This CVPR 2020 workshop paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Dithered backprop: A sparse and quantized backpropagation algorithm for more efficient deep neural network training

Simon Wiedemann, Temesgen Mehari, Kevin Kepp, Wojciech Samek

Department of Video Coding & Analytics, Fraunhofer Heinrich-Hertz Institut
Berlin, Germany

{simon.wiedemann,temesgen.mehari,kevin.kepp,wojciech.samek}@hhi.fraunhofer.de

Abstract

Deep Neural Networks are successful but highly computationally expensive learning systems. One of the main sources of time and energy drains is the well known backpropagation (backprop) algorithm, which roughly accounts for 2/3 of the computational cost of training. In this work we propose a method for reducing the computational complexity of backprop, which we named dithered backprop. It consists on applying a stochastic quantization scheme to intermediate results of the method. The particular quantisation scheme, called non-subtractive dither (NSD), induces sparsity which can be exploited by computing efficient sparse matrix multiplications. Experiments on popular image classification tasks show that it induces 92% sparsity on average across a wide set of models at no or negligible accuracy drop in comparison to state-of-the-art approaches, thus significantly reducing the computational complexity of the backward pass. Moreover, we show that our method is fully compatible to state-of-the-art training methods that reduce the bit-precision of training down to 8-bits, as such being able to further reduce the computational requirements. Finally we discuss and show potential benefits of applying dithered backprop on a distributed training settings, in that communication as well as compute efficiency may increase simultaneously with the number of participant nodes.

1. Introduction

Deep neural networks (DNNs) are powerful machine learning systems for recognizing patterns in large amounts of data. They became very popular through recent successes in computer vision, language understanding and other areas of computer science [11]. However, DNNs need to undergo a highly computationally expensive training procedure in order to extract meaningful representations from the data. For instance, [23] showed that the training process of state-of-the-art neural network architectures can produce 284 tons of carbon dioxide, nearly five times the lifetime emissions of an average car. Therefore, in order to mitigate the impact of training and/or allow for models to be trained on resource-constrained devices, more efficient algorithms have to be designed.

The backpropagation (backprop) algorithm [18] is most often applied when gradient-based optimization techniques are selected for training DNNs. However, it involves the computation of many dot products between large tensors, therefore playing a major role in the computational cost of the training procedure. Techniques such quantization and/or sparsity can be employed in order to reduce the complexity of the dot products. However, when applied in a naïve manner they may induce biased, non-linear errors which can have catastrophic effects for the convergence of the overall training algorithm.

In this work we aim to minimize the computational complexity of the backprop algorithm by carefully studying the error induced by quantization. Concretely, we propose to apply a particular type of stochastic quantization technique to the gradients of the preactivation values, known as non-subtractive dithering (NSD) [22]. NSD does not only reduce the precision of the preactivation values, but it also induces sparsity. As such, we attain sparse tensors with low precision non-zero values, properties that can be exploited in order to reduce the computational cost of the dot products they are involved in. Our contributions can be summarized as follows:

- We reduce the computational complexity of the most expensive components of the backprop algorithm by applying stochastic quantization techniques to the gradients of the preactivation values, inducing sparsity + low-precision non-zero values.

- We show on extensive experiments that we can reach a significant amount of sparsity (between 76%-99%)

across a wide set of neural network models, while maintaining the non-zero values below/equal to 8-bit precision without affecting the final accuracy neither the convergence speed.

- Finally, we discuss the positive properties that emerge when applying dithered backprop in a distributed setting. Concretely, we show that we can reduce the computational cost for training at each node by increasing the number of participant nodes.

2. Related Work

A lot of research is dedicated to improve the performance at inference time [6, 25, 28]. However, less research has focused on designing more efficient training algorithms, in particular a more efficient backward pass. In the following we discuss some of the proposed approaches.

Precision Quantization. Most of preceding work on efficient neural network training uses *Precision Quantization*. In the context of deep learning that means to transform activation, weight and gradient values to representations of lower precision than the regular single-point floating point standard. It has been shown that this can significantly reduce the time and space complexity of deep learning models [8, 9, 7, 15, 30, 14, 3].
[8] were among the first to show that it is feasible to quantize parts of state-of-the-art models without or just with negligible loss of accuracy using 10-bit multiplications. Subsequently, more people followed the example and quantized successfully whole models to 16-bit representations [15, 12]. Later, even ternary and binary weight quantizations were applied, while keeping the gradients and errors in the backward pass in full precision [7, 26]. However, these approaches sacrifice accuracy over the baseline networks. [3] accomplished to quantize weights, activations and all gradient calculations, except for the weight updates, to 8-bit. A 16-bit copy of the backpropagated gradient is saved to compute a full-precision weight update. They argue that the extra time required for this matrix multiplication is comparably small to the time required to backpropagate the error gradient and that in most layers these calculations can be made in parallel.

Efficient Approximations. Other work investigated the possible speed up gaining from efficient approximations of matrix multiplications in the backward pass. [1] reduces the complexity of the matrix multiplication by approximations through a form of column-row sampling. Using an efficient sampling heuristic, this approach achieves up to 80% reduced computation but the authors provide no analysis of the induced noise variance contained the weight gradients and its impact on the generalization performance. The meProp algorithm [24] sparsifies the pre-activation gradients by selecting the k elements with the largest magnitude. They leverage sparse matrix multiplications for a more efficient backward pass. However, since this quantization function is deterministic and operates on vectors, it results in *biased* estimates of the weight updates which can harm the convergence speed as well as generalization performance of the trained model.

In contrast, we show how *dither functions* can be used to calculate *unbiased* weight updates efficiently, due to their sparsity-inducing property when applied to gradient values. Furthermore, we show how the approach can be combined with state-of-the-art precision quantization methods in order to boost the computational efficiency of the algorithm.

3. Dithered backpropagation

For fully-connected layers the operations that need to be performed per layer during one training iteration are the following (note that these equations are analogous for convolutional layers):

Forward pass

$$z^l = W^l \cdot a^{l-1} + b^l \tag{1}$$
$$a^l = f(z^l)$$

Backward pass

$$\delta_z^l = \delta_a^l \odot f'(z^l)$$
$$\delta_a^{l-1} = (W^l)^T \cdot \delta_z^l \tag{2}$$
$$\delta_W^l = \delta_z^l \cdot (a^{l-1})^T \tag{3}$$

with W, b, z and a being the weight tensor, bias, preactivation and activation values respectively. δ_W, δ_b, δ_z and δ_a denote the error or gradients of the respective quantities. With f we denote the non-linear function whereas with f' its derivative. l is an index referring to a particular layer and T denotes the transpose operation. Finally, the symbols $\cdot$ and $\odot$ denote the dot and Hadamard product respectively.

As one can see, there are three major matrix multiplications involved at each layer during one training iteration, namely, one in the forward pass (equation 1) and two in the backward pass (equation 2 and equation 3). Since up to 90% of the computing time is spent on performing these dot product operations [24], in this work we focus on reducing their computational cost. In particular, notice how the preactivation gradients δ_z^l are present in both matrix multiplications in the backward pass. Hence, in order to save operations, we apply quantization functions that compresses these gradients.

3.1. Non-subtractive dithered quantization (NSD)

For reasons that will become more apparent in the next section, in this work we propose to apply the following

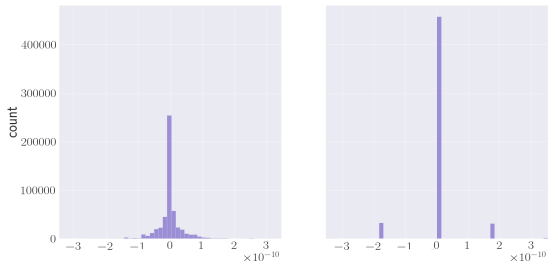

Figure 1: Distribution of preactivation gradient values before δ_z (left) and after $\widetilde{\delta_z}$ (right) NSD quantization. The gradients have become more sparse (higher count of 0 values) and the non-zero values can be represented with low bitwidths (low number of non-zero "buckets"). For instance, this example only 1 bit is required to represented all non-zero values.

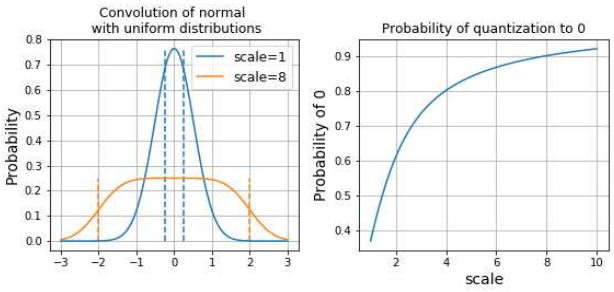

Figure 2: (left) Shape of the probability distribution resulting from the convolution of a gaussian with a uniform distribution, where the uniform distribution samples values between a range $(-\frac{\Delta}{2}, \frac{\Delta}{2})$. The shape depends on the stepsize Δ of the uniform distribution, which is chosen to be $\Delta = s\sigma$ with σ being the standard deviation of the gaussian distribution and $s \in \mathbb{N}$ a scaling factor. The dashed lines indicate the region of values between $(-\frac{\Delta}{2}, \frac{\Delta}{2})$. (right) the probability of a 0 value appearing after quantization at different scale factors. It is calculated by integrating the area between the dashed lines on the left plot. From both plots one can see that sparsity increases with the scaling factor s.

quantization function:

$$\begin{aligned} \widetilde{x} &= Q_\Delta(x + \nu) \\ &= \Delta \lfloor \frac{x + \nu}{\Delta} + \frac{1}{2} \rfloor \end{aligned} \tag{4}$$

with Δ being the quantization step size and $x \in \mathbb{R}$ an input value. $\nu \sim U(-\frac{\Delta}{2}, \frac{\Delta}{2})$ is a random number sampled from the uniform distribution between the open interval $(-\frac{\Delta}{2}, \frac{\Delta}{2})$. The quantization function in equation 4 is sometimes referred as *non-subtractive dither* (NSD) [22] in the source coding literature, with ν being a stochastic *dither signal* that is added to the input before quantization. The main motivation for adding a dither signal before quantization is to decouple the moments of the quantization error $\epsilon = Q_\Delta(x + \nu) - x$ from the input signal x. For instance, it is known that the quantization error of NSD is unbiased and has bounded variance

$$\mathbb{E}[\epsilon] = 0 \tag{5}$$

$$\mathbb{E}[\epsilon^2] < \frac{\Delta^2}{4} \tag{6}$$

3.2. Effects of applying NSD to the gradients

Hence, at each layer l, we now apply NSD to the gradients of the preacitvation values δ_z^l before computing the respective dot products. For large enough stepsizes Δ, NSD will induce sparsity (many zero values) as well as non-zero values with low bitwidth representation (see figure 1).

To make this effect more clear, consider as an example the convolution $f(t) = (G_\sigma * U_\Delta)(t)$ between G_σ a gaussian distribution with mean 0 and standard deviation σ and U_Δ a uniform distribution, sampling values in the range $(-\frac{\Delta}{2}, \frac{\Delta}{2})$. The induced average sparsity is given by the probability of f sampling a value in the same interval, thus

$$P(t = 0) = \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} f(t)$$

As figure 2 shows, the probability of 0 increases with the stepsize value. Naturally, the same applies for the maxmal bit-width of the non-zero values since the probability of a high number appearing after quantization decreases as the stepsize increases.

We can then exploit this sparsity to omit operations when computing the dot product between tensors. The altered equations for the backward pass at each layer are then given by:

$$\widetilde{\delta_z^l} = Q_{\Delta^l}(\delta_z^l + \nu^l) \tag{7}$$

$$\delta_a^{l-1} = (W^l)^T \cdot \widetilde{\delta_z^l} \tag{8}$$

$$\delta_W^l = \widetilde{\delta_z^l} \cdot (a^{l-1})^T \tag{9}$$

with $\nu^l \sim U(-\frac{\Delta^l}{2}, \frac{\Delta^l}{2})$ and $\widetilde{\delta_z^l}$ being the matrix of quantized pre-activation gradients.

Given the above analysis we propose to choose the stepsize at each layer as to be a multiple of the standard deviation, that is, $\Delta^l = s\sigma^l \; \forall l$, with σ^l being the standard deviation of the preactivation gradients and $s \in \mathbb{N}$. s is thus a global scaling factor that controls the trade-off between compute complexity and learning performance. We named our proposed modification of the backprop method ***dithered backprop***. Algorithm 1 shows a pseudocode of the

Algorithm 1 Dithered backprop quantization

procedure NSD(δ_z^l, s) ▷ Quantizes preactivation gradients δ_z^l of layer l
  $\sigma^l \leftarrow \text{std}(\delta_z^l)$ ▷ Computes standard deviation
  $\Delta^l \leftarrow s\sigma^l$
  $\widetilde{\delta_z^l} \leftarrow Q_\Delta(\delta_z^l)$ ▷ As in equation 4
  return $\widetilde{\delta_z^l}$
end procedure

quantization procedure of the preactivation gradients. After quantization, the backward pass as well as the weight update steps remain identical as in the usual algorithm.

3.3. Error statistics and convergence of the method

Due to applying NSD to all δ_z^l, dithered backprop attains perturbed estimates of the weight updates

$$\widetilde{\delta_W^l} = \delta_W^l + \epsilon_W^l$$

with ϵ_W^l being the perturbation error. Hence, this begs the question: how does this error influence the convergence of the training method?

From [4] we know that under mild assumptions regarding the loss function, if a stochastic operator is added to a training algorithm that already converges and generates unbiased estimates of the weight updates with bounded variance, then the respective training algorithm converges as well. Thus, we only need to show that the error of the weight updates is unbiased and has bounded variance, that is

$$\mathbb{E}[\epsilon_W^l] = 0 \; \forall l \tag{10}$$

$$\mathbb{E}[(\epsilon_W^l)^2] < C \; \forall l \tag{11}$$

Although in this work we do not provide a rigorous proof (mainly due to space constraints), it is relatively easy to see that equation 10 and equation 11 are satisfied by modelling the quantization error of the preactivation gradients also as additive noise $\widetilde{\delta_z^l} = \delta_z^l + \epsilon_z^l$, and taking into consideration that: 1) on a per input-sample basis, the backpropagation of the quantization error ϵ_z^l to the weight updates can be modelled by a simple linear map and 2) the error ϵ_z^l satisfies equation 5 and equation 6.

3.4. Computational complexity

Theoretical analysis

When dithered backprop is used for training, some additional computational overhead comes form applying NSD to the gradients of the preactivation values. However, we argue that this cost is asymptotically negligible compared to the cost of performing the subsequent dot products. In the following we will highlight the rationale for the case of fully-connected layers, however, we stress that it also applies analogously to convolutional layers.

Let G be a $(k \times n)$-dimensional matrix whose elements are the gradient of the preactivation values of a particular layer. As can be seen from equation 4 and algorithm 1, applying NSD to G requires: for each element,

1. calculate the standard deviation of the preactivation gradients. This requires 1 multiplication + 1 addition per element.

2. sampling from the uniform distribution between the interval $(-\frac{\Delta}{2}, \frac{\Delta}{2})$. This requires about 2 multiplications + 2 additions + 1 modulo operation.

3. Quantizing the value, which requires 1 addition + 1 multiplication + truncation of decimal bits

Overall, the cost can be approximately reduced to about 9 arithmetic operations per element. Thus, the computational complexity of applying NSD is of order $\mathcal{O}(kn)$. If we now include the cost of performing the subsequent sparse matrix-matrix dot product, then the complexity becomes of order $\mathcal{O}(kn + p_{nz}mkn)$, with p_{nz} being the empirical probability of non-zero values in G.

In contrast, the computational complexity of a matrix multiplication of the form $W \cdot G$ with W being, for instance, an arbitrary $(m \times k)$-dimensional weight matrix, is of order $\mathcal{O}(mkn)$. If we now measure the effective asymptotic savings between the dithered dot product vs the dense dot product algorithm by taking the ratio of both quantities we get

$$\text{comp. savings} = \mathcal{O}\left(\frac{1}{m} + p_{nz}\right) \xrightarrow{m \gg 1} \mathcal{O}(p_{nz}) \tag{12}$$

The above equation 12 states that the asymptotic computational savings depend inversely on the amount of rows m of the output matrix, as well as on the sparsity attained after applying NSD. Since the number of output rows m are most often much bigger than one, the computational savings are dominated solely by the sparsities achieved. Later in the experimental section we show that NSD is able to induce high sparsity ratios (between 75% - 99%) during the entire training procedure, thus in principle being able to achieve significant savings.

Practical savings

Unfortunately, the above analysis does not translate directly to real-world speedups/energy savings mainly due to the challenges that unstructured sparsity imposes on the hardware level. Nevertheless, it is worth to mention that in recent years there has been significant progress in this field, showing promising results in narrowing the gap between the theory and practice. On a software level, [10] have

shown that they can already attain up to x2.4 speedups for DNNs with 80%-90% sparsity, by optimizing the sparse dot products so that it becomes more amenable to the underlying hardware. On the other hand, many hardware accelerators have been proposed [13, 16, 5, 17] that are able to successfully exploit structured and unstructured sparsity, sometimes achieving orders of magnitude more compute efficiency. In particular, [17] attained about x1.5-x8 speedups and x1.5-x6 energy gains at sparsity ratios between 75%-95%, ratios that are typically induced by dithered backprop (see experiments section). Finally, [2] proposed an accelerator that includes an efficient implementation of dithered quantization in order to perform DNN inference with lower bit-precision. Hence, this progress motivates the study of methods akin to dithered backprop, since it seems likely to expect similar gains when such algorithms are implemented in an efficient manner on a software level and run on similarly optimized hardware architectures.

3.5. Quantizing forward pass

So far we have only discussed the reduction of the computational cost of the backprop method. Although the backward pass accounts for roughly 2/3 of the computational complexity of the training iteration (see equation 1 vs equation 2 and equation 3), we are also interested in applying methods that also reduce the computation of the forward pass. Fortunately, some research has already been done in this area.

[3], *e.g.*, quantizes activation, weight and some gradient values in the backward pass to 8-bits and show that using their method state-of-the-art results can still be achieved. In addition, they introduced Range Batch-Normalization (BN), an approximated batch norm that scales a batch by dividing it by its range. It has significantly higher tolerance to quantization noise and improved computational complexity.

Armed with this knowledge, we similarly quantize activation and weight values in the forward pass and apply dithered backprop in the backward pass, leaving also only the weight update in full precision. Therefore, all compuations, except for the weight update, can be calculated with 8-bit computations.

3.6. Usage in distributed training setting

A further interesting area of application of the dithered backprop method is distributed training. In distributed training, an algorithm called *synchronous stochastic gradient descent* (SSGD) is widely used [21]. It differs from single-threaded mini-batch SGD in that the mini-batch of size m is distributed to N total workers that locally compute sub-mini-batch gradients. These gradients are then communicated to a centralized server called *parameter server* that updates the parameter vector and then eventually sends it back. By increasing the number of training nodes and taking advantage of data parallelism, the total computation time of the forward-backward passes on the same size training data can as such be dramatically reduced.

As mentioned in the above section, dithered backprop induces unbiased noise with bounded variance to the weight updates. Therefore, if dithered backprop is applied to N nodes, then most of the induced noise cancels out on the server side due to the averaging effect. Moreover, the variance of the noise goes down with $1/N$. Thus, dithered backprop promises to reduce the computational cost per node as the number of nodes N grows, since stronger quantization can be applied without affecting the final performance of the model after training. This may be beneficial for scenarios where a large number of nodes with limited computational resources may participate in the training procedure, e.g., a large number of mobile devices connected through a communication channel with high bandwidth such as 5G.

4. Experiments

Datasets. We conducted our experiments on four different benchmark datasets for image classification, namely MNIST, CIFAR10, CIFAR100 and ImageNet.

Training Setting. For the MNIST Dataset Lenet300100 and Lenet5 were evaluated, while for CIFAR10 and CIFAR100 it was VGG11, AlexNet and ResNet18 and for ImageNet only ResNet18. For the CIFAR datasets, we reduced the capacity of the models to account for the dataset. That is, for AlexNet we reduced the dimensionality of the last two hidden layers to 2048, and for VGG11 to 512. The last layers are adapted to account for the classes, respectively.

All Models are trained via stochastical gradient descent with a `momentum` of 0.9, a `weight decay` of 5×10^{-4}, and a `batch-size` of 256 for ImageNet and 128 for the others. We used a learning rate `lr` of 0.05 for AlexNet and 0.1 for the rest of the models. For the CIFAR datasets a `lr-decay` setting of 0.1/100 and 0.1/45 is applied.

4.1. Accuracy & Induced Sparsity

For the listed data sets we conducted experiments for four different methods, according to the training setting described above. Besides the baseline method, which describes training without quantization, we applied dithered backprop as described in the above section, the precision quantization of [3] (8-bit training) which applies quantization in the forward and backward pass in order to perform training in 8-bit precision, and the combination of the latter two. Table 1 summaries our findings.

Firstly, notice how the baseline training method exhibits vastly different sparsities across different models, ranging from 2% to 92%. Models trained without batchnorm such as AlexNet exhibit already high sparsity ratios due to the derivative of the ReLU activation function, which is often 0. However, batchnorm layers cancel out this effect and there-

Model	Dataset	Baseline		Dithered Backprop		8-bit Training [3]		8bit + dith. backprop	
		acc%	sparsity%	acc%	sparsity%	acc%	sparsity%	acc%	sparsity%
LeNet5	MNIST	99.31	2.05	**99.35**	**97.52**	99.34	2.09	99.35	97.18
LeNet300100	MNIST	98.45	47.48	98.40	**94.92**	98.43	48.61	**98.52**	94.85
AlexNet	CIFAR10	91.23	91.35	**91.26**	**98.95**	91.03	64.62	90.81	97.05
ResNet18	CIFAR10	**92.67**	24.36	92.35	91.86	92.22	34.88	92.10	**92.10**
VGG11	CIFAR10	92.35	8.47	92.17	94.10	**92.44**	4.82	92.29	**94.24**
AlexNet	CIFAR100	67.98	92.23	67.78	**97.35**	**68.37**	64.39	67.63	89.51
Resnet18	CIFAR100	69.54	18.23	69.97	87.66	**70.73**	13.39	69.69	**87.74**
VGG11	CIFAR100	70.58	6.70	70.09	**91.79**	**71.29**	83.40	70.07	91.77
Resnet18	ImageNet	**71.40**	6.44	71.10	**75.80**	71.25	3.27	71.23	75.48
Average	-	83.72	33.03	83.61	**92.22**	**83.90**	35.50	83.52	91.10
Average diff.	-	0	0	0.23	59.12	0	0	0.40	55.61

Table 1: Results of experiments, where acc% means accuracy in % on test set and sparsity% the average sparsity of the gradients of the preactivation values in % over all layers and training iterations. The largest values for are marked in bold.

fore models such as LeNet5 or VGG11 exhibit high density (low sparsity). We see a similar effect on models trained with 8-bit precision. On average, the baseline backprop method was able to induce only 33% sparsity across the different models, and similarly the 8-bit backprop method only 36%.

In contrast, after applying dithered backprop, sparsity becomes very high across all networks, ranging between 76%-99%. In particular, notice how dithered backprop is able to significantly increase the sparsity of networks trained with batchnorm layers. For instance, LeNet5 goes from 2.05% to 97.52%, a substantial increase of 95.47%. On average, **dithered backprop was able to induce 92% sparsity** across the models, **increasing the sparsity ratio by 59%** from the baseline. We get similar results when applied in combination with the 8-bit training method. Here, dithered backprop increased the sparsity by 56%, inducing an average sparsity of 91% across the networks. If we consider the speedups and energy gains reported in [17], these results may potentially translate to x5 speedups and x4.5 energy gains on average if dithered backprop is run on specialized hardware.

We stress that the accuracies were approximately maintained across the experiments, changing on average only by 0.3% between the dithered and non-dithered methods. Moreover, the number of training epochs did also not change, showing that **dithered backprop did not harm the convergence speed**. Figure 3 shows an example where the test error of AlexNet is plotted over the training epochs. As can be seen, there is no recognizable difference in convergence speed between the baseline model and the dithered model. More examples can be found in the appendix.

Additionally, we also want to mention that the maximum bitwidth of the non-zero values was consistently below/equal to 8-bits (see figure 8) across all experiments.

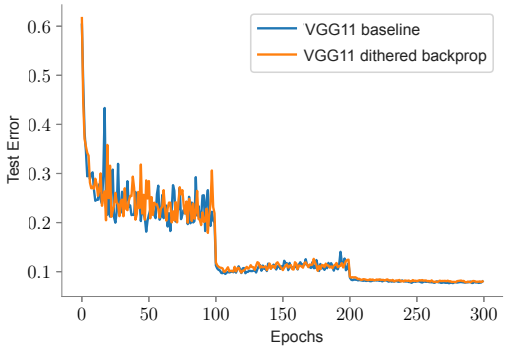

Figure 3: Test error of VGG11 trained on CIFAR10 over the training epochs.

Thus, dithered backprop is fully compatible to training methods that limit the bit-precision training to 8-bits, such as [3].

In Figure 4 we show the course of the density (# non-zero values or 1-sparsity) of the preactivation gradient over the entire training period of the VGG11 model. We can see how the density of the gradients is much lower when dithered backprop is applied across the entire training procedure. Interestingly, we also see that the density decreases at the beginning of training and stays approximately constant afterwards. Coincidentally, this trend correlates weakly with the speed of the learning progress, which can be interpreted as gradients carrying more information. However, it seems that dithered backprop is successful at eliminating redundant, non-useful information for learning since its density is much lower.

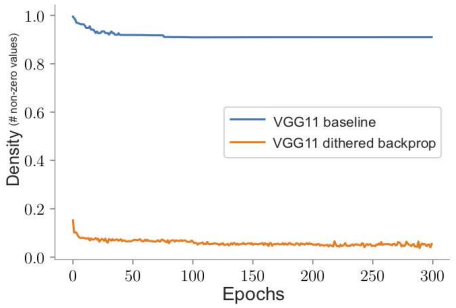

Figure 4: Average density (# non-zero values or 1-sparsity) of the preactivation gradients during training.

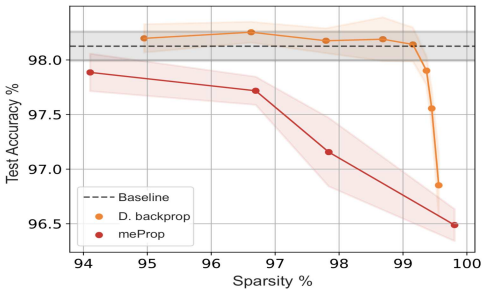

Figure 5: Learning performance at different levels of average sparsity the preactivation gradients of a multilayer perceptron with two hidden layers (500, 500) trained on MNIST, using either regular back propagation (Baseline), dithered backprop (D. backprop) or meProp [24]. Multiple runs with different random seeds were executed for each configuration. Points show mean performance with standard deviation indicated as span.

4.2. Comparison to meProp

We now benchmark dithered backprop against meProp [24], arguable the closest related work. To recall, in one of its modes meProp sparsifies the pre-activation gradients by selecting the k elements with the largest magnitude. This induces biased estimates of the weight updates, which we argue affects negatively the learning quality of the network.

Since meProp was only benchmarked on multilayer perceptrons, we chose a model with two fully-connected layers with hidden dimensions of (500, 500) and trained it on MNIST and CIFAR10 for the experiments. Figure 5 shows the final test accuracy of the model trained on MNIST at different levels of average sparsity of the preacitvation gradients. On the appendix we show the results for CIFAR10. As one can see, dithered backprop clearly outperforms meProp at all levels of sparsity. Concretely, overall dithered backprop achieved an average test accuracy of 98.14% at a sparsity of 99.15%, whereas meProp only achieved 97.89% average test accuracy at a sparsity of 94.11%.

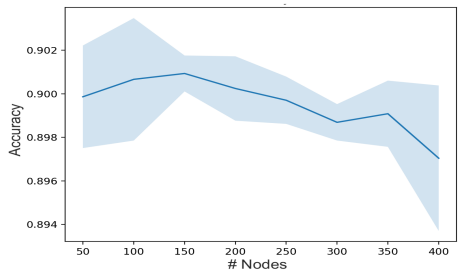

Figure 6: Accuracy of the final model of AlexNet trained on CIFAR10 with dithered backprop in a distributed training setting, at different number of participating nodes configuration. The accuracy stays approximately constant as the number of nodes increases.

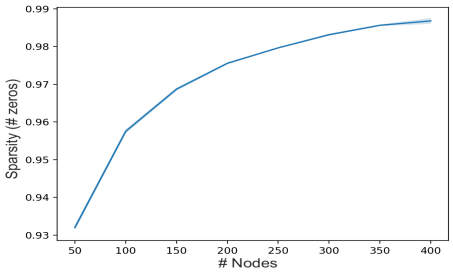

Figure 7: Average sparsity of the preactivation gradients of the fully-connected layers of AlexNet trained on CIFAR10 with dithered backprop in a distributed training setting, at different number of participating nodes configuration. As the number of nodes increases, so does the sparsity at each node and therefore its computational savings for training.

4.3. Distributed training

In the above section we argued that applying dithered backprop in a distributed training scenario may be beneficial. The rationale was that, since the noise induced by dithered backprop on the weight updates is unbiased with bounded variance, then this should cancel out as the number of nodes grows due to the averaging of the parameters on the server. In this section we try to show this effect experimentally.

To investigate this, we ran several experiments of the same model with different amount of nodes N. While increasing N, we also increase the scaling factor s of the dither method in order to increase the quantization strength. At each training iteration, each node runs one forward and dithered backward pass of batchsize 1, then sends its parameter gradients to the server where it is subsequently averaged with the gradients of all other nodes. Finally the averaged parameter gradient are broadcasted back to each node, and a new train-

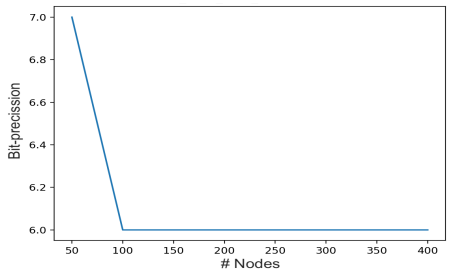

Figure 8: Maximal, worst-case bit-precision of the fully-connected layers of AlexNet trained on CIFAR10 with dithered backprop in a distributed training setting, at different number of participating nodes configuration. As the number of nodes increases, the number of bits necessary to represent the non-zero values decreases, and with it the computational cost for training at each node.

ing iteration subsequently starts again. We then measure the final accuracy of the model, average sparsity and worst-case bit-precision at all N configurations.

Figures 6, 7 and 8 show the respective trends for the fully-connected layers of AlexNet trained on CIFAR10. On the appendix we show the same plots for the convolutional layers as well. Each plot shows the average trend and the standard deviation over 3 different runs of the same experiments. As one can see, we can increase the sparsity and lower the bit-precision as the number of participating nodes N increases, while negligibly affecting the final accuracy of the model. In other words, **dithered backprop allows to to reduce the computational cost of performing a training iteration at each node as the number of participant nodes increases**.

As a side note, we want to remark that in the generall case high sparsities on the preactivation gradients do not necessarily translate to communication savings. For batch-sizes bigger than one the weight updates are with high probability densely populated, so that the full model would have to be communicated at each iteration. Only when the batch-size per node is equal to one (as was in the case of our experimental setup), sparsities on the preactivation gradients directly translate to sparsity on the weight updates and consequently to savings in communication cost.

5. Conclusion

In this work we proposed a method for reducing the computational complexity of the backpropagation (backprop) algorithm. Our method, called *dithered backprop*, is based on applying dithered quantization on the tensor of the preactivation gradients in order to induce sparsity and non-zero values with low bit-precision. It is also simple in that it has only one global hyperparameter which controls the trade-off between computational complexity and learning performance of the model.

Extensive experimental results show that dithered backprop is able to attain high sparsity ratios, between 75%-99% across a wide set of neural network models, boosting the sparsity by 59% on average from the original backprop method. In addition, we showed that dithered backprop maintains the bit-precision of the non-zero values to less/equal 8-bits during the entire training process, thus being fully compatible with methods that limit the training to 8-bit precision only. However, in its current form, dithered backprop induces unstructured sparsity which is not amenable to conventional hardware such as CPUs or GPUs. In future work we will consider modifications that translate directly to speedups and energy gains without having to rely on specialized hardware. Moreover, we will also consider applying efficient compression algorithms to the gradients in order to reduce memory complexity of training as well [29, 27].

We also showed that beneficial properties emerge when dithered backprop is applied in the context of distributed training. For instance, we showed experimentally that as the number of participating nodes increases, so does the computational savings per node as well. This effect can be advantageous when a large number of nodes with resource-constrained computational engines participate in the training procedure, such as mobile phones. A further interesting future work direction is to apply dithered backprop jointly with methods that drastically reduce the communication cost [19, 20], with the goal of minimizing both the communication as well as computation cost of the distributed training system.

Acknowledgements

This work was partly supported by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref 01IS18037A)

References

[1] Menachem Adelman and Mark Silberstein. Faster neural network training with approximate tensor operations, 2018. 2

[2] K. Ando, K. Ueyoshi, Y. Oba, K. Hirose, R. Uematsu, T. Kudo, M. Ikebe, T. Asai, S. Takamaeda-Yamazaki, and M. Motomura. Dither nn: An accurate neural network with dithering for low bit-precision hardware. In *2018 International Conference on Field-Programmable Technology (FPT)*, pages 6–13, Dec 2018. 5

[3] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural In-*

formation Processing Systems 31, pages 5145–5153. Curran Associates, Inc., 2018. 2, 5, 6

[4] Léon Bottou. Online learning and stochastic approximations, 1998. 4

[5] Y. Chen, T. Yang, J. Emer, and V. Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, June 2019. 5

[6] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, Jan 2018. 2

[7] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, volume 28, pages 3123–3131. Curran Associates, Inc. 2

[8] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications, 2014. 2

[9] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1, 2016. 2

[10] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets, 2019. 4

[11] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep Learning*, volume 1. MIT press Cambridge, 2016. 1

[12] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning (ICML)*, volume 37, pages 1737–1746, 2015. 2

[13] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: Efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254, 2016. 5

[14] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal on Machine Learning Research*, 18(1):6869–6898, 2017. 2

[15] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *International Conference on Learning Representations (ICML)*, 2018. 2

[16] Bert Moons, Daniel Bankman, and Marian Verhelst. *Embedded Deep Learning: Algorithms, Architectures and Circuits for Always-on Neural Network Processing*. Springer Publishing Company, Incorporated, 1st edition, 2018. 5

[17] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 27–40, June 2017. 5, 6

[18] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986. 1

[19] F. Sattler, S. Wiedemann, K. Müller, and W. Samek. Robust and communication-efficient federated learning from non-i.i.d. data. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2019. 8

[20] F. Sattler, S. Wiedemann, K. Müller, and W. Samek. Sparse binary compression: Towards distributed deep learning with minimal communication. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019. 8

[21] Felix Sattler, Thomas Wiegand, and Wojciech Samek. Trends and advancements in deep neural network communication. *arXiv preprint arXiv:2003.03320*, 2020. 5

[22] Leonard Schuchman. Dither signals and their effect on quantization noise. 12(4):162–165. 1, 3

[23] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *CoRR*, abs/1906.02243, 2019. 1

[24] Xu Sun, Xuancheng Ren, Shuming Ma, and Houfeng Wang. meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting, 2017. 2, 7

[25] V. Sze, Y. Chen, T. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, Dec 2017. 2

[26] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems 30*, pages 1509–1519. Curran Associates, Inc., 2017. 2

[27] S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinc, D. Neumann, T. Nguyen, H. Schwarz, T. Wiegand, D. Marpe, and W. Samek. Deepcabac: A universal compression algorithm for deep neural networks. *IEEE Journal of Selected Topics in Signal Processing*, pages 1–1, 2020. 8

[28] S. Wiedemann, A. Marban, K. Müller, and W. Samek. Entropy-constrained training of deep neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019. 2

[29] S. Wiedemann, K. Müller, and W. Samek. Compact and computationally efficient representation of deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(3):772–785, 2020. 8

[30] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients, 2016. 2