Low-memory stochastic backpropagation with multi-channel randomized trace estimation

Mathias Louboutin[1], Ali Siahkoohi[1], Rongrong Wang[2], Felix J. Herrmann[1]
[1] School of Computational Science and Engineering, Georgia Institute of Technology
[2] Department of Mathematics, Michigan State University

Abstract

Thanks to the combination of state-of-the-art accelerators and highly optimized open software frameworks, there has been tremendous progress in the performance of deep neural networks. While these developments have been responsible for many breakthroughs, progress towards solving large-scale problems, such as video encoding and semantic segmentation in 3D, is hampered because access to on-premise memory is often limited. Instead of relying on (optimal) checkpointing or invertibility of the network layers—to recover the activations during backpropagation—we propose to approximate the gradient of convolutional layers in neural networks with a multi-channel randomized trace estimation technique. Compared to other methods, this approach is simple, amenable to analyses, and leads to a greatly reduced memory footprint. Even though the randomized trace estimation introduces stochasticity during training, we argue that this is of little consequence as long as the induced errors are of the same order as errors in the gradient due to the use of stochastic gradient descent. We discuss the performance of networks trained with stochastic backpropagation and how the error can be controlled while maximizing memory usage and minimizing computational overhead.

1 Introduction

Convolutional layers continue to form a key component of current neural network designs. Even though the computational demands during the forward evaluation are relatively modest, significant computational resources are needed during training, which typically requires storage of the state variables (activations) and dense operations between the input and output. By using accelerators (e.g. GPUs, TPUs, Inferentia), the arithmetical component of training is met as long as memory usage is controlled.

Unfortunately, restricting memory usage without introducing significant computational overhead remains a challenge and can lead to difficult to manage additional complexity. Examples include (optimal) checkpointing [1, 2], where the state is periodically stored and recomputed during the backward pass, invertible networks [3–5], where the state can be derived from the output, and certain approximation methods where computations are made with limited precision arithmetic [6] or where unbiased estimates are made of the gradient using certain approximations [7, 8], e.g., via randomized automatic differentiation (RAD, [9]) or via direct feedback alignment (DFA, [10–12]).

Our work is based on the premise that exact computations are often not needed, which is an approach advocated in the field of randomized linear algebra [13, 14], and more recently in the field parametric machine learning [9]. There the argument has been made that it is unnecessary to spend computational resources on exact gradients when stochastic optimization is used. A similar argument was used earlier in the context of parameter estimation with partial-differential equation constraints [15–17]. However, contrary to intervening into computational graphs as in RAD, our approach exploits the underlying linear algebra structure exhibited by the gradient of convolutional layers.

By means of relatively straightforward algebraic manipulations, we write the gradient with respect to a convolution weight in terms of the matrix trace of the outer product between the convolutional layer input, the backpropagated residual, and a shift. Next, we approximate this trace with an unbiased randomized trace estimation technique [14, 18–21] for which we prove convergence and derive theoretical error bounds by extending recent theoretical results [22]. To meet the challenges of training the most popular convolutional neural networks (CNN), we present a randomized probing technique capable of handling multiple input/output channels. We validate our approach on the MNIST and CIFAR10 datasets for which we achieve overall (savings of individual convolutional layers is much larger) network memory savings of at least a factors of $2.5\times$. Our results are reproducible at: Anonymous.

2 Theory

To arrive at our low-memory footprint convolutional layer, we start by casting the action of these layers into a framework that exposes the underlying linear algebra. By doing this, gradients with respect to the convolution weights can be identified as traces of a matrix. By virtue of this identification, these traces can be approximated by randomized trace estimation [18], which greatly reduces the memory footprint at negligible or even negative (speedup) computational overhead. We start by deriving expressions for the single channel case, followed by a demonstration that randomized trace estimation leads to unbiased estimates for the gradient with respect to the weights. Next, we justify the use of randomized trace estimation by proving that its validity can be extended to arbitrary matrices. Aside from proving convergence as the number of probing vectors increases, we also provide error bounds before extending the proposed technique to the multi-channel case. The latter calls for a new type of probing to minimize cross-talk between the channels via orthogonalization. We derive bounds for the accuracy for this case as well.

Single channel case Let us start by writing the action of a single channel convolutional layer as follows

$$\mathbf{Y} = \mathbf{W}\mathbf{X} \in \mathbb{R}^{N \times B}, \quad \text{where} \quad \mathbf{W} = \sum_{i=1}^{n_w} w_i \mathbf{T}_{k(i)}, \tag{1}$$

and N, B, n_w are the number of pixels, batchsize, and number of convolution weights (K^2 for a K by K kernel), respectively. For the ith weight w_i, the convolutions themselves correspond to applying a circular shift with offset $k(i)$, denoted by $\mathbf{T}_{k(i)}$, followed by multiplication with the weight. Given this expression for the action of a single-channel convolutional layer, expressions for the gradient with respect to weights can easily be derived by using the chain rule and standard linear algebra manipulations [23]—i.e, we have

$$\begin{aligned}
\frac{\partial}{\partial w_i} f(\mathbf{W}\mathbf{X}) &= \text{tr}\left(\left(\frac{\partial f(\mathbf{W}\mathbf{X})}{\partial \mathbf{W}} \right)^{\top} \frac{\partial \mathbf{W}}{\partial w_i} \right) \\
&= \text{tr}\left(\left(\delta\mathbf{Y}\mathbf{X}^{\top} \right)^{\top} \mathbf{T}_{k(i)}^{\top} \right) = \text{tr}\left(\mathbf{X}\delta\mathbf{Y}^{\top}\mathbf{T}_{-k(i)} \right), \quad i = 1, \ldots, n_w.
\end{aligned} \tag{2}$$

This expression for the gradient with respect to the convolution weights corresponds to computing the trace—i.e., the sum along the diagonal elements denoted by $\text{tr}(\mathbf{A}) = \sum_i A_{ii}$, of the outer product between the residual collected in $\delta\mathbf{Y}$ and the layer's input $\mathbf{X}$, after applying the shift. The latter corresponds to a right circular shift along the columns.

Computing estimates for the trace through the action of matrices—i.e., without access to entries of the diagonal, is common practice in the emerging field of randomized linear algebra [13, 14]. Going back to the seminal work by Hutchinson [19, 20], unbiased matrix-free estimates for the trace of a matrix exist involving probing with random vectors $\mathbf{z}_j$, $j = 1, \ldots, r$, with r the number of probing vectors and $\mathbb{E}(\mathbf{z}_j \mathbf{z}_j^{\top}) = \mathbf{I}$ with $\mathbf{I}$ the identity matrix. Under this assumption, unbiased randomized trace estimates can be derived from

$$\text{tr}(\mathbf{A}) = \text{tr}\left(\mathbf{A}\mathbb{E}\left[\mathbf{z}\mathbf{z}^{\top} \right] \right) = \mathbb{E}\left[\mathbf{z}^{\top}\mathbf{A}\mathbf{z} \right] \approx \frac{1}{r} \sum_{j=1}^{r} \left[\mathbf{z}_j^{\top}\mathbf{A}\mathbf{z}_j \right]. \tag{3}$$

By combining (2) with the above unbiased estimator for the trace, we arrive at the following approximation for gradient with respect to the convolution weights:

$$\delta w_i \approx \frac{1}{r} \sum_{j=1}^{r} \left(\mathbf{z}_j^\top \mathbf{X} \right) \left(\delta \mathbf{Y}^\top \mathbf{T}_{-k(i)} \mathbf{z}_j \right), \quad i = 1, \ldots, n_w. \tag{4}$$

From this expression the memory savings during the forward pass are obvious since $\overline{\mathbf{X}} = \sum_{j=1}^{r} \left(\mathbf{z}_j^\top \mathbf{X} \right) = \mathbf{Z}^\top \mathbf{X}$, where $\overline{\mathbf{X}} \in \mathbb{R}^{r \times B}$ with $r \ll N$. However, convergence rate guarantees were only established under the additional assumption that $\mathbf{A}$ is positive semi-definite (PSD, [24]). While the outer product $\mathbf{X} \delta \mathbf{Y}^\top \mathbf{T}_{-k(i)}$ we aim to probe here is not necessarily PSD, improving upon recent results by [22], we show that the condition of PSD can be relaxed to asymmetric matrices by a symmetrization procedure that does not change the trace. More precisely, we show in the following proposition that the gradient estimator in (4) is unbiased and converges to the true gradient as $r \to \infty$ with a rate of about $r^{-1/2}$ (for details of the proof, we refer to the appendix A.2).

Proposition 1. *Let* $\mathbf{A} \in \mathbb{R}^{N \times N}$ *be a square matrix and the probing vectors are i.i.d. Gaussian with* 0 *mean and unit variance. Then for any small number* $\delta > 0$, *with probability* $1 - \delta$, *we have*

$$\left| \frac{1}{r} \sum_{i=1}^{r} \left[\mathbf{z}_i^\top \mathbf{A} \mathbf{z}_i \right] - \mathrm{tr}\left(\mathbf{A} \right) \right| \leq \frac{4 \|\mathbf{A}\|_2}{r} \log \frac{2}{\delta} + \frac{2 \|\mathbf{A}\|_F}{\sqrt{r}} \log^{1/2} \frac{2}{\delta}.$$

Imposing a small probability of failure δ means the $\log \frac{2}{\delta}$ term in the upper bound is large, which implies that neither term in the upper bound is dominating for all the r values. Depending on which term is dominant, the range of r can be divided into two regimes, the small r regime and the large r regime. In the small r regime, the first term dominates, and the error decays linearly in r. In the large r regime, the second term dominates and the error decays as the $\sqrt{r}$. The phase transition happens when r is about $4\|\mathbf{A}\|_2^2/\|\mathbf{A}\|_F^2 \log 2/\delta \equiv \frac{4}{\rho} \log 2/\delta$, where $\rho \equiv \|\mathbf{A}\|_F^2/\|\mathbf{A}\|_2^2$ is known as the effective rank, which reflects the rate of decay of the singular values of $\mathbf{A}$. We see that as r increases, the larger the effective rank is, the earlier the phase transition occurs, after which the decay rate of the error will slow down. Before discussing details of the proposed algorithm, let us first extend the above randomized trace estimator to multi-channel convolutions.

Multi-channel case In general, convolutional layers involve several input and output channels. In that case, the output of the mth channel can be written as

$$\mathbf{Y}^m = \sum_{n=1}^{C_{\text{in}}} \mathbf{W}^{n,m} \mathbf{X}^n, \quad \text{where} \quad \mathbf{W}^{n,m} = \sum_{i=1}^{n_w} w_i^{n,m} \mathbf{T}_{k(i)} \tag{5}$$

for $n = 1, \ldots, C_{\text{in}}$, $m = 1, \ldots, C_{\text{out}}$ with $C_{\text{in}}, C_{\text{out}}$ the number of input and output channels and $w_i^{n,m}$ the ith weight between the nth input at m^{th} output channel. In this multi-channel case, the gradients consist of the single channel gradient for each input/output channel pair, i.e., $\delta w_i^{n,m} = \mathrm{tr}(\mathbf{X}^n (\delta \mathbf{Y}^m)^\top \mathbf{T}_{-k(i)})$.

While randomized trace estimation can in principle be applied to each input/output channel pair independently, we propose to treat all channels simultaneously to further improve computational performance and memory use. Let the outer product of the $(n, m)^{\text{th}}$ input/output channel be $\mathbf{A}^{n,m}$, i.e, $\mathbf{A}^{n,m} = (\mathbf{X}^n (\delta \mathbf{Y}^m)^\top \mathbf{T}_{-k(i)})^\top$, computing $\delta w_i^{n,m}$ means estimating $\mathrm{tr}(\mathbf{A}^{m,n})$. To save memory, instead of probing each $\mathbf{A}^{n,m}$, we probe the stacked matrix

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}^{1,1} & \cdots & \mathbf{A}^{1,C_{\text{in}}} \\ \vdots & & \vdots \\ \mathbf{A}^{C_{\text{out}},1} & \cdots & \mathbf{A}^{C_{\text{out}},C_{\text{in}}} \end{pmatrix}$$

by r length $C_{in} \times N$ probing vectors stored in $\mathbf{Z} \in \mathbb{R}^{NC_{in}, r}$, and estimate each $\mathrm{tr}(\mathbf{A}^{m,n})$ via the following estimators

$$G^{m,n}(\mathbf{A}) := \frac{1}{r} \sum_{j=1}^{r} \mathbf{z}_{n,j}^\top \left(\mathbf{A} \mathbf{z}_j \right)_m, \tag{6}$$

3

where $(\cdot)_m$ extracts the mth block from the input vector. That is to say, we simply stack the input and residual, yielding matrices of size $(N \times C_{\text{in}}) \times B$ and $(N \times C_{\text{in}}) \times B$ whose outer product $\mathbf{X}\delta\mathbf{Y}^\top$ (i.e. $\mathbf{A}^\top$ of the $\mathbf{A}$ in (6)) is no longer necessarily square. To estimate the trace of each $N \times N$ sub-block, in (6), we (*i*) probe the full outer product from the right with r probing vectors $\mathbf{z}_j$ of length $(N \times C_{\text{in}})$; (*ii*) reshape the resulting matrix into a tensor of size (N, C_{out}, B) while the probing matrix is shaped into a tensor of size (N, C_{in}, B) (i.e.,separate each block of $\mathbf{A}z_j$), and (*iii*) probe each individual block again from the left. This leads to the desired gradient collected in a $C_{\text{in}} \times C_{\text{out}}$ matrix. We refer to Figure 1, which illustrates this multi-channel randomized trace estimation. After (*i*), we only need to save $\overline{\mathbf{X}} = \sum_{j=1}^{r} \left(\mathbf{z}_j^\top \mathbf{X}\right) = \mathbf{Z}^\top \mathbf{X}$ in memory rather than $\mathbf{X}$ that leads to a memory reduction by a factor of $\frac{NC_{\text{in}}}{r}$.

Unfortunately, the improved memory use and computational performance boost of the above multi-channel probing reduces the accuracy of the randomized trace estimation because of crosstalk amongst the channels. Since this cross-talk is random, the induced error can be reduced by increasing the number of probing vectors r, but this will go at the expense of more memory use and increased computation. To avoid this unwanted overhead, we introduce a new type of random probing vectors that minimizes the crosstalk by again imposing $\mathbb{E}(\mathbf{z}\mathbf{z}^\top) = \mathbf{I}$ but now on the multi-channel probing vectors that consist of multiple blocks corresponding to the number of input/output channels.

Explicitly, we draw each $\mathbf{z}_{n,j}$, the nth block of the j probing vector, according

$$\mathbf{z}_{n,j} \sim \begin{cases} \mathcal{N}(\mathbf{0}, \mathbf{I}_N) & \text{with probability } p_n \\ 0 & \text{with probability } 1 - p_n \end{cases}. \tag{7}$$

For different values of (n, j), the $\mathbf{z}_{n,j}$'s are drawn independently with a predefined probability p_n of generating a nonzero block. Compared to conventional (Gaussian) probing vectors (see Figure 2 top left), these multi-channel probing vectors contain sparse non-zero blocks (see Figure 2 top right), which reduces the crosstalk (juxtapose with second row of Figure 2). It can be shown that crosstalk becomes less when $p_n \downarrow 0$ and $r \to \infty$.

Given probing vectors drawn from (7), we have to modify the scaling factor of the multi-channel randomized trace estimator (6) to ensure it is unbiased,

$$\widetilde{G}^{m,n}(\mathbf{A}) := \frac{1}{\text{nnz}(z_m)} \sum_{j=1}^{r} \mathbf{z}_{n,j}^\top \left(\mathbf{A}\mathbf{z}_j\right)_m, \tag{8}$$

where $\text{nnz}(z_m)$ is the number of non-zero columns in block m. We proof the following convergence result for this estimator (the proof can be found in appendix A.2).

Theorem 1 (Succinct version). *Let $p = \min\limits_{n} p_n$, r be the number of probing vectors. For any small number $\delta > 0$, with probability over $1 - \delta - 2C_{in}e^{-rp^2/2}$, we have for any $n = 1, \ldots, C_{in}$ and $m = 1, \ldots, C_{out}$,*

$$\left|\widetilde{G}^{n,m}(\mathbf{A}) - \text{tr}(\mathbf{A}^{n,m})\right| \leq c \cdot \frac{\frac{1}{\sqrt{p_m}}\|\mathbf{A}^{n,m}\|_F + \sum\limits_{j=1, j\neq m}^{C_{in}} \sqrt{\frac{p_k}{p_m}}\|\mathbf{A}^{n,j}\|_F}{\sqrt{r}} \log\frac{C_{out}C_{in}}{\delta},$$

where c is an absolute constant and C_{in} and C_{out} are the numbers of input and output channels.

Theorem 1 provides convergence guarantee for our special multi-channel simultaneous probing procedure. Similar to Proposition 1, Theorem 1 in its original form (supplementary material) also has a two-phase behaviour. So the discussion under Proposition 1 applies here. For simplification of presentation, we only presented the bound for the large r regime in this succinct version. Still, we can see that the error bound for estimating $\text{tr}(\mathbf{A}^{n,m})$ not only depends on the norm of the current block $\mathbf{A}^{n,m}$, but also other blocks in that row, which is expected since we simultaneous probe the entire row instead of each block individually for memory efficiency. Admittedly, due to technical difficulties, we can not theoretically show that decreasing the sampling probability p decreases the error. Nevertheless, we observe better performance in the numerical experiments.

3 Stochastic optimization with multi-channel randomized trace estimation

Given the expressions for the approximate gradient calculations of convolutional layers and bounds on their error, we are now in a position to introduce our algorithm and analyze its performance
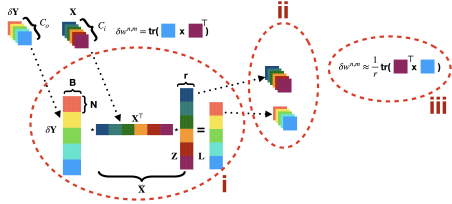
4

Figure 1: Multi-channel randomized trace estimation. This figure shows the three steps of the algorithm to estimate the trace of a sub block of the outer product.
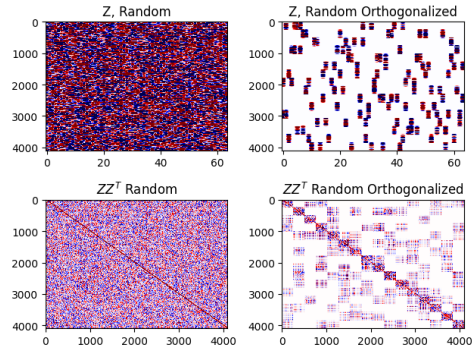

Figure 2: Probing matrices and corresponding approximation of the identity for $C_{\text{in}} = 16$, $C_{\text{out}} = 16$, $r = 64$, and $N = 256$. We can see that the orthogonalization step by zeroing out blocks (compare plots in top row) leads to a near block diagonal approximation of the identity with much less cross-talk between the different channels (compare plots in second row).

on stylized examples and the MNIST and CIFAR10 datasets in the Experiment section 4. We will demonstrate that for fixed memory usage the errors in the gradient are of the same order as errors induced by selecting different min-batches. This confirms similar observations made by [9]. We conclude this section by comparing memory usage and speed of an actual neural network.

Low-memory stochastic backpropagation The key point of the randomized trace estimator in Equation (8) is that it allows for on-the-fly compression of the state variables during the forward pass. For a single convolutional layer $\mathbf{Y} = \mathbf{conv}(\mathbf{X}, \mathbf{w})$ with input $\mathbf{X}$ and convolution weights $\mathbf{w}$, our approximation involves three simple steps, namely **(1)** probing of the state variable $\overline{\mathbf{X}} = \mathbf{Z}^\top \mathbf{X}$, **(2)** matrix-free formation of the outer product $\mathbf{L} = \overline{\mathbf{X}}\mathbf{Y}^\top$, and **(3)** approximation of the gradient via $\delta w_i = \frac{1}{r} \text{tr}(\mathbf{L}\mathbf{T}_{-k(i)}\mathbf{Z})$, $i = 1, \ldots, r$. These three steps lead to major memory reductions even for a relatively small image size of $N = 32 \times 32$ and $C_{\text{in}} = 16$. In that case, our approach leads to a memory reduction by a factor of $\frac{NC_{\text{in}}}{r} = 2^{14-\gamma}$ for $r = 2^\gamma$. For $\gamma = 7$ this leads to $100\times$ memory saving. Because the probing vectors are generated on the fly, we only need to allocate memory for $\overline{\mathbf{X}}$ during the forward pass as long as we also store the state s of the random generator. During backpropagation, we initialize the state, generate the probing vectors, followed by applying a shift and product by $\mathbf{L}$. These steps are summarized in Algorithm 1. This simple yet powerful algorithm provides a virtually memory free estimate of the true gradient with respect to its weights.

Algorithm 1 Low-memory approximate gradient convolutional layer. The random seed s and random probing matrix $\mathbf{Z}$ are independently redrawn for each layer and training iteration.

Forward pass:
1. Forward convolution $\mathbf{Y} = \mathbf{conv}(\mathbf{X}, \mathbf{w})$
2. Draw a new random seed s and probing matrix $\mathbf{Z}[s]$
3. Compute and save $\overline{\mathbf{X}} = \mathbf{Z}^\top[s]\mathbf{X} \in \mathbb{R}^{r \times B}$
4. Store $\overline{\mathbf{X}}, s$
Backward pass:
1. Load random seed s and probed forward $\overline{\mathbf{X}}$
2. Redraw probing matrix $\mathbf{Z}[s]$ from s
3. Compute backward probe $\mathbf{L} = \bar{\mathbf{X}}\mathbf{Y}^\top$
4. Compute gradient $\delta w_i = \frac{1}{r} \text{tr}(\mathbf{L}\mathbf{T}_{-k(i)}\mathbf{Z}[s])$

Minibatch versus randomized trace estimation errors   Simply stated, stochastic optimization involves gradients that contain random errors known as gradient noise. As long as this noise is not too large and independent for different gradient calculations, algorithms such as stochastic gradient descent where gradients are computed for randomly drawn minibatches, converge under certain conditions. In addition, the presence of gradient noise helps the algorithm to avoid bad local minima, which arguably leads to better generalization of the trained network [25, 26]. Therefore, as long as the batchsize is not too large, one can expect the trained network to perform well.

We argue that the same applies to stochastic optimization with gradients approximated by (multichannel) randomized trace estimation as long the errors behave similarly. In a setting where memory comes at a premium this means that we can expect training to be successful for gradient noise with similar variability. To this end, we conduct an experiment where the variability of $5 \times 5 \times C_{in} \times C_{out}$ ($C_{in} = C_{out} = 16$) convolution weights are calculated for the true gradient for different randomly drawn minibatches of size $B = 128$. We do this for a randomly initialized image classification network designed for the CIFAR10 dataset (for network details, see Table 4 in appendix A.3).

For comparison, approximate gradients are also calculated for randomized trace estimates obtained by probing independently ("Indep." in blue), multi-channel ("Multi" in orange), and multichannel with orthogonalization ("Multi-Ortho" in green). The batchsizes are for a fixed probing size of $r = 2048$ selected such that the total memory use is the same as for the true gradient calculations. From the plots in Figure 3, we observe that as expected the independent probing is close to the true gradient followed by the more memory efficient multi-channel probing with and without orhogonalization. While all approximate gradient are within the 99% confidence interval, the orhogonalization has a big effect when the gradients are small (see conv3).
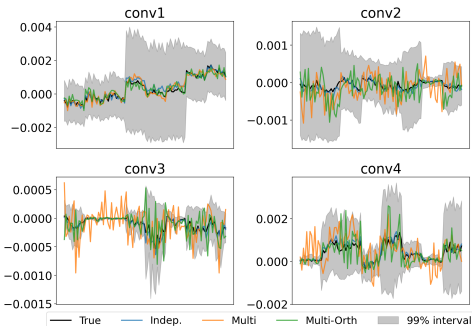

Figure 3: Randomized trace estimation of the gradient of our randomly initialized CNN for the CIFAR10 dataset. While gradient noise is present, its magnitude is reduced by the orthogonalization when weights are small.
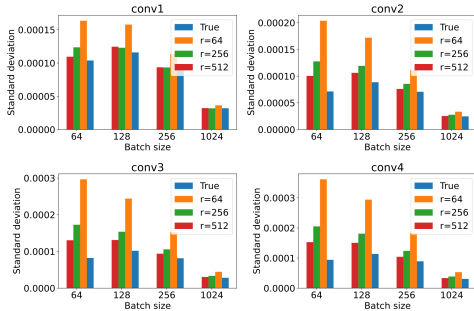

Figure 4: Standard deviation of the gradients w.r.t the weights for each of the four convolutional layers in the neural network. The standard deviation is computed over 40 minibatches randomly drawn from the CIFAR10 dataset.

To better understand, the interplay between different batchsizes $B = \{64, 128, 256, 1024\}$ and numbers of probing vectors $r = \{64, 256, 512\}$, we also computed estimates for the standard deviation from 40 randomly drawn minibatches. As expected, the standard deviations of the network weights gradients increase for smaller batchsize and number of probing vectors. Moreover, the variability of the approximates obtained with randomized trace estimation are for the deeper convolutional layers larger for $B = 64$. However, since we can afford larger batchsizes for similar memory usage, we can control the variability for a given memory budget by using a larger batch size.

Overall effective memory savings   Approximate gradient calculations with multi-channel randomized trace estimation can lead to significant memory savings within convolutional layers. Because these layers operate in conjunction with other network layers such as `ReLU` and batchnorms, the overall effective memory savings depend on the ratio of pure convolutional and other layers and on the interaction between them. This is especially important for layers such as `ReLU`, which rely on the next layer to store the state variable during backpropagation. Unfortunately, that approach no longer works because our low-memory convolutional layer does not store the state variable. However, this situation can be remedied easily by only keeping track of the signs [9].
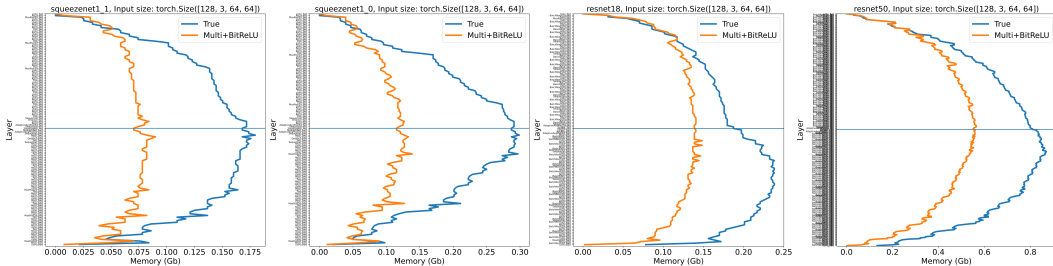
6

Figure 5: Network memory usage for a single gradient. We show the memory usage for known networks for low and high probing sizes for a fixed input size.

To assess what the effective memory savings are of the multi-channel trace estimation, we include in Figure 5 layer-by-layer comparisons of memory usage for different versions of the popular SqueezeNet [27] and ResNet [28]. The memory use for the conventional implementation is plotted in blue and our implementation in orange. The results indicate that memory savings by a factor of two or more are certainly achievable, which allows for a doubling of the batchsize or increases in the width/depth of the network. As expected, the savings depend on the ratio of CNN versus other layers.

Wall-clock benchmarks   Ideally, reducing the memory footprint during training should not go at the expense of computational overhead that slows things down. To ensure this is indeed the case, we implemented the multi-channel randomized trace estimation optimized for CPUs in Julia [29] and for GPUs in PyTorch [30]. Implementation details and benchmarks are included in appendix A.5.

Our extensive benchmarking experiments demonstrate highly competitive performance for both CPUs, against the state-of-the-art NNLib [31, 32], and GPUs, against the highly optimized implementation of convolutional layers in CUDA. On CPUs, we even outperform for large images and large batchsizes the standard *im2col* [33] implementation by up to $10\times$ as long as the number of probing vectors remains relatively small. We observe similar behavior for GPUs, where we remain competitive and even at times outperform highly optimized CuDNN kernels [34] with room for further improvement. In all cases, there is a slight decrease in performance when the number of channels increases. Overall, approximate gradient calculations with multi-channel randomized trace estimation substitute expensive convolutions between the input and output channels by a relatively simple combination of matrix-free actions of the outer product on random probing vectors on the right and dense linear matrix operations on the left (cf.(8) and Algorithm 1).

4   Experiments

Table 1: Training accuracy for varying batchsizes B and number of probing vectors r on the MNIST dataset.

	$B = 64$	$B = 128$	$B = 256$
True	0.9905	0.9898	**0.9901**
$r = 2$	0.9625	0.9692	0.9745
$r = 16$	0.9753	0.9803	**0.9823**
$r = 64$	0.9777	0.9723	0.9782
$r = 256$	0.9718	0.9706	0.9791

Even though memory and computational gains of our proposed method can be significant during backpropagation, accuracy of trained networks needs to be verified. To this end, we conduct a number of experiments on the MNIST and CIFAR10 datasets. In these experiments, we vary the batchsize B and the number of probing vectors r. Implementations both in Julia and Python are evaluated.

MNIST dataset   We start by training two "MNIST networks" (detailed in Table 2 and 3 of appendix A.3 for Julia and PyTorch with training parameters listed in appendix A.4) for varying batchsizes $B = \{64, 128, 256\}$ and number of probing vectors $r = \{2, 16, 64, 256\}$. The network test accuracies for the the Julia implementation, where the default convolutional layer implementation is replaced by **XConv.jl**, are listed in Table 1 for the default implementation and for our implemen-

7

tation where gradients of the convolutional layers are replaced by our approximations. The results show that our low-memory implementation remains competitive (compare numbers in bold) even for a small number of probing vectors, yielding a memory saving of about $2.5\times$.

We obtained the results listed in Table 1 with the ADAM [35] optimization algorithm. In an effort to add robustness when training overparameterized deep neural networks, we switch in the next example to stochastic line searches (SLS, [36]) that remove the need to set hyperparameters manually. With this algorithm, the line search parameters are set automatically at the cost of an extra gradient calculation. Figure 6 shows the test accuracies as a function of the number of epochs, batchsize $B = 2^\gamma, \gamma = 6, \ldots, 11$ and number of probing vectors $r = 2^\gamma, \gamma = 4, \ldots, 8$. Because the randomized trace estimation is unbiased, we observe convergence as r increases. Despite relatively large approximation errors for small r, we also notice that the induced randomness by our approximate gradient calculations does not adversely affect the line searches. As in the previous example, we achieve competitive results with slight random fluctuations for $r = 16$ for all batchsizes, resulting in a reduction in memory use by a factor of about $2.5\times$.

CIFAR10 dataset To conclude our empirical validation of approximate gradient calculations with multi-channel randomized trace estimation, we train a network on the CIFAR10 dataset. Compared to the previous examples, this is a more challenging larger realistic training problem. To mimic an actual training scenario, memory usage is fixed between the regular gradient, and the approximate gradients obtained by probing independently ("Indep." in blue with $r = 32$), multi-channel ("Multi." in green with $r = 256$), and multi-channel with orthogonalization ("Multi-Ortho" in red with $r = 256$). The batchsize for the approximate gradient examples is increased from $B = 128$ to $B = 256$ to reflect the smaller memory footprint. Results for the training/testing loss and accuracy are included in Figure 7. The following observations can be made from these plots. First, there is a clear gap between the training/testing loss for the true and approximate gradients. This gap is also present in the training/testing albeit it is relatively small. However, because of doubling the batchsize the runtime for the training is effectively halved.

5 Related work

The continued demand to train larger and larger networks, for tasks such as video compression and classification in 3D, puts pressure on the memory of accelerators (GPUs, etc.), which is in short supply. This memory pressure is exacerbated when training relies on backpropagation that in its mundane form calls for storage of the state variables during the forward pass. To relieve this memory pressure several attempts have been made, ranging from the use of optimal checkpointing [1, 2] to
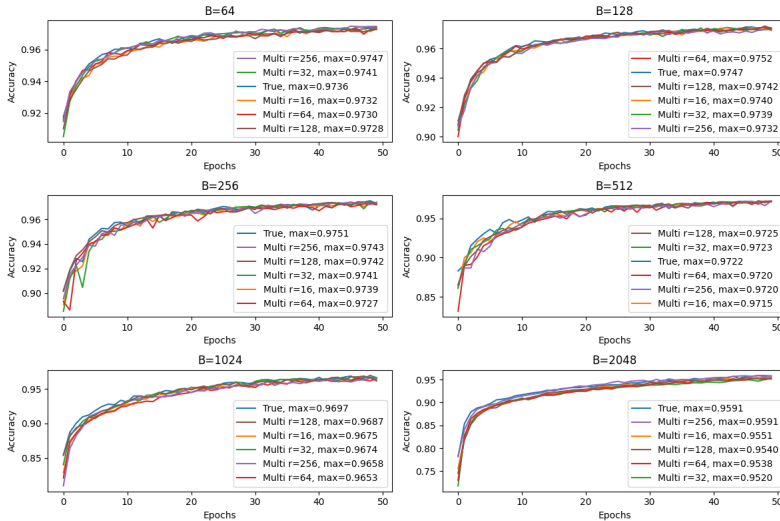

Figure 6: Training for varying batchsizes and probing sizes. This experiment ran with the Stochastic Line Search algorithm (SLS, [36])
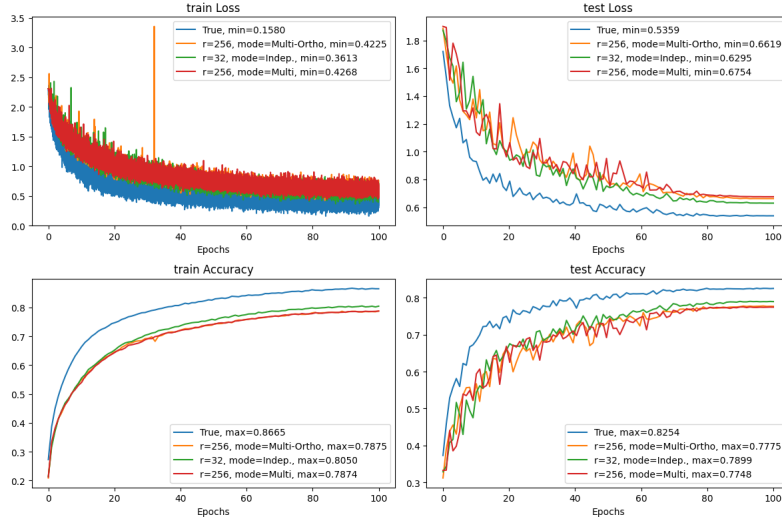
8

Figure 7: CIFAR10 training with equivalent memory comparing our approximate method to standard training. The four panels show the training and test loss (top row) and training and test accuracy (bottom row) after 100 epochs.

the use of invertible neural networks [3–5]. While these approaches can reduce the memory footprint during training, they introduce significant computational overhead, algorithmic complexity, and invertible neural network implementations that may lack in expressibility.

Alternatively, people have relied on approximate arithmetic [6], on replacing symmetric backpropagation by direct feedback alignment [10–12], where random projections on the residual are used, or on approximations of gradients and Jacobians with techniques borrowed from randomized linear algebra [14, 18, 20]. Compared to these other approaches, the latter is capable of producing approximations that are unbiased, a highly desirable feature when training neural networks with stochastic optimization [25]. Instead of randomizing the forward pass in stochastic computational graphs, we propose to approximate gradients by exploiting the special structure of gradients of convolutional layers. This structure allows us to use the relatively simple method of randomized trace estimation to approximate the gradient while reducing the memory footprint significantly. While perhaps less versatile than the recently proposed method of randomized automatic differentiation [9], our approach does not need intervention in the computational graph and acts as a drop-in replacement for the 2D and 3D convolutional layers in existing machine learning frameworks.

6 Conclusion and Future work

We introduced a novel take on convolutional layers grounded on recent work in randomized linear algebra that allows for unbiased estimation of the trace via randomized probing. Aside from being memory efficient—i.e., the state variable only needs to be stored in a majorly compressed form, the proposed approach, where gradients with respect to convolution weights are approximated by traces, also has computational advantages outperforming state-of-the-art neural network implementations. In addition, randomized trace estimation comes with convergence guarantees and error estimates, which have the potential to inform the proposed algorithm. While there is still room for improvements, networks trained with approximate gradients calculated with randomized probing have a performance that is very close to that of the most advanced training methods with an error that decreases with the number of randomized probing vectors. The latter opens enticing perspectives given recent developments in specialized photonic hardware where the speed of randomized probing is drastically increased [37]. This will allow future implementations of our approach to scale to large problems in video representation learning and other 3D applications.

References

[1] Andreas Griewank and Andrea Walther. Algorithm 799: Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation. *ACM Trans. Math. Softw.*, 26(1):19–45, March 2000. ISSN 0098-3500. doi: 10.1145/347837.347846. URL https://doi.org/10.1145/347837.347846.

[2] Olivier Beaumont, Lionel Eyraud-Dubois, Julien Herrmann, Alexis Joly, and Alena Shilova. Optimal checkpointing for heterogeneous chains: how to train deep neural networks with limited memory. *CoRR*, abs/1911.13214, 2019. URL http://arxiv.org/abs/1911.13214.

[3] Eldad Haber and Lars Ruthotto. Stable Architectures for Deep Neural Networks. *Inverse Problems*, 34(1):014004, dec 2017. doi: 10.1088/1361-6420/aa9a90. URL https://doi.org/10.1088/1361-6420/aa9a90.

[4] Jörn-Henrik Jacobsen, Arnold W.M. Smeulders, and Edouard Oyallon. i-RevNet: Deep Invertible Networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJsjkMb0Z.

[5] Tristan Hascoet, Quentin Febvre, Yasuo Ariki, and Tetsuya Takiguchi. Reversible designs for extreme memory cost reduction of cnn training. *arXiv preprint arXiv:1910.11127*, 2019. URL https://arxiv.org/abs/1910.11127.

[6] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. *CoRR*, abs/1502.02551, 2015. URL http://arxiv.org/abs/1502.02551.

[7] Ziheng Wang, Sree Harsha Nelaturu, and Saman Amarasinghe. Accelerated CNN Training through Gradient Approximation. In *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pages 31–35. IEEE, 17 February 2019. doi: 10.1109/EMC249363.2019.00014. URL https://ieeexplore.ieee.org/document/9027239.

[8] Arild Nøkland and Lars Hiller Eidnes. Training Neural Networks with Local Error Signals. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4839–4850. PMLR, 09–15 June 2019. URL http://proceedings.mlr.press/v97/nokland19a.html.

[9] Deniz Oktay, Nick McGreivy, Joshua Aduol, Alex Beatson, and Ryan P Adams. Randomized Automatic Differentiation. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=xpx9zj7CUlY.

[10] Arild Nø kland. Direct Feedback Alignment Provides Learning in Deep Neural Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/d490d7b4576290fa60eb31b5fc917ad1-Paper.pdf.

[11] Donghyeon Han and Hoi jun Yoo. Efficient Convolutional Neural Network Training with Direct Feedback Alignment. *arXiv preprint arXiv:1901.0198*, 2019. URL https://arxiv.org/abs/1901.01986.

[12] Charlotte Frenkel, Martin Lefebvre, and David Bol. Learning Without Feedback: Fixed Random Learning Signals Allow for Feedforward Training of Deep Neural Networks. *Frontiers in Neuroscience*, 15:20, 2021. ISSN 1662-453X. doi: 10.3389/fnins.2021.629892. URL https://www.frontiersin.org/article/10.3389/fnins.2021.629892.

[13] Joel A. Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Streaming Low-Rank Matrix Approximation with an Application to Scientific Simulation. *SIAM Journal on Scientific Computing*, 41(4):A2430–A2463, July 2019. doi: 10.1137/18M1201068. URL https://doi.org/10.1137/18M1201068.

[14] Per-Gunnar Martinsson and Joel A. Tropp. Randomized Numerical Linear Algebra: Foundations & Algorithms. *Acta Numerica*, 29:403–572, 2020. doi: 10.1017/S0962492920000021. URL https://doi.org/10.1017/S0962492920000021.

[15] Eldad Haber, Matthias Chung, and Felix J. Herrmann. An effective method for parameter estimation with pde constraints with multiple right hand sides. *SIAM Journal on Optimization*, 22(3), 07 2012. URL http://dx.doi.org/10.1137/11081126X.

[16] Aleksandr Y. Aravkin, Michael P. Friedlander, Felix J. Herrmann, and Tristan van Leeuwen. Robust inversion, dimensionality reduction, and randomized sampling. *Mathematical Programming*, 134(1):101–125, August 2012. doi: 10.1007/s10107-012-0571-6. URL http://www.springerlink.com/content/35rwr101h5736340/.

[17] Tristan van Leeuwen and Felix J. Herrmann. 3d frequency-domain seismic inversion with controlled sloppiness. *SIAM Journal on Scientific Computing*, 36(5):S192–S217, 10 2014. doi: 10.1137/130918629. URL http://epubs.siam.org/doi/abs/10.1137/130918629. (SISC).

[18] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *J. ACM*, 58(2), April 2011. ISSN 0004-5411. doi: 10.1145/1944345.1944349. URL https://doi.org/10.1145/1944345.1944349.

[19] M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 18(3):1059–1076, 1989. doi: 10.1080/03610918908812806. URL https://doi.org/10.1080/03610918908812806.

[20] Raphael A. Meyer, Cameron Musco, Christopher Musco, and David P. Woodruff. Hutch++: Optimal Stochastic Trace Estimation. *arXiv e-prints*, art. arXiv:2010.09649, October 2020. URL https://arxiv.org/abs/2010.09649.

[21] Farbod Roosta-Khorasani and Uri Ascher. Improved bounds on sample size for implicit matrix trace estimators. *Found. Comput. Math.*, 15(5):1187–1212, October 2015. ISSN 1615-3375. doi: 10.1007/s10208-014-9220-1. URL https://doi.org/10.1007/s10208-014-9220-1.

[22] Alice Cortinovis and Daniel Kressner. On randomized trace estimates for indefinite matrices with an application to determinants. *arXiv preprint arXiv:2005.10009*, abs/2005.10009, 2021. URL http://arxiv.org/abs/2005.10009.

[23] K. B. Petersen and M. S. Pedersen. The Matrix Cookbook, October 2008. URL http://www2.imm.dtu.dk/pubdb/p.php?3274. Version 20081110.

[24] Bryan Kaperick. *Diagonal Estimation with Probing Methods*. PhD thesis, Virginia Polytechnic Institute and State University, 2019. URL https://vtechworks.lib.vt.edu/bitstream/handle/10919/90402/Kaperick_BJ_T_2019.pdf.

[25] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. 2015.

[26] W. Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, Justin K. Terry, Furong Huang, and Tom Goldstein. Understanding Generalization Through Visualizations. In *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*, volume 137 of *Proceedings of Machine Learning Research*, pages 87–97. PMLR, 12 Dec 2020. URL http://proceedings.mlr.press/v137/huang20a.html.

[27] 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 <0.5MB model size. *CoRR*, abs/1602.07360, 2016. URL http://arxiv.org/abs/1602.07360.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90. URL https://ieeexplore.ieee.org/document/7780459.

[29] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, 2017. doi: 10.1137/141000671. URL https://julialang.org/assets/research/julia-fresh-approach-BEKS.pdf.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[31] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable Modelling with Flux. *CoRR*, abs/1811.01457, 2018. URL https://arxiv.org/abs/1811.01457.

[32] Mike Innes. Flux: Elegant Machine Learning with Julia. *Journal of Open Source Software*, 2018. doi: 10.21105/joss.00602. URL https://joss.theoj.org/papers/10.21105/joss.00602.

[33] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High Performance Convolutional Neural Networks for Document Processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), October 2006. Université de Rennes 1, Suvisoft. URL https://hal.inria.fr/inria-00112631. http://www.suvisoft.com.

[34] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: Efficient Primitives for Deep Learning. *arXiv preprint arXiv:1410.0759*, 2014. URL https://arxiv.org/abs/1410.0759.

[35] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[36] Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless Stochastic Gradient: Interpolation, Line-Search, and Convergence Rates. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/2557911c1bf75c2b643afb4ecbfc8ec2-Paper.pdf.

[37] A. Saade, F. Caltagirone, I. Carron, L. Daudet, A. Drémeau, S. Gigan, and F. Krzakala. Random projections through multiple optical scattering: Approximating kernels at the speed of light. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6215–6219, 2016. doi: 10.1109/ICASSP.2016.7472872.

[38] Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge University Press, November 2018.

A Appendix

A.1 Implementation and code availability

For the anonymous submission, we provide the software in a `.zip` format with author information removed and will replace it by the GitHub repository after review. The directory `paper` contains the scripts to reproduce the figures. However, this software is intended to be a usable software package rather than only a set of runnable examples that can be easily plugged into existing framework seamlessly. The code is therefore organized to be installed and used as a standard `pip` and Julia package.

Our probing algorithm is implemented both in Julia, using `LinearAlgebra.BLAS` on CPU and `CUDA.CUBALS` on GPU for the linear algebra computations, and in PyTorch using standard linear algebra utilities. The Julia interface is designed so that preexisting networks can be reused as we are overloading `rrule` (see ChainRulesCore.jl) to switch easily between the conventional true gradient (NNlib.jl) and ours. The PyTorch implementation defines a new layer that can be swapped for the conventional convolutional layer, `torch.nn.Conv2d` or `torch.nn.Conv3d`, in any network using the `convert_net` utility function.

A.2 Proofs of Preposition 1 and Theorem 1

For a square matrix $\mathbf{A}$, let $G(\mathbf{A})$ be the trace estimator:

$$G(\mathbf{A}) = \frac{1}{r} \sum_{j=1}^{r} \mathbf{z}_j^\top \mathbf{A} \mathbf{z}_j$$

where $\mathbf{z}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_N)$ be i.i.d. Gaussian vectors. We now prove the proposition and theorem stated in Section 2.

A.2.1 Proof of Proposition 1

We restate Proposition 1 here.

Proposition 1. *Let $\mathbf{A} \in \mathbb{R}^{N \times N}$ be a square matrix. Then for any small number $\delta > 0$, with probability $1 - \delta$,*

$$|G(\mathbf{A}) - \mathrm{tr}(\mathbf{A})| \leq \left(\frac{4\|\mathbf{A}\|_2}{r} \log \frac{2}{\delta} + \frac{2\|\mathbf{A}\|_F}{\sqrt{r}} \log^{1/2} \frac{2}{\delta} \right).$$

The proof uses the following result on trace estimation of symmetric matrices.

Lemma 1 (Theorem 5 of [22]). *Let $\mathbf{B} \in \mathbb{R}^{N \times N}$ be symmetric. Then*

$$P(|G(\mathbf{B}) - \mathrm{tr}(\mathbf{B})| \geq \epsilon) \leq 2 \exp\left(-\frac{r\epsilon^2}{4\|\mathbf{B}\|_F^2 + 4\epsilon\|\mathbf{B}\|_2} \right)$$

for all $\epsilon > 0$.

Proof of Proposition 1. For a symmetric matrix $\mathbf{B}$, Lemma 1 immediately implies that for any small number $\delta > 0$, with probability $1 - \delta$,

$$|G(\mathbf{B}) - \mathrm{tr}(\mathbf{B})| \leq \frac{4\|\mathbf{B}\|_2}{r} \log \frac{2}{\delta} + \frac{2\|\mathbf{B}\|_F}{\sqrt{r}} \log^{1/2} \frac{2}{\delta}.$$

Now for our asymmetric $\mathbf{A}$, let $\mathbf{B} = \frac{\mathbf{A} + \mathbf{A}^\top}{2}$. Then $G(\mathbf{A}) = G(\mathbf{B})$, $\mathrm{tr}(\mathbf{A}) = \mathrm{tr}(\mathbf{B})$, $\|\mathbf{B}\|_2 \leq \|\mathbf{A}\|_2$, and $\|\mathbf{B}\|_F \leq \|\mathbf{A}\|_F$, then the proposition follows. □

A.2.2 Preparation lemmas for Theorem 2

Lemma 2. *Let $\mathbf{A} \in \mathbb{R}^{N \times N}$ be a square matrix, $\mathbf{z}_j, \mathbf{x}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_N)$ be random Gaussian vectors for $j = 1, \ldots, r$, and all the $\mathbf{x}_j$ and $\mathbf{z}_j$ are independent of each other. Then for any δ with probability $1 - \delta$,*

$$\left| \frac{1}{r} \sum_{j=1}^{r} \mathbf{z}_j^\top \mathbf{A} \mathbf{x}_j \right| \leq c \left(\frac{\|\mathbf{A}\|_2}{r} \log \frac{2}{\delta} + \frac{\|\mathbf{A}\|_F}{\sqrt{r}} \log^{1/2} \frac{2}{\delta} \right)$$

13

where c is some absolute constant independent of r.

proof of Lemma 2. Set $T := \frac{1}{r} \sum\limits_{j=1}^{r} \mathbf{z}_j^\top \mathbf{A} \mathbf{x}_j$. For each summand, we have

$$\mathbf{z}_j^\top \mathbf{A} \mathbf{x}_j = \mathbf{z}_j^\top \mathbf{U} \mathbf{S} \mathbf{V}^\top \mathbf{x}_j = \tilde{\mathbf{z}}_j^\top \mathbf{S} \tilde{\mathbf{x}}_j = \sum_t s_t \tilde{\mathbf{z}}_j[t] \tilde{\mathbf{x}}_j[t] \equiv \sum_t s_t f_{j,t}, \tag{9}$$

where the first equality used the singular value decomposition $\mathbf{A} = \mathbf{U} \mathbf{S} \mathbf{V}^\top$, in the second equality, we defined $\tilde{\mathbf{z}}_j = \mathbf{U}^\top \mathbf{z}_j$ and $\tilde{\mathbf{x}}_j = \mathbf{V}^\top \mathbf{x}_j$, which are still Gaussian. In the third equality, we used s_t to denote the tth diagonal entry of $\mathbf{S}$ and $\tilde{\mathbf{z}}_j[t]$ and $\tilde{\mathbf{x}}_j[t]$ to denote the tth entry of $\tilde{\mathbf{z}}_j$ and $\tilde{\mathbf{x}}_j$, respectively. In the last equality, we defined $f_{j,t} := \tilde{\mathbf{z}}_j[t] \tilde{\mathbf{x}}_j[t]$. Since $\mathbf{z}_j$ and $\mathbf{x}_j$ are i.i.d., so are $f_{j,t}$. And since $f_{j,t}$ are products of independent sub-Gaussian random variables, they obey the sub-exponential distribution, i.e.,

$$\|f_{j,t}\|_{\psi_1} \leq \|\tilde{\mathbf{z}}_j[t]\|_{\psi_2} \|\tilde{\mathbf{x}}_j[t]\|_{\psi_2} = c^2,$$

where $\| \cdot \|_{\psi_1}$ denotes the sub-exponential norm and $\| \cdot \|_{\psi_2}$ the sub-Gaussian norm. We also used the property that there is a constant c, such that for any σ, a Gaussian variable $a \sim \mathcal{N}(0, \sigma^2)$ has a sub-Gaussian norm $\|a\|_{\psi_2} \leq c\sigma$, and this property is applied on $\tilde{\mathbf{z}}_j[t]$ and $\tilde{\mathbf{x}}_j[t]$ who are both $\mathcal{N}(0,1)$ variables due to the rotation invariance of Gaussian vectors.

Apply the Bernstein inequality [38] to $T = \frac{1}{r} \sum_{j,t} s_t f_{j,t}$, we obtain

$$P(|T| \geq \tilde{t}) \leq e^{-c' \min\{\frac{r\tilde{t}^2}{4\|\mathbf{A}\|_F^2}, 4\frac{r\tilde{t}}{\|\mathbf{A}\|_2}\}},$$

where c' is some absolute constant. Letting δ to be the right hand side probability, the above implies

$$P\left(|T| \geq c \left(\frac{\|\mathbf{A}\|_2}{r} \log \frac{2}{\delta} + \frac{\|\mathbf{A}\|_F}{\sqrt{r}} \log^{1/2} \frac{2}{\delta} \right) \right) \leq \delta,$$

with some constant c. Then the lemma is proved. □

Lemma 3. *Let $\mathbf{A} \in \mathbb{R}^{N \times N}$ be a square matrix, $\mathbf{z}_j, \mathbf{x}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_N)$ be random Gaussian vector for $j = 1, \ldots, r$, and all the $\mathbf{x}_j$'s and $\mathbf{z}_j$'s are independent of each other. Let $\mathbf{y}_j$ be the random vector that equals to $\mathbf{x}_j$ with probability p and equals to 0 with probability $1 - p$. Then for any $\delta > 0$ with probability over $1 - \delta - 2e^{-rp^2/2}$,*

$$\left| \frac{1}{r} \sum_{j=1}^{r} \mathbf{z}_j^\top \mathbf{A} \mathbf{y}_j \right| \leq c \left(\frac{\|\mathbf{A}\|_2}{r} \log \frac{2}{\delta} + \frac{\sqrt{p}\|\mathbf{A}\|_F}{\sqrt{r}} \log^{1/2} \frac{2}{\delta} \right),$$

where c is some absolute constant independent of r.

Proof. The proof is very similar to that of Lemma 2. Set $T := \frac{1}{r} \sum\limits_{j=1}^{r} \mathbf{z}_j^\top \mathbf{A} \mathbf{y}_j$. Let $g_j = 1_{\{\mathbf{y}_i \neq \mathbf{0}\}}$ be the indicator function of whether $\mathbf{y}_j$ is non-zero. Then clearly $\mathbf{y}_j = \mathbf{x}_j g_j$. For each summand, we have

$$\mathbf{z}_j^\top \mathbf{A} \mathbf{y}_j = \mathbf{z}_j^\top \mathbf{U} \mathbf{S} \mathbf{V}^\top \mathbf{x}_j g_j = \tilde{\mathbf{z}}_j^\top \mathbf{S} \tilde{\mathbf{x}}_j g_j = \sum_t s_t \tilde{\mathbf{z}}_j[t] \tilde{\mathbf{x}}_j[t] g_j \equiv g_j \sum_t s_t f_{j,t}, \tag{10}$$

where in the first equality, we used the singular value decomposition, $\mathbf{A} = \mathbf{U} \mathbf{S} \mathbf{V}^\top$, and in the second equality, we defined $\tilde{\mathbf{z}}_j = \mathbf{U}^\top \mathbf{z}_j$ and $\tilde{\mathbf{x}}_j = \mathbf{V}^\top \mathbf{x}_j$. In the third equality, we used s_t to denote the tth diagonal entry of $\mathbf{S}$ with $\tilde{\mathbf{z}}_j[t]$ and $\tilde{\mathbf{x}}_j[t]$ denoting the tth entry of $\tilde{\mathbf{z}}_j$ and $\tilde{\mathbf{x}}_j$, respectively. In the last equality, we defined $f_{j,t} := \tilde{\mathbf{z}}_j[t] \tilde{\mathbf{x}}_j[t]$. From Lemma 2, $f_{i,j}$ follow sub-exponential distributions, i.e., $\|f_{j,t}\|_{\psi_1} \leq c^2$.

Conditional on g_j, applying the Bernstein inequality to $\tilde{T} := \frac{1}{\sum_j 1_{\{g_j \neq 0\}}} \sum_{j,t} s_t f_{j,t} g_j$, we obtain

$$P(|\tilde{T}| \geq \tilde{t}) \leq e^{-c' \min\left\{ \frac{\tilde{t}^2 \sum_i 1_{\{g_i \neq 0\}}}{4 \|\mathbf{A}\|_F^2}, 4 \frac{\tilde{t} \sum_j 1_{\{g_j \neq 0\}}}{\|\mathbf{A}\|_2} \right\}}.$$

Letting δ to be the right hand side probability, the above implies

$$P\left(|\tilde{T}| \geq c\left(\frac{\|\mathbf{A}\|_2}{\sum_j 1_{\{g_j \neq 0\}}} \log \frac{2}{\delta} + \frac{\|\mathbf{A}\|_F}{\sqrt{\sum_j 1_{\{g_j \neq 0\}}}} \log^{1/2} \frac{2}{\delta}\right)\right) \leq \delta.$$

Since $\sum_j 1_{\{g_j \neq 0\}} \sim \mathbf{B}(r, p)$, we then have with probability of $1 - 2e^{rp^2/2}$, $3rp/2 \geq \sum_j 1_{\{g_j \neq 0\}} \geq rp/2$. Plugging this estimate into the above, we have

$$P\left(|\tilde{T}| \geq c\left(\frac{\|\mathbf{A}\|_2}{pr} \log \frac{2}{\delta} + \frac{\|\mathbf{A}\|_F}{\sqrt{pr}} \log^{1/2} \frac{2}{\delta}\right)\right) \leq \delta.$$

Then, with this bound of $|\tilde{T}|$, we have

$$|T| = \left|\frac{1}{r}\sum_{j=1}^{r} \mathbf{z}_j^\top \mathbf{A}\mathbf{y}_j\right| = \left|\frac{\sum_j 1_{\{g_j \neq 0\}}}{r}\tilde{T}\right| \leq c\left(\frac{\|\mathbf{A}\|_2}{r} \log \frac{2}{\delta} + \frac{\sqrt{p}\|\mathbf{A}\|_F}{\sqrt{r}} \log^{1/2} \frac{2}{\delta}\right),$$

which is the statement of this lemma. □

A.2.3 Multi-channel result: Proof of Theorem 1

For simplicity, we assume the number of input and output channels are the same and both equal to C. Let

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}^{1,1} & \cdots & \mathbf{A}^{1,C} \\ \vdots & & \vdots \\ \mathbf{A}^{C,1} & \cdots & \mathbf{A}^{C,C} \end{pmatrix}$$

the goal is estimate $\mathrm{tr}(\mathbf{A}^{m,n})$, for $m, n = 1, \ldots, C$.

Let $\mathbf{Z} \in \mathbb{R}^{NC \times r}$ be the "orthogonalized" matrix of r probing vectors. We further denote by $\mathbf{z}_{n,\cdot}$ the nth row block of $\mathbf{Z}$, which is the block containing the $(1 + (n-1))^{th}$ to the $(nN)^{th}$ rows of $\mathbf{Z}$. We also denote by $\mathbf{z}_j \in \mathbb{R}^{NC}$, $j = 1, \ldots, r$ the jth column of $\mathbf{Z}$ (i.e., the jth probing vector), and by $\mathbf{z}_{n,j}$ the nth block of $\mathbf{z}_j$. For any $n = 1, \ldots, C$, $j = 1, \ldots, r$, we define $\mathbf{z}_{n,j}$ as

$$\mathbf{z}_{n,j} \sim \begin{cases} \mathcal{N}(\mathbf{0}, \mathbf{I}_N), & \text{with probability } p_n \\ 0, & \text{with probability } 1 - p_n \end{cases}. \tag{11}$$

For different values of (n, j), $\mathbf{z}_{n,j}$ are independent of each other. Here p_n is a predefined probability for randomly generating each nonzero block.

With these probing vectors, we defined the following estimator for $\mathrm{tr}(\mathbf{A}^{m,n})$

$$G^{m,n}(\mathbf{A}) := \frac{1}{\mathrm{nnz}(\mathbf{z}_{n,\cdot})} \sum_{j=1}^{r} \mathbf{z}_{n,j}^\top (\mathbf{A}z_j)_m,$$

where $\mathrm{nnz}(\mathbf{z}_{n,\cdot})$ is the number of nonzeros columns of $\mathbf{z}_{n,\cdot}$, which is also a random variable.

Theorem 2. *Let $p = \min_n p_n$, r be the number of probing vectors. For any small number $\delta > 0$, with probability over $1 - \delta - 2Ce^{-rp^2/2}$, we have for any $n, m = 1, \ldots, C$,*

$$|G^{m,n}(\mathbf{A}) - \mathrm{tr}(\mathbf{A}^{m,n})| \leq c\left(\frac{\sum_{k=1}^{C}\|\mathbf{A}^{m,k}\|}{p_j r} \log \frac{C^2}{\delta} + \frac{\frac{1}{\sqrt{p_j}}\|\mathbf{A}^{m,n}\|_F + \sum_{j=1,j\neq n}^{C}\sqrt{\frac{p_j}{p_n}}\|\mathbf{A}^{m,j}\|_F}{\sqrt{r}} \log^{1/2}\frac{C^2}{\delta}\right),$$

where c is an absolute constant and C is the number of channels. For sufficiently large number of probing vectors, the above bound reduces to

$$|G^{m,n}(\mathbf{A}) - \mathrm{tr}(\mathbf{A}^{m,n})| \leq c \cdot \frac{\frac{1}{\sqrt{p_n}}\|\mathbf{A}^{m,n}\|_F + \sum_{j=1,j\neq n}^{C}\sqrt{\frac{p_j}{p_n}}\|\mathbf{A}^{m,j}\|_F}{\sqrt{r}} \log^{1/2}\frac{C^2}{\delta}.$$

15

Proof. First we show the estimator is unbiased. For simplicity of notation, let $g_{n,l} = 1_{\{\mathbf{z}_{n,l} \neq \mathbf{0}\}}$ be the random variable that indicates whether $\mathbf{z}_{n,l}$ is non-zero. By definition, conditional on $g_{n,l} = 1$, $\mathbf{z}_{n,l}$ is Gaussian, and this Gaussian distribution is independent of $g_{n,l}$. Also $\sum_l g_{n,l} \sim \mathbf{B}(r, p_n)$ is Binomial distribution with probability p_j. Then the estimator can be written as

$$G^{m,n}(\mathbf{A}) := \frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \mathbf{z}_{n,l}^{\top}(\mathbf{A}z_l)_m g_{n,l}.$$

Taking the expectation, we have

$$\mathbb{E}(G^{m,n}(\mathbf{A})) = \mathbb{E}_g \left[\mathbb{E}_{\mathbf{Z}|g} \left(\frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \mathbf{z}_{n,l}^{\top}(\mathbf{A}z_l)_m g_{n,l} \right) \right]$$

$$= \mathbb{E}_g \left[\frac{1}{\sum_l g_{n,l}} \mathbb{E}_{\mathbf{Z}|g} \left(\sum_{l=1}^{r} \sum_{k=1}^{C} \mathbf{z}_{n,l}^{\top} \mathbf{A}^{m,k} \mathbf{z}_{k,l} g_{n,l} \right) \right]$$

$$= \mathbb{E}_g \left[\frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \sum_{k=1}^{C} \mathbb{E}_{\mathbf{Z}|g} \left(\mathrm{tr}(\mathbf{A}^{m,k} \mathbf{z}_{k,l} \mathbf{z}_{n,l}^{\top}) g_{n,l} \right) \right]$$

$$= \mathbb{E}_g \left[\frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \sum_{k=1}^{C} \mathrm{tr}(\mathbf{A}^{m,k} \mathbb{E}_{\mathbf{Z}|g}(\mathbf{z}_{k,l} \mathbf{z}_{n,l}^{\top})) g_{n,l} \right]$$

$$= \mathbb{E}_g \left[\frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \mathbf{A}^{m,n} g_{n,l} \right]$$

$$= \mathrm{tr}(\mathbf{A}^{m,n}),$$

where the second to last equality used $\mathbb{E}_{\mathbf{Z}|g}(\mathbf{z}_{n,l}\mathbf{z}_{n,l}^{\top}) = g_{n,l}\mathbf{I}_N$ and $\mathbb{E}_{\mathbf{Z}|g}(\mathbf{z}_{k,l}\mathbf{z}_{n,l}^{\top}) = 0$ for $k \neq n$. Then we estimate the large deviation,

$$G^{m,n}(\mathbf{A}) - \mathrm{tr}(\mathbf{A}^{m,n}) = \frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \sum_{k=1}^{C} \mathbf{z}_{n,l}^{\top} \mathbf{A}^{m,k} \mathbf{z}_{k,l} g_{n,l} - \mathrm{tr}(\mathbf{A}^{m,n})$$

$$= \frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \mathbf{z}_{n,l}^{\top} \mathbf{A}^{m,n} \mathbf{z}_{n,l} g_{n,l} - \mathrm{tr}(\mathbf{A}^{m,n}) + \sum_{k=1,k\neq n}^{C} \left(\frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \mathbf{z}_{n,l}^{\top} \mathbf{A}^{m,k} \mathbf{z}_{k,l} g_{n,l} \right).$$

The first term is bounded by Proposition 1, and the second term is bounded by Lemma 2. Explicitly, $\sum_l g_{n,l}$ is the number of probing vectors we are using in probing $G^{m,n}(\mathbf{A})$, so conditional on $g_{n,l}$, Proposition 1 yields an upper bound on the first term in the above right hand side, with probability $1 - \delta'$

$$\left| \frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \mathbf{z}_{n,l}^{\top} \mathbf{A}^{m,n} \mathbf{z}_{n,l} g_{n,l} - \mathrm{tr}(\mathbf{A}^{m,n}) \right| \leq \frac{4\|\mathbf{A}^{m,n}\|_2}{\sum_l g_{n,l}} \log \frac{2}{\delta'} + \frac{2\|\mathbf{A}^{m,n}\|_F}{\sqrt{\sum_l g_{n,l}}} \log^{1/2} \frac{2}{\delta'}. \quad (12)$$

By Lemma 3, the bound on the second term is, with probability $1 - \delta' - 2Ce^{-rp^2/2}$,

$$\left| \sum_{k=1,k\neq n}^{C} \left(\frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \mathbf{z}_{n,l}^{\top} \mathbf{A}^{m,k} \mathbf{z}_{k,l} g_{n,l} \right) \right| \leq \sum_{k=1,k\neq n}^{C} \left| \frac{1}{\sum_l g_{n,l}} \sum_{l=1}^{r} \mathbf{z}_{n,l}^{\top} \mathbf{A}^{m,k} \mathbf{z}_{k,l} g_{n,l} \right|$$

$$\leq c \sum_{k=1,k\neq n}^{C} \left(\frac{\|\mathbf{A}^{m,k}\|_2}{\sum_l g_{n,l}} \log \frac{2}{\delta'} + \frac{\sqrt{p_k}\|\mathbf{A}^{m,k}\|_F}{\sqrt{\sum_l g_{n,l}}} \log^{1/2} \frac{2}{\delta'} \right). \quad (13)$$

Since $\sum_l g_{n,l} \sim \mathbf{B}(r, p_n)$, so with probability over $1 - Ce^{-rp_n^2/2}$, we have $\sum_l g_{n,l} > p_n r/2$. Combining this, (12), and (13) gives

$$|G^{m,n}(\mathbf{A}) - \mathrm{tr}(\mathbf{A}^{m,n})| \leq c \left(\frac{\sum_{k=1}^{C} \|\mathbf{A}^{m,k}\|}{p_n r} \log \frac{2}{\delta'} + \frac{\frac{1}{\sqrt{p_n}}\|\mathbf{A}^{m,n}\|_F + \sum_{k=1,k\neq n}^{C} \sqrt{\frac{p_k}{p_n}}\|\mathbf{A}^{m,k}\|_F}{\sqrt{r}} \log^{1/2} \frac{2}{\delta'} \right).$$

For sufficiently large r, the second term is dominant and the bound reduces to

$$|G^{m,n}(\mathbf{A}) - \mathrm{tr}(\mathbf{A}^{m,n})| \leq c' \frac{\frac{1}{\sqrt{p_n}}\|\mathbf{A}^{m,n}\|_F + \sum_{k=1,k\neq n}^{C} \sqrt{\frac{p_k}{p_n}}\|\mathbf{A}^{m,k}\|_F}{\sqrt{r}} \log^{1/2} \frac{1}{\delta'}.$$

So far the result is for a given pair of m, n. By the union bound of probability, the probability of failure for any m, n is $\delta = \delta'C^2$. Then with probability over $1 - \delta - 3Ce^{-rp^2/2}$, we have

$$|G^{m,n}(\mathbf{A}) - \mathrm{tr}(\mathbf{A}^{m,n})| \leq c \cdot \frac{\frac{1}{\sqrt{p_n}}\|\mathbf{A}^{m,n}\|_F + \sum_{k=1,k\neq n}^{C} \sqrt{\frac{p_k}{p_n}}\|\mathbf{A}^{m,k}\|_F}{\sqrt{r}} \log^{1/2} \frac{C^2}{\delta}.$$

$\square$

A.3 Networks

We describe here the network architectures used in our experiments. The architecture used in the MNIST experiments are standard architectures inspired by existing networks for this dataset. The CIFAR10 architecture is intentionally chosen to be mostly convolutional and obtained from [9].

Table 2: MNIST network and sizes for training with our Julia implementation for a batchsize B.

Layer	kernel size	Input size ($C_o \times N_x \times N_y$)	Output size ($C_o \times N_x \times N_y$)
Conv2d	(3, 3)	B×1×28×28	B×16×28×28
ReLU	–	B×16×28×28	B×16×28×28
Maxpool	(2, 2)	B×16×28×28	B×16×14×14
Conv2d	(3, 3)	B×16×14×14	B×32×14×14
ReLU	–	B×32×14×14	B×32×14×14
Maxpool	(2, 2)	B×32×14×14	B×32×7×7
Conv2d	(3, 3)	B×32×7×7	B×32×7×7
ReLU	–	B×32×7×7	B×32×7×7
Maxpool	(2, 2)	B×32×7×7	B×32×3×3
Flatten	–	B×32×3×3	B×288
Dense	–	B×288	B×10

Table 3: MNIST network and sizes for training with PyTorch on the MNIST dataset for a batch size B.

Layer	kernel size	Input size ($C_o \times N_x \times N_y$)	Output size ($C_o \times N_x \times N_y$)
Conv2d	(3, 3)	B×1×28×28	B×32×28×28
ReLU	–	B×32×28×28	B×32×28×28
Conv2d	(3, 3)	B×32×28×28	B×64×28×28
ReLU	–	B×64×28×28	B×64×28×28
Maxpool	(2, 2)	B×64×28×28	B×64×14×14
Dropout	–	B×64×14×14	B×64×14×14
Flatten	–	B×64×14×14	B×12544
Dense	–	B×12544	B×128
ReLU	–	B×128	B×128
Dropout	–	B×128	B×128
Dense	–	B×128	B×10
Log Softmax	–	B×10	B×10

Table 4: CIFAR10 network and sizes for the training with PyTorch on the CIFAR10 dataset for a batch size B.

Layer	kernel size	Input size ($C_o \times N_x \times N_y$)	Output size ($C_o \times N_x \times N_y$)
Conv2d	(5, 5)	B×3×32×32	B×16×32×32
ReLU	–	B×16×32×32	B×16×32×32
Conv2d	(5, 5)	B×16×32×32	B×32×32×32
ReLU	–	B×32×32×32	B×32×32×32
AvgPool	(2, 2)	B×32×32×32	B×32×16×16
Conv2d	(5, 5)	B×32×16×16	B×32×16×16
ReLU	–	B×32×16×16	B×32×16×16
Conv2d	(5, 5)	B×32×16×16	B×32×16×16
ReLU	–	B×32×16×16	B×32×16×16
AvgPool	(2, 2)	B×32×16×16	B×32×8×8
Flatten	–	B×32×8×8	B×2048
Dense	–	B×2048	B×10
Log Softmax	–	B×10	B×10

A.4 Training parameters

We now detail the training hyperparameters for the results presented in section 4.

A.4.1 MNIST with Julia

- NVIDIA Quadro P1000 GPU
- 20 epochs
- ADAM with initial learning rate of 0.003
- MNSIST dataset for varying batchsize B and probing size r
- Julia implementation

A.4.2 MNIST with PyTorch

- NVIDIA Tesla K80 (Azure NC24 4xK80, one K80 per case) GPU
- 50 epochs
- Stochastic LineSearch (SLS [36]) with initial learning rate of 1.0 and default SLS parameters
- MNSIST dataset for varying batch size and probing size
- PyTorch implementation

A.4.3 CIFAR10 with PyTorch

- NVIDIA Tesla K80 (Azure NC6) GPU
- 100 epochs
- Stochastic Gradient Descent optimizer
- Initial learning rate of 0.001 with cosine annealing scheduler. 1.5× learning rate with probing.
- CIFAR10 dataset
- PyTorch implementation

A.5 Wall-clock benchmarks

We show in the following figures the wall-clock benchmark results discussed in Section 3 with the hardware description.
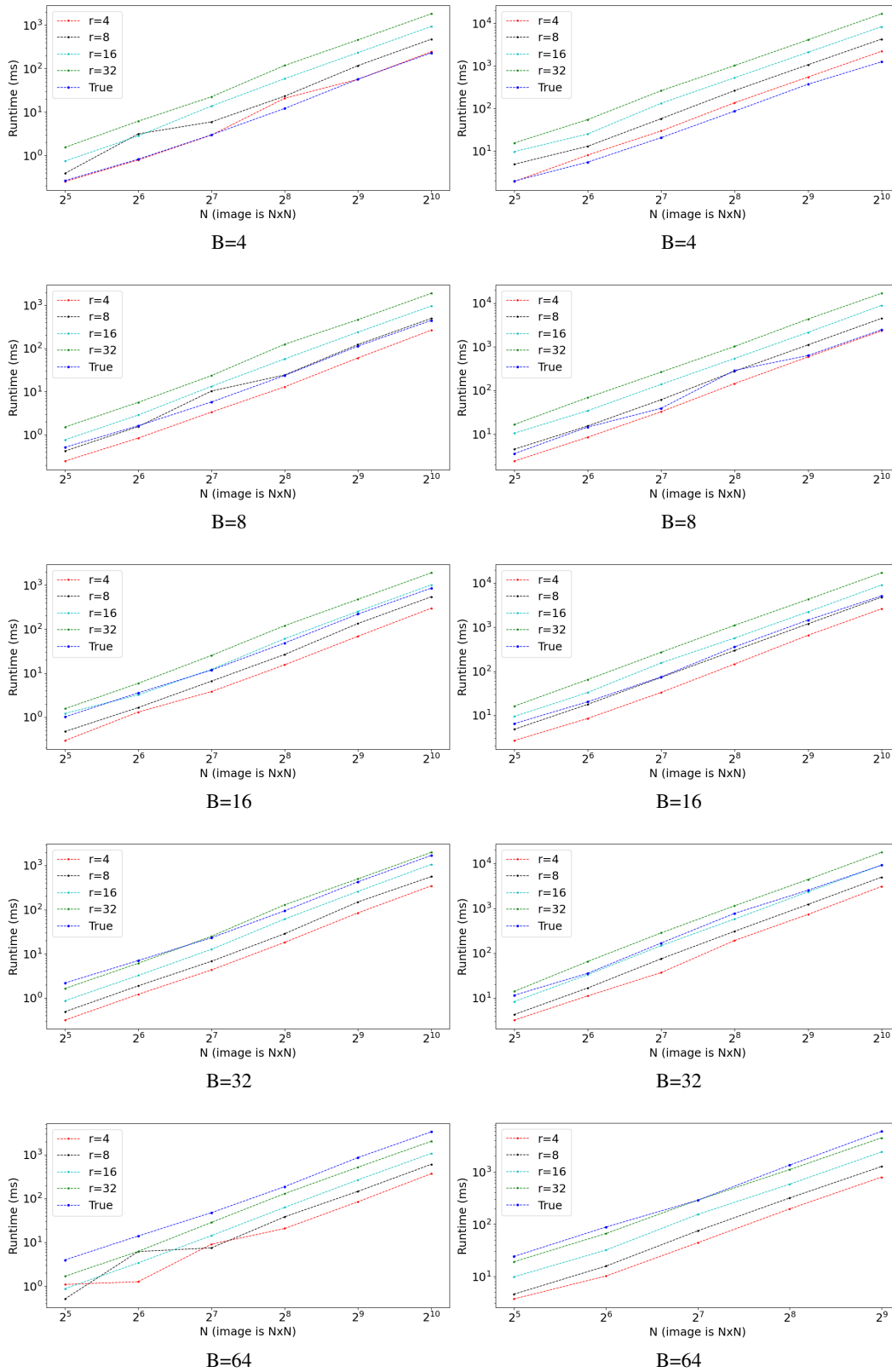
Figure 8: CPU benchmark on a *Intel(R) Xeon(R) CPU E3-1270 v6 @ 3.80GHz* node. The left column contains the runtimes for 4 channels and the right column for 32 channels. We can see that for large images and batchsizes, our implementation provides a consequent performance gain of up to $10\times$.
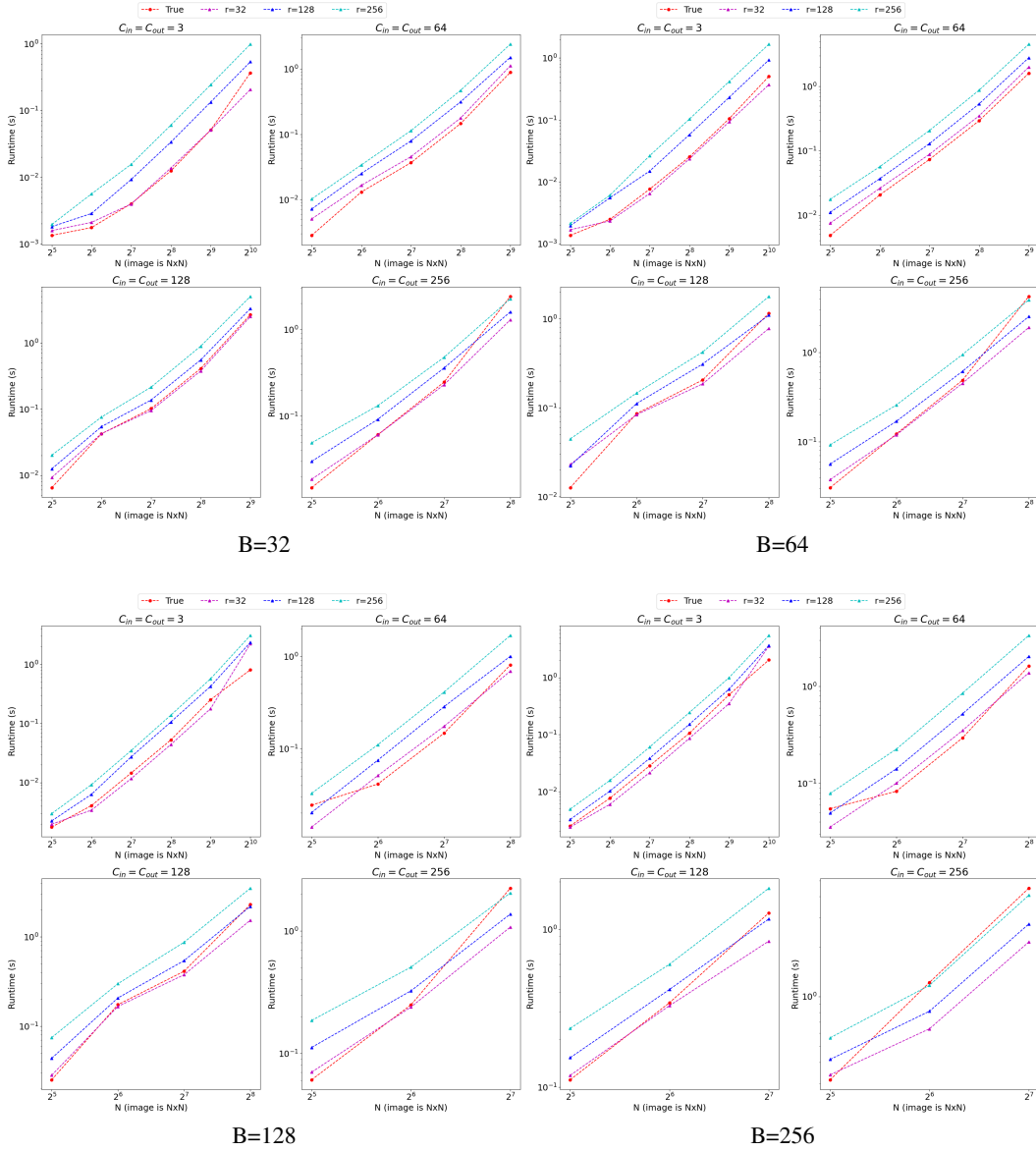
Figure 9: GPU benchmark on a *Tesla K80* (Azure NC6 instance) for a single gradient for varying batchsizes B, image sizes N and number of channel $C_{\text{in}} = C_{\text{out}}$. We observe that for larger scale problems we perform as well if not better than state-of-the-art CuDNN kernels such as for $B = 256, C_{\text{in}} = 256$.