch 30 for the precision training phase, while fine-tuning starts with the same learning rate of 0.01 which is decayed by 10 at epochs 15 and 30. Note that, as in the CIFAR-10 experiments, our training budget is considerably smaller than BSQ's, which also starts from a pre-trained model but has a budget of 180 additional epochs.

Results in Table 3 show that SMOL outperforms competing methods while achieving higher performance than the full-precision baselines. On ResNet-18, our method offers 0.6% higher accuracy than LQ-Nets at 1.7 lower bpp (69.9% at 2.3 bpp compared to 69.3% at 4.0 bpp), while also outperforming the full-precision model by 0.3%.

Table 3: Performance of different quantization methods on ResNet-18 and ResNet-50 models trained on ImageNet. * denotes results with Zero Precision Allocation.

Method	ResNet-18			ResNet-50		
	Average Bpp $\downarrow$	Compression Ratio $\uparrow$	Test Accuracy (%)	Average Bpp $\downarrow$	Compression Ratio $\uparrow$	Test Accuracy (%)
FP	32.0	1.0	69.6	32.0	1.0	76.1
SLB	2.0/4	16.0	67.5			
LQ-Nets	3.0/3	10.7	68.2	3.0/3	10.7	74.2
SMOL*				0.8/4	40.0	74.6
SMOL*	2.3/4	7.6	69.9	2.2/4	14.5	74.9
LQ-Nets	4.0/4	8.0	69.3	4.0/4	8.0	75.1
DSQ	4.0/4	8.0	69.6			
BSQ				3.3/4	9.7	75.2
BSQ				3.7/4	8.6	75.3
SMOL*				4.1/4	7.0	77.1
SMOL	4.5/4	7.1	70.6			
SMOL	4.2/4	7.6	70.4	5.3/4	5.9	76.9

On ResNet-50, SMOL outperforms LQ-Nets at 2.2 lower bpp, with an average number of bits of 0.8 – lower than a binary network. With 4.1 bpp, our method provides a 1.0% improvement over the full-precision baseline, suggesting that the noise injection adopted by our method has additional regularizing effects that can further improve generalization. Performance by average precision plots for BSQ, SMOL, and LQ are given in Figure 4.

4.3 Image Generation with DCGAN

We train a DCGAN [39] model on the CIFAR-10 dataset to perform unconditional image generation at a resolution of 32×32. The generator is implemented as four transposed convolutional layers with batch norm and ReLU activations, while the discriminator consists of four strided convolutional layers with batch norm and LeakyReLU activations. Models are trained with the binary cross-entropy loss using Adam with a learning rate = $2 \cdot 10^{-4}$ and $(\beta_1, \beta_2) = (0.5, 0.999)$. We only quantize the weights of the generator since it is the network used for deployment.

For BSQ, we first train a full-precision DCGAN for 30 epochs, quantize its weights to 8-bits, and conduct 100 epochs of precision learning with a pruning interval of 10 epochs and $\lambda = 2 \cdot 10^{-3}$. The models are then fine-tuned for 30 epochs, with a $10\times$ decayed learning rate. Similarly, for SMOL we train for a total of 160 epochs: 130 for precision training followed by 30 epochs for fine-tuning.

We randomly generate 10,000 samples for all methods, each assessed with Inception Score(IS) [42] and Fréchet Inception Distance (FID) [15]. As shown in Table 4, our method consistently outperforms BSQ, with higher generation quality at lower bpp, demonstrating generalization capability to the challenging task of quantizing GANs.

We drastically improve BSQ's FID from 37.1 at 3.9 bpp to 29.9 at only 3.5 bpp. For BSQ at 2.8 bpp, SMOL* achieves 7.2 better FID with only 1.0 bpp. Even when evaluated at an extremely low bpp of 0.2, our method still generates images with good quality.

4.4 Heterogeneous Precisions for Quantized Activations

In order to extend SMOL to train precisions for hidden activations, we introduce new trainable parameters to estimate the perturbation limit of activation outputs instead of weights. For an activation tensor u, we instantiate a new variable s of the same shape which will be be trained jointly with the network's original parameters.

Table 4: Performance of BSQ and SMOL on image generation on CIFAR-10 with DCGANs.

Method	DCGAN		
	Avg. Bpp ↓	Inception Score ↑	FID ↓
FP (30 epochs)	32.0	4.70	38.6
FP (160 epochs)	32.0	5.67	25.8
BSQ	2.8	4.85	38.3
SMOL*	0.2	4.75	35.6
SMOL*	0.5	5.03	34.1
SMOL*	1.0	5.01	31.1
SMOL	1.7	5.02	34.1
BSQ	3.9	4.95	37.1
SMOL*	3.5	5.29	29.9

Table 5: Performance of PACT and SMOL when quantizing activations of a ResNet-20 trained on CIFAR-10.

Method	ResNet-20	
	Avg. Bpa ↓	Test Acc. (%)
FP	32.0	91.6
PACT	2.0	89.2
SMOL-A	1.8	90.3
PACT	3.0	91.4
SMOL-A	2.9	91.8
SMOL-A	2.5	91.4
PACT	5.0	91.6

Similarly to the weight quantization case described in Section 3, at each training iteration t we sample a tensor $\epsilon^{(t)}$ with the same shape as u, where each component is drawn uniformly from $[-1, +1]$, and generate perturbed activations $u + \frac{M}{2} \cdot \sigma(s) \odot \epsilon^{(t)}$, which are used in place of u throughout the next layers of the network. The scalar M denotes the range of activation function used to compute u, *i.e.*, $M = 1$ for sigmoid and $M = 2$ for tanh activations, and is used to match the magnitudes of the activations and its perturbations. Since ReLU activations are unbounded, we use PACT [3] which clips values to $[0, \alpha]$ where α is a new trainable parameter – this yields $M = \alpha$. Once s has been trained, we map its components to integer precisions which are used to quantize the activations u.

Table 5 shows the performance of a ResNet-20 trained on CIFAR-10 whose activations are quantized by PACT and SMOL (referred as SMOL-A): our method provides off-the-shelf improvements over PACT, offering higher accuracy at lower average bits per activation (bpa). All networks are trained for 200 epochs, following the training settings from Section 4.1.

4.5 Computational Efficiency

A key question is whether per-parameter precisions can result in inference energy cost reductions. The power required to multiply a 2-bit and 3-bit weight with a 4-bit activation is $2.41\times$ and $3.83\times$ higher than what is required for a 1-bit weight; and the latency is $1.91\times$ and $2.10\times$ higher, respectively. These numbers are estimated using ripple-carry adder based multiplier designs (which are suitable for low-precision operations) for the corresponding precision settings [2]. For the accumulation operations, we assume that the power and latency values are the same for different precision settings.

We take as an example the last convolutional layer of ResNet-20, for which BSQ assigns 2 bits for all parameters while our method assigns 1, 2, and 3 bits to 74%, 24.5%, and 1.5% of the parameters, respectively. In this case, the layer with precisions assigned by SMOL requires only 57.5% of computation power while improving latency by 36% compared to BSQ, which amounts to an energy cost reduction (power x latency) of 62.7%. Note that in real hardware designs, additional control overheads, *e.g.,* around 25% [34], are required to perform fine-grained mixed-precision operations. Although estimates, these suggest that a fine-grained precision allocation scheme can result in significant energy savings on hardware designed to support the corresponding arithmetic operations.

5 Discussion

With advances in hardware design and significant overparameterization of recent deep networks, learning compact model representations that can be leveraged by specialized hardware has become a key problem – one that is not restricted to the machine learning and systems communities.

Our work goes a step further in terms of network quantization by proposing SMOL, a modular and principled method that is able to learn heterogeneous precisions at the finest granularity, where different weights of the same model can be represented in distinct precisions regardless of how the weights are connected in the network's topology.

Figure 5: Image generations with a DCGAN trained on CIFAR-10, quantized with BSQ and SMOL.

Moreover, SMOL operates by introducing auxiliary variables to the target model, thus being agnostic to the network's topology, the choice of optimizer, or the underlying training setting, and hence it can be easily combined with other algorithms such as neural architecture search.

Our procedure to allocate precisions lies in the optimization of a trade-off between the model's performance and its cost – a cost function that can be chosen according to the underlying application or the user's concern, *e.g.,* computational or environmental cost.

Finally, our method offers an unification of pruning, binarization, and quantization, while at the same time achieving state-of-the-art results in image classification and image generation with quantized models. Our results show that adopting heterogeneous precisions yields highly compact models that not only perform well, but can also result in significant computational advantages when deployed on specialized hardware.

Acknowledgments and Disclosure of Funding

This work was supported in part by the University of Chicago CERES Center. The authors have no competing interests.

References

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432*, 2013.

[2] S.D. Brown and Z.G. Vranesic. *Fundamentals of Digital Logic with VHDL Design*. McGraw-Hill series in electrical and computer engineering. McGraw-Hill, 2009.

[3] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. PACT: Parameterized clipping activation for quantized neural networks. *arXiv:1805.06085*, 2018.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

[5] Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. HAWQ: Hessian AWare quantization of neural networks with mixed-precision. In *ICCV*, 2019.

[6] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *ICLR*, 2020.

[7] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. Training with quantization noise for extreme model compression. In *ICLR*, 2021.

[8] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[9] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[10] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *ICCV*, 2019.

[11] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *NeurIPS*, 2015.

[12] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, 2017.

[15] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

[17] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q. Weinberger. CondenseNet: An efficient DenseNet using learned group convolutions. In *CVPR*, 2018.

[18] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NeurIPS*, 2016.

[19] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.*, 18:187:1–187:30, 2017.

[20] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *arXiv:1602.07360*, 2016.

[21] Zhaoming Jiang, Zhuoran Song, Xiaoyao Liang, and Naifeng Jing. PRArch: Pattern-based reconfigurable architecture for deep neural network acceleration. In *HPCC*, 2020.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[24] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham M. Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *ICML*, 2020.

[25] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In *ICLR*, 2017.

[26] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *NeurIPS*, 2017.

[27] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[28] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019.

[29] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[30] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *NeurIPS*, 2017.

[31] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *ICLR*, 2018.

[32] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018.

[33] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv:1603.01025*, 2016.

[34] Eunhyeok Park, Dongyoung Kim, and Sungjoo Yoo. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *ISCA*, 2018.

[35] Jorn W. T. Peters and Max Welling. Probabilistic binary neural networks. *arXiv:1809.03368*, 2018.

[36] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

[37] Ameya Prabhu, Girish Varma, and Anoop M. Namboodiri. Deep expander networks: Efficient deep networks from graph theory. In *ECCV*, 2018.

[38] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In *CVPR*, 2020.

[39] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

[40] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[41] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[42] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *NeurIPS*, 2016.

[43] Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. In *NeurIPS*, 2020.

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[45] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zheng, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. Using DeepSpeed and Megatron to train Megatron-Turing NLG 530B, a large-scale generative language model. *arXiv:2201.11990*, 2022.

[46] Zhuoran Song, Bangqi Fu, Feiyang Wu, Zhaoming Jiang, Li Jiang, Naifeng Jing, and Xiaoyao Liang. DRQ: Dynamic region-based quantization for deep neural network acceleration. In *ISCA*, 2020.

[47] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization with mixed precision. In *CVPR*, 2019.

[48] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of ConvNets via differentiable neural architecture search. *arXiv:1812.00090*, 2018.

[49] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In *ICLR*, 2019.

[50] Huanrui Yang, Lin Duan, Yiran Chen, and Hai Li. BSQ: Exploring bit-level sparsity for mixed-precision neural network quantization. *arXiv:2102.10462*, 2021.

[51] Zhaohui Yang, Yunhe Wang, Kai Han, Chunjing Xu, Chao Xu, Dacheng Tao, and Chang Xu. Searching for low-bit weights in quantized neural networks. In *NeurIPS*, 2020.

[52] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[53] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *ECCV*, 2018.

[54] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

[55] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv:1606.06160*, 2016.

[56] Bohan Zhuang, Lingqiao Liu, Mingkui Tan, Chunhua Shen, and Ian D. Reid. Training quantized neural networks with a full-precision auxiliary module. In *CVPR*, 2020.

[57] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Appendix

We present additional experiments and discussions in the following sections. Appendix A provides the derivation of our precision-training framework, including steps and approximations taken to arrive at our proposed regularized loss. Appendix B presents a discussion on how to accurately compare methods that use different bit values, which is crucial when comparing our method against BSQ [50] as it adopts unsigned bits and a mutual notion of precision must be established. Appendix C has additional details on our main experiments, including how λ was chosen and how activations were quantized. Finally, Appendix D offers additional experimental results with the goal of studying key questions regarding our method, including whether it is able to assign precisions in a layer-wise fashion instead of per-parameter.

A Framework Derivation

A.1 Preliminaries

We start by defining a function family $\mathcal{F}$ parameterized by $\Theta \subseteq \mathbb{R}^d$: for any $\theta \in \Theta$, we have a function $f_\theta \in \mathcal{F}$, $f_\theta : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ that is induced by θ. In our practical setting, the variables θ denote a configuration for the parameters of neural network while f_θ represents the function that the network implements under such parameter values.

For the sake of convenience, we directly define a loss functional $L : \Theta \to \mathbb{R}$ that measures the (average) instant loss of a function $f_\theta \in \mathcal{F}$ induced by some $\theta \in \Theta$. For example, L can be explicitly given by $L(\theta) = \frac{1}{N} \sum_{i=1}^N \ell(f_\theta(x_i), y_i)$, where $((x_i, y_i))_{i=1}^N$ is a fixed dataset and ℓ is an instant loss (*i.e.,* squared loss or cross-entropy).

Now, consider the problem of finding a parameter setting $\theta \in \Theta$ where each component θ_i is represented by exactly p_i many (signed) bits given a bit-value map $(v_1, v_2, \dots)$ that assigns a scalar $v_j \in \mathbb{R}$ to each position $j \in \mathbb{N}$ in a bit string. In other words, we have

$$\theta_i = \sum_{j=1}^{p_i} b_{i,j} \cdot v_j \, ,$$

where $b_{i,j} \in \{\pm 1\}$ is the j'th bit used to represent θ_i and $(v_j)_{j=1}^P$ is the bit-value map. W.l.o.g. we will assume that the bit-value map is (countably) infinite and that $\mathbb{V}^d \subseteq \Theta$, where $\mathbb{V} = \{\sum_{j=1}^\infty b_j \cdot v_j \mid b_j \in \{\pm 1\}\}$ is the set of all possible values that each parameter can assume.

We can then formalize the problem of finding a set of bit precisions $p \in \mathbb{N}^d$ and bit strings $b \in \{\pm 1\}^{d \times \infty}$ such that the total precision $\sum_{i=1}^d p_i$ is minimized and the parameter setting $\theta \in \Theta$ induced by b achieves 'small loss':

Definition A.1. (Mixed Precision Problem)

$$\min_{p \in \mathbb{N}^d, \, b \in \{\pm 1\}^{d \times \infty}} \quad \sum_{i=1}^d p_i \qquad \text{s.t.} \quad L(V(b, p)) \leq L^* + \delta \, , \; V(b_i, p_i) = \sum_{j=1}^{p_i} b_{i,j} \cdot v_j \qquad (3)$$

where $L^* = \min_{\theta \in \mathbb{V}^d} L(\theta)$ is the smallest achievable loss by $\mathcal{F}$ over $\mathbb{V}^d$ and $\delta \in \mathbb{R}^+$ is a scalar that captures tolerance in terms of suboptimality. Here, V is the function that maps bit strings $b \in \{\pm 1\}^{d \times \infty}$ and precision values $p \in \mathbb{N}^d$ to parameter values $\theta \in \mathbb{V}^d$ element-wise:

$$V(b_i, p_i) = \sum_{j=1}^{p_i} b_{i,j} \cdot v_j \, .$$

The above problem poses a few obstacles: first, it involves a search over multiple binary values for each of the d parameters of the network; second, the derivative w.r.t. each integer precision p_i is undefined, making it ill-suited for gradient-based methods.

Alternatively, we can directly optimize $\theta \in \mathbb{V}^d$ under the constraint that its finite-precision representation is used to evaluate the loss L:

Definition A.2. (Mixed Quantization Problem)

$$\min_{p \in \mathbb{N}^d, \ \theta \in \mathbb{V}^d} \sum_{i=1}^{d} p_i \qquad \text{s.t.} \quad L(Q(\theta, p)) \leq L^* + \delta \tag{4}$$

where Q maps $\theta \in \mathbb{V}$ to a finite-precision representation of itself with exactly p many bits. To define Q precisely we will first introduce a notion of inverse for V (a 'notion' since V is not generally bijective).

For any $\theta_i \in \mathbb{V}$, we let $V^{-1}(\theta_i)$ denote a setting (b_i, p_i), with $p_i \in \mathbb{N}$ and $b_i \in \{\pm 1\}^\infty$ such that $V(b_i, p_i) = \theta_i$ and, moreover, for any (b'_i, p'_i) such that $V(b'_i, p'_i) = \theta_i$, it follows that $p'_i \geq p_i$. That is, V^{-1} maps each element $\theta_i \in \mathbb{V}$ to a precision-bitstring tuple that perfectly represents θ_i using the least number of bits. We also define $V_p^{-1}(\theta_i) = p_i$ and $V_b^{-1}(\theta_i) = b_i$ where, as before, $V^{-1}(\theta_i) = (b_i, p_i)$.

Then, we can define Q by $Q(\theta_i, p_i) = \sum_{j=1}^{p_i} V_b^{-1}(\theta_i)_j \cdot v_j$, *i.e.*, it is the value induced by the *shortest* bit string that represents θ, when considering only its first p_i bits. Note that this yields $Q(\theta_i, p_i) = V(V_b^{-1}(\theta_i), p_i)$, which will be a key property in the proofs below.

While it might be tempting to see Q as a quantization mapping, it not always acts as such. For example, consider the value map given by $v_j = 2^{1-j}$, where $\theta_i = 1$ can be represented by $p^{(1)} = 1$, $b_i^{(1)} = (1, 1, 1, 1, \dots)$, by $p^{(2)} = \infty$, $b_i^{(2)} = (1, -1, 1, 1, \dots)$, and so on. In this case, we could let $V^{-1}(1) = (b^{(1)}, p^{(1)})$, but not $V^{-1}(1) = (b^{(2)}, p^{(2)})$. For the former, we would have $Q(1, 2) = \frac{3}{2}$, which is *not* a quantization – in the common sense – of $\theta_i = 1$ under the considered value map.

With this definition of Q we can show that the optimization problems in Equations (3) and (4) are equivalent (under V and V^{-1}).

Lemma A.3. *For any setting (b, p) that satisfies $L(V(b, p)) \leq L^* + \delta$ in Equation (3), we have that $(\theta = V(b, p), p)$ satisfies $L(Q(\theta, p)) \leq L^* + \delta$ in Equation (4).*

Proof. By definition, we have:

$$\begin{aligned} Q(\theta, p) &= Q(V(b, p), p) \\ &= V\left(V_b^{-1}(V(b, p)), p\right) \\ &= V(b, p) \\ &= \theta \end{aligned} \tag{5}$$

Therefore, the parameter setting induced by $V(p, b)$ in Equation (3) will be the same as the one induced by $Q(\theta, p)$ in Equation (4), and hence the losses will be the same and satisfiability follows. $\square$

Lemma A.4. *For any setting (θ, p) that is a minimizer of the problem in Equation (4), we have that $(b = V_b^{-1}(\theta), p)$ satisfies $L(V(b, p)) \leq L^* + \delta$ in Equation (3).*

Proof. Since (θ, p) minimizes $\sum_{i=1}^{d} p_i$ by assumption, we have that no θ_i can be represented with less than p_i many bits and hence $p = V_b^{-1}(\theta)$. Therefore

$$\begin{aligned} V(b, p) &= V(V_b^{-1}(\theta), p) \\ &= V(V_b^{-1}(\theta), V_p^{-1}(\theta)) \\ &= V(V^{-1}(\theta)) \\ &= \theta \end{aligned} \tag{6}$$

Therefore, the parameter setting induced by $Q(\theta, p) = \theta$ in Equation (4) will be the same as the one induced by $V(b, p)$ in Equation (3), and hence the losses will be the same and satisfiability follows. $\square$

With the two Lemmas we can prove that the two problems are indeed equivalent:

Corollary A.5. *The optimization problems in Equations* (3) *and* (4) *are equivalent under* V *and* V^{-1}. *That is, for any* (b, p) *that minimizes* (3), *we have that* $(\theta = V(b, p), p)$ *minimizes* (4), *and for every* (θ, p) *that minimizes* (4) *we have that* $(b = V_b^{-1}(\theta), p)$ *minimizes* (3). *Moreover, the induced function* f_θ *will be the same in all cases.*

Proof.

- (4) → (3): assume for the sake of contradiction that $(b = V_b^{-1}(\theta), p)$ is suboptimal in (3), then there exists a satisfiable (b', p') with $\|p'\| < \|p\|$, but from Lemma A.3 it follows that $(\theta = V(b', p'), p')$ is satisfiable in (4), which contradicts the optimality of (θ, p).

- (3) → (4): assume for the sake of contradiction that $(\theta = V(b, p), p)$ is suboptimal in (4), then there exists a satisfiable (θ', p') with $\|p'\| < \|p\|$, but from Lemma A.4 it follows that $(b = V_b^{-1}(\theta', p'), p')$ is satisfiable in (3), which contradicts the optimality of (b, p).

$\square$

A.2 Stochastic Approximation

We now present a sequence of steps, involving approximations, that yield our final method.

1. We can first write
$$Q(\theta, p) = \theta - (\theta - Q(\theta, p)) = \theta + R(\theta, p)$$
which allows us to show that
$$R(\theta_i, p_i) = \left(\sum_{j=p+1}^{V_p^{-1}(\theta_i)} V_b^{-1}(\theta_i)_j \cdot v_j \right) + \left(\sum_{j=V_p^{-1}(\theta_i)}^{p+1} V_b^{-1}(\theta_i)_j \cdot v_j \right)$$
by relying on the definition of Q and V^{-1} alone.

2. Next, we consider the following assumption: if (θ_i, p_i) is satisfiable, then (θ_i, p_i') is satisfiable for any $p_i' > p_i$ (i.e. assigning more bits to a parameter preserves its satisfiability). This assumption can be more simply framed as
$$L(Q(\theta, p)) < L^* + \delta \implies L(Q(\theta, p')) < L^* + \delta, \quad \forall i \; p_i' > p_i \,,$$
and is the core assumption for our method.

3. Using the definition of R and w.l.o.g. denoting $Q(\theta, p)$ simply by θ', we get from the above equation that
$$L(\theta') < L^* + \delta \implies L(\theta' + R(\theta', p')) < L^* + \delta, \quad \forall i \; p_i' > V_p^{-1}(\theta_i')$$
i.e., using the trinary system, this means that if a bitstring $(1, 1, 0, 0, 0, \dots)$ is satisfiable, then any bitstring $(1, 1, \pm 1, \pm 1, \pm 1, \dots)$ will also be satisfiable. Now, we can frame the following optimization problem which has infinitely many constraints:
$$\min_{p \in \mathbb{N}^d, \, \theta \in \mathbb{V}^d} \quad \sum_{i=1}^d p_i \qquad \text{s.t.} \quad L(\theta + R(\theta, p')) \leq L^* + \delta, \quad \forall i \; p_i' > p_i = V_p^{-1}(\theta_i) \quad (7)$$

4. Note that $R(\theta_i, p_i')$ includes $V_b^{-1}(\theta_i)$ which admits any configuration for all bits after $V_b^{-1}(\theta_i)$-pth one due to the fact that Q operates via performs truncation. We further assume that $\{R(\theta_i, p_i') \mid p_i' > V_p^{-1}(\theta_i)\}$ is dense in some interval $I(\theta_i, p_i) = [l, u] \subset \mathbb{V}$.

   We then frame a stochastic approximation for the problem above, which is designed by sampling from the infinitely many constraints that were introduced:
$$\min_{p \in \mathbb{N}^d, \, \theta \in \mathbb{V}^d} \quad \sum_{i=1}^d p_i \qquad \text{s.t.} \quad \frac{1}{K} \sum_{k=1}^K L(\theta + \epsilon^{(k)}) \leq L^* + \delta, \quad \forall (i, k), \; \epsilon_i^{(k)} \sim \mathbb{P}(I(\theta_i, p_i))$$
$$(8)$$

5. Next, we pose the constraints using Lagrange multipliers to yield a regularized objective:

$$\min_{p\in\mathbb{N}^d,\ \theta\in\mathbb{V}^d} \quad \frac{1}{K}\sum_{k=1}^{K} L(\theta + \epsilon^{(k)}) + \lambda\|p\|, \quad \forall(i,k),\ \epsilon_i^{(k)} \sim \mathbb{P}(I(\theta_i, p_i)) \tag{9}$$

6. Note that the derivation so far is agnostic to the underlying value map. We now assume the value map is given by $v_j = 2^{1-j}$, where we get $\mathbb{P}(I(\theta_i, p_i)) = \mathcal{U}(\pm 2^{1-p_i})$ under a bit distribution of $\mathcal{U}(\pm 1)$ for all bits after the $V_p^{-1}(\theta_i)$-th one:

$$\min_{p\in\mathbb{N}^d,\ \theta\in\mathbb{V}^d} \quad \frac{1}{K}\sum_{k=1}^{K} L(\theta + \epsilon^{(k)}) + \lambda\|p\|, \quad \forall k,\ \epsilon_i^{(k)} \sim \mathcal{U}^d(\pm 2^{1-p_i}) \tag{10}$$

7. To allow for gradients to flow through the sampling procedure, we apply the reparameterization trick and a continuous relaxation on p:

$$\min_{p\in\mathbb{R}_+^d,\ \theta\in\mathbb{V}^d} \quad \frac{1}{K}\sum_{k=1}^{K} L(\theta + 2^{1-p}\odot\epsilon^{(k)}) + \lambda\|p\|, \quad \forall k,\ \epsilon^{(k)} \sim \mathcal{U}^d(\pm 1) \tag{11}$$

8. Finally, we reparameterize both p and θ to yield variables that can be optimized over the reals – although not necessary, this results in a procedure that does not require variables to be projected to a domain after each update.
   We define $s = \sigma^{-1}(2^{1-p})$, $\theta = \tau(w)$, and frame the problem as optimization over (s, w) instead, with the advantage that, unlike (p, θ), they both belong to $\mathbb{R}^d$:

$$\min_{s,w\in\mathbb{R}^d} \quad \frac{1}{K}\sum_{k=1}^{K} L(\tau(w) + \sigma(s)\odot\epsilon^{(k)}) + \lambda\|\log_2(1 + e^{-s})\|, \quad \forall k,\ \epsilon^{(k)} \sim \mathcal{U}^d(\pm 1) \tag{12}$$

where ideally we have $\tau : \mathbb{R} \to \mathbb{V}$ be such that $\tau(\mathbb{R}) = \mathbb{V}$.

This concludes the derivation of our regularized loss. Note that instead of adopting the reparameterization $\theta = \tau(w)$, we can directly optimize θ while guaranteeing that $\theta \in \mathbb{V}$: this can be easily achieved by projecting the parameters θ to an interval, for example to $[-1, +1]$ when using signed bit representations.

B Bit Counting: Implicit vs Explicit Sign Bits

The process of quantizing data is heavily dictated by whether bits are chosen to be signed (taking values $\{+1, -1\}$) or unsigned (values $\{0, 1\}$), and comparing methods that use different settings might require adopting notions of 'precision' that diverge from the ones used in their original papers.

Most methods on quantizing neural networks [53, 51, 10] – ours included – adopt signed bits, which is often done implicitly by splitting the $[-1, +1]$ interval uniformly in 2^p bins to extract the values that quantized data can assume. In this case, assigning a value of 2^{1-i} to each i-th bit results in a bitstring $(b_1, b_2, \ldots, b_p)$ being mapped to $\sum_{i=1}^{p} b_i \cdot 2^{1-i}$, a variable that can assume 2^p different values uniformly distributed over an interval centered at the origin *i.e.,* $[-M, +M]$ for positive M.

On the other hand, adopting unsigned bits results in the possible values for $\sum_{i=1}^{p} b_i \cdot 2^{1-i}$ to be uniformly distributed over a $[0, M']$ interval instead (for positive M'), hence bitstrings can only be mapped to non-negative values. An additional 'sign bit' can be adopted to allow the representation of negative numbers: in particular, points are encoded to unsigned bitstrings of the form $(b_s, b_1, \ldots, b_{p'})$, which are then mapped to $s(b_s) \cdot \sum_{i=1}^{p'} b_i \cdot 2^{1-i}$ where $s(b_s) = +1$ if $b_s = 1$ and $s(b_s) = -1$ if $b_s = 0$: note that such mapping allows for a total of $2^{p'+1}$ different outputs. In this case the term 'precision' p can be used to either denote $p' + 1$ (the total number of bits used to encode each point) or p' (number of bits except for the 'sign bit' b_s). However, when comparing a method that relies on such representation to one that adopts signed bits instead, we must count the additional 'sign bit' *i.e.,* $p = p' + 1$, as in this case both methods can represent exactly 2^p values given a precision of p bits.

When considering results reported by BSQ [50], we account for an additional 'sign bit' that is implicitly used when quantizing full-precision weights with bitstrings. More specifically, once a

model has been fine-tuned, BSQ converts its full-precision weights $W_s \in \mathbb{R}^d$ to binary tensors $W_p^{(0)}, \ldots, W_p^{(n-1)}, W_n^{(0)}, \ldots, W_n^{(n-1)}$, each $\in \{0, 1\}^d$, where $(W_p^{(b)})_{b=0}^{n-1}$ encodes the positive components of W_s while $(W_n^{(b)})_{b=0}^{n-1}$ encodes the absolute value of its negative components.

These binary tensors are then mapped to quantized weights $W_{final} = \frac{1}{2^n-1} \sum_{b=0}^{n-1} (W_p^{(b)} - W_n^{(b)}) 2^{n-b-1}$, whose components can take values $\frac{1}{2^n-1}\{0, 1, \ldots, 2^n - 1\}$ by setting $W_n^{(b)} = 0$ and enumerating all configurations for $(W_p^{(b)})_{b=0}^{n-1}$, while setting $W_p^{(b)} = 0$ and considering all configurations for $(W_n^{(b)})_{b=0}^{n-1}$ yields the values $\frac{-1}{2^n-1}\{0, 1, \ldots, 2^n - 1\}$: in total, we have that elements of W_{final} can assume $2^{n+1} - 1$ distinct values ($2^n - 1$ positive, $2^n - 1$ negative, and 0). An implicit 'sign bit' is also used in the forward computation of quantized models, which maps weights W to $\frac{1}{2^n-1} \cdot \text{sign}(W) \odot \lceil |W| \cdot (2^n - 1) \rceil$: the implicit representation of n bits does not encode the sign of the weights, which is recovered after quantizing its absolute value over 2^n bins.

To summarize, BSQ uses unsigned bit representations to encode weights but without an explicit 'sign bit': the weight signs are either bypassed and hence not encoded (as in the forward pass), or are implicitly encoded by allocating two bitstrings for each weight (as when converting full-precision weights to bitstrings)[1] – where which one is used is dictated the the weight's sign.

To account for the mismatch in how many values can be represented with p many bits by BSQ compared to our method, we account for one more bit when reporting BSQ results, which designates the implicit 'sign bit' in addition to the unsigned n-bit representations.

C Experimental Details

Values for λ: When training ResNet-20 networks on CIFAR-10, we used $\lambda \in \{10^{-6}, 7 \cdot 10^{-7}, 5 \cdot 10^{-7}\}$ when running SMOL, where larger values for λ result in less bits per parameter: $\lambda = 10^{-6}$ resulted in a network with 2.1 bpp while $\lambda = 5 \cdot 10^{-7}$ yielded a model with 2.8 bpp. For MobileNetV2 we used $\lambda \in \{10^{-7}, 2 \cdot 10^{-7}\}$, while for ShuffleNet we adopted $\lambda \in \{5 \cdot 10^{-8}, 7 \cdot 10^{-10}\}$

For our ImageNet experiments, we adopted $\{10^{-6}, 10^{-8}\}$ and $\{10^{-7}, 10^{-8}\}$ as values for λ when training ResNet-18 and ResNet-50 networks, respectively. Different values for λ were required (compared to the ones we used for CIFAR-10) due to the ImageNet ResNets having considerably more parameters than the ResNet-20 model we adopted for CIFAR-10: note that our regularizer is a sum over all precisions instead of, for example, an average.

Finally, for image generation we used $\lambda \in \{2 \cdot 10^{-5}, 10^{-5}, 5 \cdot 10^{-6}, 2 \cdot 10^{-6}, 10^{-6}\}$, where the most compact model (0.2 bpp) was achieved with $\lambda = 2 \cdot 10^{-5}$ along with zero precision allocation. In most cases we saw a decrease of between 1 and 2 bpp when enabling zero precision allocation: training a DCGAN with $\lambda = 10^{-5}$ results in 0.5 bpp with zero precision allocation and 1.7 without it.

Quantized activations for ImageNet Experiments: We follow BSQ [50] and replace all ReLU modules by ReLU6 throughout the network, resulting in outputs constrained to the $[0, 6]$ interval. Activations are not quantized prior to fine-tuning *i.e.,* precision training with SMOL uses full-precision activations, which are only quantized once precisions have been allocated and fine-tuning has started. All our experiments that quantize activations consider 4-bit precisions *i.e.,* the activation tensors are quantized using $2^4 = 16$ different values.

Also following BSQ, we use straight-through estimators to allow for gradient flow through the activation quantization procedure. The quantization values are uniform over half of the activation range – typically $[0, 3]$, which results in the full-precision activations being quantized to one of the 16 values in the set $\{0, \frac{1}{5}, \frac{2}{5}, \ldots, \frac{14}{5}, \frac{15}{5}\}$.

---

[1]These refer to *bit_STE* and *finetune_to_bit* in the official github repository: https://github.com/yanghr/BSQ/tree/9f09b6284d264ce07445e42ddd18764c7b3d0405

Figure 6: Layer-wise precisions allocated by SMOL-L on ResNet-20 models trained on CIFAR-10. The x-axis denotes the layer indices: leftmost points denote earlier layers while rightmost denote layers in the later stages of ResNet, which are closer to the final fully-connected layer.

D   Additional Experimental Questions

D.1   What Precision does SMOL Allocate to Each Layer?

In Section 4.1 we present results for SMOL when training layer-wise precisions on a ResNet-20 trained on CIFAR-10 (SMOL-L in Table 1), where higher performance under lower bpp is achieved compared to BSQ.

Figure 6 shows the precisions allocated to each layer of the ResNet-20 under three different values for λ: in all cases, later convolutional layers are assigned low precisions compared to earlier stages of the network. The first convolution, along with the fully-connected layer at the end of the network, are assigned significantly higher precision than other layers.

D.2   More Precision Training or More Fine-tuning?

In our main experiments we allocate roughly half of the training budget to precision training and the other half to fine-tuning *i.e.,* further optimization of the weight parameters while applying quantization and adopting straight-through estimators. Here we show how the performance and size (measured in average number of bits per parameter) of the final model behaves under different ratios for the number of epochs allocated to precision training over the total training budget.

We use the settings described in Section 4.1, training models with $\lambda = 7 \cdot 10^{-7}$ but changing what how many of the total 650 epochs are used for precision training – the remaining ones are always allocated to fine-tuning the model.

Results are shown in Figure 7: there is significant change in the model's bpp when the fraction of epochs allocated to precision training is lower than $\frac{250}{650}$, indicating that SMOL requires between 200 and 250 epochs to fully optimize the auxiliary variables s. For ratios larger than $\frac{250}{650}$ our method yields networks with similar bits-per-parameter (around 2.5 bpp), but the test accuracy starts to decrease as the ratio increases due less epochs being used to fine-tune the model. The best performance is achieved by allocating 350 epochs for precision training, which results in the remaining 300 epochs being assigned for fine-tuning.

Figure 7: Performance and average precision of ResNet-20 trained on CIFAR-10 with SMOL when allocating different number of epochs to precision training versus fine-tuning. Curve shows increases of 50 epochs over a total of 650 epochs.

Table 6: Comparison between a ResNet-20 trained on CIFAR-10 with SMOL and its performance when re-trained, either when the per-weight precision assignments are maintained or randomly permuted (shuffled).

Method	ResNet-20		
	Average Bpp $\downarrow$	Compression Ratio ($\times$) $\uparrow$	Test Accuracy (%)
Original (SMOL)	2.5	12.8	92.8
Re-trained with same precisions	2.5	12.8	91.5
Re-trained with shuffled precisions	2.5	12.8	89.3

D.3 Does SMOL Learn Structured Precision Maps?

To evaluate whether the per-parameter precision assignments learned by SMOL have an underlying structure or not, we set up an additional experiment which aims to re-train and evaluate a network after destroying any possible structure in its precision map. In particular, we start from a ResNet-20 model that has been fully trained and fine-tuned by SMOL, and proceed to re-train it under two different settings.

For the first, we re-initialize the weight parameters (using a different random seed) but fully preserve the precision assignments generated by SMOL *i.e.,* each i-th weight w_i with precision p_i is randomly re-initialized ($w_i \sim \mathcal{D}_{init}$), but p_i is kept the same. As for the second setting, we also apply a random permutation on the precision tensor of each layer *i.e.,* weights w_i and w_j of the same layer have their precisions p_i and p_j swapped – with the goal of destroying any possible structure that the tensor p might have.

We then re-train the two networks using the same training settings adopted in Section 4.1, but skipping precision training and hence allocating a budget of 300 epochs to weight training under quantization and straight-through estimators, akin to fine-tuning.

Table 6 and Figure 8 present results: randomly permuting (shuffling) the precisions yields in significantly lower performance (89.3% compared to 91.5%), suggesting that the precision assignments generated by SMOL do have an underlying structure, which can be some form of per-layer organization *i.e.,* allocating high or low precision to all elements in the same convolutional filter, or cross-layer structure *i.e.,* feature maps (layer outputs) have low or high precision connections to both the previous and the next layer. We leave further the characterization of such structure to future work.

Figure 8: Generalization performance when re-training networks, with and without randomly permuted (shuffled) precisions. The model whose precisions have been shuffled has visibly lower performance across all stages of training and achieves significantly lower final performance.

D.4 Can SMOL Quantize Transformer Models?

Since our results in Section 4 focus on CNN-like networks like ResNets and MobileNetV2, an immediate question is whether SMOL is able to efficiently quantize architectures of different families, for example Transformers which do not use convolutions and whose main component is the attention mechanism.

To answer this question, we run preliminary experiments where we train a standard encoder-decoder Transformer with 6 attention blocks on the neural machine translation IWSLT'14 German to English task, where we quantize all weights except for the layer normalization parameters. A total of 50 epochs were used for training: for SMOL, the first 30 were used to train the precisions and the remaining 20 for fine-tuning.

Table 7: Performance of SMOL when quantizing a Transformer on IWSLT'14.

	Transformer	
Method	Avg. Bpp ↓	BLEU Score ↑
FP	32.0	34.9
SMOL	3.9	34.7
SMOL	5.8	34.9

Results in 7 show that SMOL can match the performance of the full-precision model at 5.8 bpp, which amounts to a $5.5\times$ compression ratio. At a lower bpp of 3.9 ($8.2\times$ ratio), the BLEU score is lower by only 0.2. Note that we use the same training budget that is commonly employed when training a full-precision model for this task, which is likely suboptimal for quantization. Therefore, although promising, these results are very preliminary and can likely be improved by simply training for more epochs.