

A APPENDIX

A.1 1:2 MINIMUM VARIANCE UNBIASED ESTIMATOR- FULL

For a block $a \triangleq [a_1, a_2]$, one entry needs to be pruned:

$$\theta(\mathbf{a}) = \begin{cases} [v_1, 0] & , \text{w.p. } p \\ [0, v_2] & , \text{w.p. } 1 - p \end{cases} \quad (22)$$

We wish to design an unbiased estimator for this pruning method where $E[\theta(\mathbf{a})] = [a_1, a_2]$:

$$E[\theta(\mathbf{a})] = p \cdot [v_1, 0] + (1 - p) \cdot [0, v_2] = [a_1, a_2] \quad (23)$$

Therefore, the following constraints apply:

$$\begin{aligned} p \cdot v_1 &= a_1 & \Rightarrow & p = \frac{a_1}{v_1} \\ (1 - p) \cdot v_2 &= a_2 & \Rightarrow & v_2 = \frac{a_2}{1 - \frac{a_1}{v_1}} \end{aligned} \quad (24)$$

To find an unbiased estimator which minimizes the total block variance, we first calculate the variance for each element in the block $\theta(\mathbf{a}) = [\theta_1, \theta_2]$ as follows:

$$\begin{aligned} \text{Var}[\theta_1] &= E[\theta_1^2] - E^2(\theta_1) = v_1^2 \cdot p - a_1^2 \\ \text{Var}[\theta_2] &= E[\theta_2^2] - E^2[\theta_2] = v_2^2 \cdot (1 - p) - a_2^2 \end{aligned} \quad (25)$$

Then, the total variance of a block is the sum of its element variances:

$$\text{Var}_B[\theta] = \text{Var}[\theta_1] + \text{Var}[\theta_2] = v_1^2 p - a_1^2 + v_2^2 - v_2^2 p - a_2^2 \quad (26)$$

Putting Equation (24) into Equation (26) yields the following expression for the total variance in the block, that depends on v_1

$$\begin{aligned} \text{Var}_B[\theta] &= v_1^2 \cdot \frac{a_1}{v_1} - a_1^2 + \frac{a_2^2}{\left(1 - \frac{a_1}{v_1}\right)^2} \cdot \left(1 - \frac{a_1}{v_1}\right) - a_2^2 \\ &= v_1 \cdot a_1 - a_1^2 + \frac{a_2^2}{1 - \frac{a_1}{v_1}} - a_2^2 \\ &= v_1 \cdot a_1 - a_1^2 + \frac{a_2^2 \cdot v_1}{v_1 - a_1} - a_2^2 \end{aligned} \quad (27)$$

By finding the derivative of Equation (27) with respect to v_1 and setting it to zero we get the following equation:

$$\frac{\partial \text{Var}_B[\theta]}{\partial v_1} = \frac{a_2^2}{v_1 - a_1} - \frac{a_2^2 v_1}{(v_1 - a_1)^2} + a_1 = 0 \quad (28)$$

The solution to Equation (28) gives two possible solutions for v_1 , but only one is feasible (the first):

$$\begin{aligned} v_1 &= a_1 + a_2 \\ v_1 &= a_1 - a_2 \end{aligned} \quad (29)$$

Therefore, the following unbiased estimator has the lowest variance of all unbiased estimators

$$\theta(\mathbf{a}) = \begin{cases} [\text{sign}(a_1) \cdot (|a_1| + |a_2|), 0] & , \text{w.p. } \frac{|a_1|}{|a_1| + |a_2|} \\ [0, \text{sign}(a_2) \cdot (|a_1| + |a_2|)] & , \text{w.p. } \frac{|a_2|}{|a_1| + |a_2|} \end{cases} \quad (30)$$

Substituting into Equation (27) the optimal solution $v_q = a_1 + a_2$, the optimal method outlined in Equation (30) has a variance of $2a_1 a_2$. Therefore, since the method is unbiased it results with a mean-square-error of

$$MSE = \text{Bias}^2 + \text{Var} = 0 + 2a_1 a_2 = 2a_1 a_2 \quad (31)$$

In Table 2 we compare different 1:2 structured sparsity on the neural gradients on ResNet18 Cifar10 dataset. We show that although the proposed MVUE method doesn't minimize the MSE, it gets the best results.

A.2 MINIMUM-VARIANCE UNBIASED ALGORITHM FOR 2:4

Given a block $[a_1, a_2, a_3, a_4]$, assume without loss of generality that $a_4 > a_3 > a_2 > a_1$. We need to choose two elements a_i, a_j from the block with a probability $p_{i,j}$. We have three cases:

A.2.1 CASE 1: $a_4 \leq 2a_1 + a_3$:

$$\begin{aligned}
 p_{12} &= 0 \\
 p_{13} &= \frac{2a_1 + a_3 - a_4}{2(a_1 + a_2 + a_3 + a_4)} \\
 p_{14} &= \frac{2a_1 - a_3 + a_4}{2(a_1 + a_2 + a_3 + a_4)} \\
 p_{23} &= \frac{2a_2 + a_3 - a_4}{2(a_1 + a_2 + a_3 + a_4)} \\
 p_{24} &= \frac{2a_2 - a_3 + a_4}{2(a_1 + a_2 + a_3 + a_4)} \\
 p_{34} &= \frac{-a_1 - a_2 + a_3 + a_4}{a_1 + a_2 + a_3 + a_4}
 \end{aligned} \tag{32}$$

Using the above solution, one can verify that all probabilities are between 0 and 1, normalized, and adhere to the optimality conditions outlined in Equation (16). Here is an example for p_1 and p_2 :

$$\frac{p_1}{p_2} = \frac{p_{12} + p_{13} + p_{14}}{p_{12} + p_{23} + p_{24}} = \frac{a_1}{a_2} \tag{33}$$

A.2.2 CASE 2: $2a_1 + a_3 \leq a_4 \leq a_1 + a_2 + a_3$:

$$\begin{aligned}
 p_{12} &= 0 \\
 p_{13} &= 0 \\
 p_{14} &= \frac{2a_1}{a_1 + a_2 + a_3 + a_4} \\
 p_{23} &= \frac{a_1 + a_2 + a_3 - a_4}{a_1 + a_2 + a_3 + a_4} \\
 p_{24} &= \frac{-a_1 + a_2 - a_3 + a_4}{a_1 + a_2 + a_3 + a_4} \\
 p_{34} &= \frac{-a_1 - a_2 + a_3 + a_4}{a_1 + a_2 + a_3 + a_4}
 \end{aligned} \tag{34}$$

A.2.3 CASE 3: $a_4 \geq a_1 + a_2 + a_3$:

Choose a_4 with probability 1, and also choose one a_i from $\{a_1, a_2, a_3\}$ with probability

$$\tilde{p}_i = \frac{a_i}{a_1 + a_2 + a_3} \tag{35}$$

A.3 A COMPARISON OF OPTIMAL 1:2 AND OPTIMAL 2:4 - PROOF

To prove this claim we first find $\text{Var}[\theta_{2:4}(\mathbf{a})]$ by assigning a probability $p_i = \frac{a_i}{a_1 + a_2 + a_3 + a_4}$ to each element i in Equation 12:

$$\begin{aligned}
 \text{Var}[\theta(\mathbf{a})] &= \frac{a_1}{2} (a_1 + a_2 + a_3 + a_4) - a_1^2 + \frac{a_2}{2} (a_1 + a_2 + a_3 + a_4) - a_2^2 \\
 &\quad + \frac{a_3}{2} (a_1 + a_2 + a_3 + a_4) - a_3^2 + \frac{a_4}{2} (a_1 + a_2 + a_3 + a_4) - a_4^2 \\
 &= a_1 \cdot a_2 + a_1 \cdot a_3 + a_1 \cdot a_4 + a_2 \cdot a_3 + a_2 \cdot a_4 + a_3 \cdot a_4 \\
 &\quad - \frac{a_1^2}{2} - \frac{a_2^2}{2} - \frac{a_3^2}{2} - \frac{a_4^2}{2}
 \end{aligned} \tag{36}$$

Table 7: ResNet18 (R18) and ResNet50 (R50) top-1 accuracy on ImageNet dataset with greedy N:M fine-grained sparse activations.

Method	R18 top-1	R50 top-1
Baseline	70.6%	76.6%
4:8 activations	70.6%	76.45%

the left hand side of Equation 19 is given by Equation 36 and its right-hand side equals to $2a_1a_2 + 2a_3a_4$. Let D be the difference between the left-handside of Equation 19 and its right handside. To prove our claim we need to show that D is negative:

$$\begin{aligned}
D &= a_1a_2 + a_1a_3 + a_1a_4 + a_2a_3 + a_2a_4 + a_3a_4 \\
&\quad - \frac{a_1^2}{2} - \frac{a_2^2}{2} - \frac{a_3^2}{2} - \frac{a_4^2}{2} - (2a_1a_2 + 2a_3a_4) \\
&= \underbrace{\left(-\frac{a_1^2}{2} + a_1a_4 - \frac{a_4^2}{2}\right)}_{-0.5(a_1-a_4)^2} + \underbrace{\left(-\frac{a_2^2}{2} + a_2a_3 - \frac{a_3^2}{2}\right)}_{-0.5(a_2-a_3)^2} + \\
&\quad + \underbrace{a_4a_2 + a_1a_3 - a_1a_2 - a_3a_4}_{-(a_1-a_4) \cdot (a_2-a_3)}
\end{aligned} \tag{37}$$

Let $A \triangleq (a_1 - a_4)$ and $B \triangleq (a_2 - a_3)$ then we have that

$$D = -\frac{A^2}{2} - A \cdot B - \frac{B^2}{2} = -\frac{1}{2}(A + B)^2 < 0 \tag{38}$$

Our claim is thus proven.

A.4 EXPERIMENTS DETAILS

In all our experiments we use 8 GPU GeForce Titan Xp or GeForce RTX 2080 Ti.

N:M structured sparsity on the neural gradients In the vision models, we used the standard pre-processing of ImageNet ILSVRC2012 dataset. We train for 90 epochs, use an initial learning rate of 0.1 with a $\frac{1}{10}$ decay at epochs 30,60,80. We use standard SGD with momentum of 0.9 and weight decay of $1e-4$. The batch size used is 256. Following the DNNs quantization conventions (Banner et al., 2018; Nahshan et al., 2019; Choi et al., 2018b) we kept the first and last layer (FC) at higher precision.

Accelerating all training phases In these experiments, we used the exact same experiment setting as Hubara et al. (2021).

A.5 ACCELERATING INFERENCE

N:M sparsity on the activations In Table 7 we experimented with greedy N:M fine-grained sparse activations on ResNet18 and ResNet50 over ImageNet, wherein for each block of size M we keep the M-N larger elements. Note that in CNNs the activations memory footprint is much larger than the weights footprint (especially for the first set of layers), so in term of memory reduction activations pruning is more effective than weights pruning. Throughout our experiments, we did not change the training regime and used the sparse activations from scratch. As can be seen in Figure 3 applying only fine-grained sparse activations results in notable accuracy degradation for both training and validation. However, a simple fix is to apply ReLU before the fine-grained sparse activations; this results in on-par accuracy over ResNet18 and ResNet50.

N:M sparsity on the weights and activations Inference requires compressing only the weights and activations. Thus, we experimented in Table 8 with greedy N:M fine-grained sparsity of weight and activations. To compete with the latest inference acceleration results based on quantization-aware

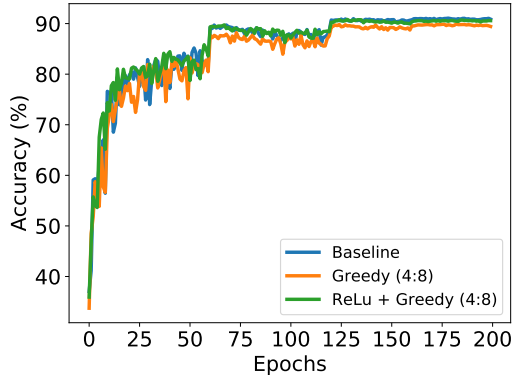


Figure 3: Top-1 accuracy on ResNet18 CIFAR10 dataset with 4:8 fine-grained sparsity on the activations. ReLU + greedy refers to applying ReLU and then the structured sparsity, while greedy does not include the ReLU function. As can be seen, applying only N:M structured sparsity as the activation function leads to accuracy degradation.

techniques, we further quantize the weights and activation to 4-bit using the SAWB method (Choi et al., 2018a). Since on average we have $N/2$ MAC operations for each block, the BOPS (Wang et al., 2020) reduction is equivalent to 2-bit quantization. We experimented with N:M fine-grained sparse activations and weights using ResNet18 over ImageNet using two training schemes:

Scheme A is similar to Zhou et al. (2021) regime, in which the fine-grained sparsity is applied from scratch. Here we additionally applied 4-bit asymmetric quantization for both weights and activations. *Scheme B* is similar to Nvidia (2020) regime and has three phases: (a) train with sparse activation full precision model; (b) set a mask for the weights; and (c) train a sparse weights and activation model while using 4bit quantization-aware training.

In all our experiments we kept last fully connected layer dense and in full precision. We used 2:4 structure for the weights and 4:8 structure for the activations. Notice that 4:8 sparsity was previously showed in Hubara et al. (2021) as a feasible method. In Table 10 we showed the overhead of the 4:8 sparsity in comparison to 2:4 and standard ReLU activation.

Table 8: Inference acceleration of ResNet18 on ImageNet dataset. We quantize the weights and activations to 4-bit and apply in both greedy N:M fine-grained sparsity getting an equivalent of 2-bit inference. We compare our results with different 2-bit quantization-aware training methods (PACT (Choi et al., 2018b), QIL (Jung et al., 2019), LSQ (Esser et al., 2019)): we achieve comparable results in Scheme A, and better results in Scheme B.

Method	Baseline	2:4 activations	Scheme A (ours)	Scheme B (ours)	PACT 2-bit	QIL 2-bit	LSQ 2-bit
Accuracy (%)	70.6	70.6	65.62	67.22	64.4	65.7	66.9

B PRUNING OVERHEAD

The pruning overhead depend on the hardware at hand the gradients numerical precision and the user implementation. Nevertheless in this section we would try to roughly access the overhead both quantitatively and by measuring it.

B.1 N:M STRUCTURED PRUNING ON THE NEURAL GRADIENTS

In Table 3 we measured the overhead of the proposed MVUE 1:2, MVUE 2:4 and Approx MVUE 2:4 over regular training of ResNet50. Importantly, the measurements were done the FP32 over

Titan1080 where done using current unoptimized code, so there is much room for improvement. To explain this in more detail, we focus on the overhead of the two parts of the proposed algorithms:

1. Random sampling according to the probabilities p_i .
2. Calculating the probabilities p_i .

Part 1: the overhead of random sampling A possible implementation of the random sampling is with stochastic-rounding (SR) (Crocì et al., 2022), where the probabilities are pre-computed. In Table 9 we show the small overhead of SR in software with a non-optimizer CUDA kernel. This overhead can be reduced more in modern deep learning accelerators (Tes; Hab; Gra) which include SR in hardware.

Notice that we can define sampling using random i.i.d samples: Assume for a block $a \triangleq [a_1, a_2]$, according to the proposed MVUE 1:2 we should sample a_1 with probability p_1 and sample a_2 with probability $1 - p_1$, then we can implement this sampling by random variable $\varepsilon \sim U[0, 1]$ and if $\varepsilon < p_1$ we choose a_1 , otherwise we choose a_2 . This can be easily extended to the proposed approx-MVUE 2:4. Now, it is possible to drastically reduce the overhead of random sampling if we re-use the random samples ε (i.e., sample a new ε only every 50 iterations). In Figure 4 we show a comparison of the proposed approx-MVUE 2:4 and the same algorithm where we re-use the random samples every 50 iterations. As can be seen sampling every 50 iterations does not affect the accuracy. As we can notice, we are able to reduce the overhead of the part (1) of the proposed algorithm without affecting the accuracy. Since the overhead of part (1) can be significantly reduced by amortization, we next focus on part (2).

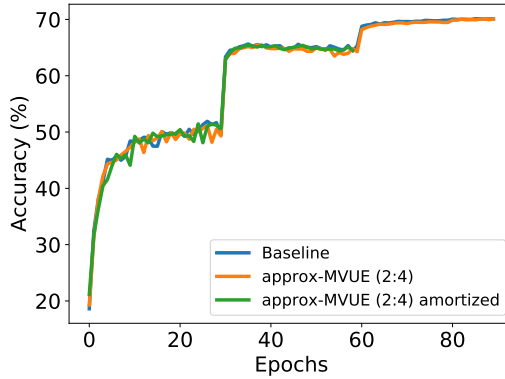


Figure 4: Top-1 validation accuracy of ResNet18 over ImageNet dataset with the proposed approx-MVUE 2:4 and the amortization of the random samples every 50 iteration to reduce the overhead. Notice that both methods achieved similar accuracy.

Table 9: Measured time of a non-optimized CUDA kernel implementation of stochastic-rounding for different sizes of a random tensor. For each tensor size we repeat the experiment 100 times and average the results.

Tensor size	Stochastic rounding [micro sec]	Round-to-nearest [micro sec]
10^3	2.64	2.55
10^4	2.64	2.6
10^5	3.94	3.89
10^6	4.07	4.01
10^7	6.89	6.78
10^8	9.83	9.77

Part 2: Quantitative estimation of the probability calculation overhead The practical overhead depends on many details, such as the numerical precision, the hardware architecture,

and the pruning implementation. Therefore, we will estimate the overhead of the probability calculation by using the simplified measure of counting the number of mathematical operations in Algorithm 1 per block size $M = 4$ or $M = 2$. A simple derivation leads to 22 additions, 10 divisions, 4 multiplications, and 5 comparisons for $M = 4$ (and one addition, 3 divisions, and one comparison for $M = 2$). To sum all operation's overhead, we followed Horowitz (2014); Mach et al. (2020) and converted each operation overhead to a single multiplication's overhead by assuming: 1 div = 4 mul = 16 add/subtract/compare. Converting and summing the operations result in 50.75 mul for $M = 4$ and 12.5 mul for $M = 2$. Note that this is only a rough estimation based on the operations power consumption. Now we will analyze the potential gain and overhead for Linear and Convolution layers assuming 50% sparsity.

First, we focus on the simpler case of a Linear Layer. There, given activation $h \in R^{B \times C_I}$ and gradients $g \in R^{B \times C_O}$, where B is the batch size and C_I, C_O are the number of input and output channels respectively, the weight update would be $\Delta \in R^{C_I \times C_O}$. Thus, the total number of multiplication operations reduction would be $C_I B C_O / 2$. Since we prune only the gradients, pruning overhead, in this case, is $50.75 B C_O / 4$ for $M = 4$. Comparing gain and overhead suggests that when $C_I > 26$ for $M = 4$ (and $C_I > 7$ for $M > 2$), the gain surpasses the overhead. Similar derivation can be done for convolution.

Second, we calculate the overhead of the Convolution Layer. Given activation $A \in R^{B \times C_I \times H \times W}$ and gradients $G \in R^{B \times C_O \times H \times W}$, where H, W are the spatial dimensions, the update would be $\Delta = R^{C_I \times C_O \times k \times k}$ (K is the kernel size). Thus, the total number of multiplication operations reduction is $k^2 C_I B H W C_O / 2$. Similarly the pruning overhead in this case is $50.75 H W B C_O / 4$ for $M = 4$. Comparing gain and overhead suggest that for convolution layers if $C_I > 26/k^2$ for $M = 4$ (and $C_I > 7/k^2$ for $M > 2$), then the gain surpass the overhead. So, if for example $k = 3$, as is common in many filters, we have a net gain if $C_I \geq 3$ for $M = 4$ (and for any C_I for $M = 2$).

Algorithm 1 set mask index and scale

Require: $X \in \mathbb{R}^{K \times M}, R \in \mathbb{R}^{K \times M}, M$

```

out ← abs(out)
s ← sum(outs, dim = 1)                                ▷ 3 or 1 additions (for M = 4 or 2)
v ← outs/s                                              ▷ 4 or 2 divisions (for M = 4 or 2)
vv ← v/(1 - v)                                          ▷ 4 subtractions and 4 divisions (for M = 4, not required for M = 2)
q ← sum(vv, dim = 1)                                    ▷ 3 additions (for M = 4, not required for M = 2)
p ← v · (1 + q - vv)  ▷ 8 additions and 4 multiplications (for M = 4, not required for M = 2)
indices ← topk(outs/rdn, K = M)  ▷ 5 comparisons (for M = 4, not required for M = 2)
scale ← 1/p                                             ▷ 2 or 1 divisions (for M = 4 or 2)
return indices, scale

```

B.2 N:M STRUCTURED PRUNING ON THE ACTIVATIONS

In Appendix A.5 we showed the effect of applying greedy N:M fine-grained structured sparsity on the activations and showed we are able to preserve the accuracy and potentially accelerate by x2 the multiplication with the activations in the forward phase. In Table 10 we show the overhead of 2:4 and 4:8 structured pruning of the activations, in comparison to standard ReLU activations over regular training of ResNet18. The measurements were done on FP32 over a Titan1080 GPU using current unoptimized code, we believe there is potentially place for improvement. As can be seen, even with the unoptimized implementation, the overhead is negligible. Similar overhead can be found in Weixiang et al. (2022) for weight pruning.

Quantitative estimation of the activations pruning overhead Note that for the activations we do not use the approx-MVUE, since a simple magnitude-based approach works well. Thus a similar derivation to Appendix B.1 results in 5 comparisons for $M = 4$ and for $M = 8$. For a Linear Layer with activations $h \in R^{B \times C_I}$ and weights $W \in R^{C_I \times C_O}$, the operations reduction would be $B C_I C_O / 2$ while pruning the activations requires $1.25 B C_I / 4$ (for $M = 4$). Therefore for $M = 4$ pruning is always efficient ($C_O > 1$) and for $M = 8$ it is efficient if $C_O > 3$. Similarly, for a

Convolution layer, with activation $h \in R^{B \times C_I \times H \times W}$ and weights $C_I \times C_O \times k \times k$ we get that the operations reduction would be $BC_I C_O HW k^2 / 2$ while the pruning requires $1.25 BC_I HW / 4$ (for $M = 4$). Therefore here as well for $M = 4$ pruning is always efficient ($C_O > 1$) and for $M = 8$ it is efficient if $C_O > 3/k^2$.

Table 10: Overhead of 2:4 and 4:8 activation pruning in comparison to standard ReLU activation in ResNet18 ImageNet dataset.

Method	Overhead (%)
2:4	0.1 %
4:8	0.17 %

B.3 COMPARISON WITH SDGP (MCDANEL ET AL., 2022)

In parallel to our work, McDanel et al. (2022), proposed to prune the neural gradients to accelerate only the backward phase, while the update and forward phase are not pruned. Their method is based on the traditional greedy method, followed by a rescaling of the remaining elements to keep the l_1 norm. In order to check the effect of their method we show in Table 11 the results of applying SDGP also in the update phase. As can be seen, while their method works well in the backward phase, their biased method creates a high degradation in the update phase. Notice that in all their experiments they use FFCV (Leclerc et al., 2022) training regime, which achieves a higher baseline.

Table 11: ResNet18 top-1 validation accuracy in ImageNet dataset while applying SDGP (McDanel et al., 2022) in the backward and update phases.

Method	Accuracy (%)
Baseline	71.4 %
SDGP backward	71.2 %
SDGP update	64.2 %

B.4 ADDITIONAL SPARSITY RESULTS

In Table 12 we extend Table 4 to additional N:M sparsity setups. "Approx-4:8" is similar to "Approx-2:4" but we extend it to additional elements. "MVUE 1:4" is similar to "MVUE 1:2", but for a bigger block. Notice that, as remarked in Section 6 applying higher sparsity ratios requires additional hardware support since we increase the required bandwidth as for every single block of neural gradient we bring four activations to the engine.

Table 12: Extension of Table 4 to additional sparsity setups for ResNet18 and ResNet50 in ImageNet dataset.

Model	FP32	Greedy	MVUE 1:2	Approx-2:4	Approx-4:8	MVUE 1:4
ResNet18	70.6 %	48.2 %	70.58 %	70.6 %	70.6 %	69.93 %
ResNet50	77.2 %	59.3 %	76.4 %	77.12 %	77.15 %	75.8 %