Adaptive Precision Training (AdaPT):
A dynamic quantized training approach for DNNs

Lorenz Kummer[12]   Kevin Sidak [13]   Tabea Reichmann [14]   Wilfried Gansterer [15]

August 2021

Abstract

Quantization is a technique for reducing deep neural networks (DNNs) training and inference times, which is crucial for training in resource constrained environments or applications where inference is time critical. State-of-the-art (SOTA) quantization approaches focus on *post-training* quantization, i.e., quantization of pre-trained DNNs for speeding up inference. While work on quantized training exists, most approaches require refinement in full precision (usually single precision) in the final training phase or enforce a global word length across the entire DNN. This leads to suboptimal assignments of bit-widths to layers and, consequently, suboptimal resource usage. In an attempt to overcome such limitations, we introduce AdaPT, a new fixed-point quantized sparsifying training strategy. AdaPT decides about precision switches between training epochs based on information theoretic conditions. The goal is to determine on a *per-layer basis* the lowest precision that causes no quantization-induced information loss while keeping the precision high enough such that future learning steps do not suffer from vanishing gradients. The benefits of the resulting fully quantized DNN are evaluated based on an analytical performance model which we develop. We illustrate that an average speedup of 1.27 compared to stan-

dard training in float32 with an average accuracy increase of 0.98% can be achieved for AlexNet/ResNet on CIFAR10/100 and we further demonstrate these AdaPT trained models achieve an average inference speedup of 2.33 with a model size reduction of 0.52.

1   Introduction

With the general trend in machine learning towards large model sizes to solve increasingly complex problems, inference in time critical applications or training under resource and/or productivity constraints is becoming more and more challenging. Applications, where time- and space efficient models are crucial, include robotics, augmented reality, self driving vehicles, mobile applications, applications running on consumer hardware, or scientific research where a high number of trained models is often needed for hyper parameter optimization. Accompanied by this, already some of the most common DNN architectures, like AlexNet [1] or ResNet [2], suffer from over-parameterization and overfitting [3, 4]

Possible solutions to the aforementioned problems include pruning (see, for example, [5, 6, 7, 8, 9]), or quantization. While network pruning, which aims at reducing the number of parameters in a given DNN architecture (e.g. by sparsification of weights tensors and using a sparse tensor format), is a successful strategy for reducing network size and runtime, it generally does not attempt to tailor parameter bit-width to exploit advantages available in low bit-width arithmetic, thus neglecting this potential to reduce

[1]Faculty of Computer Science, University of Vienna
[2]lorenz.kummer@univie.ac.at
[3]kevin.sidak@univie.ac.at
[4]tabea.reichmann@univie.ac.at
[5]wilfried.gansterer@univie.ac.at

computational resource consumption. We use pruning in the form of sparsification but our focus lies on making better use of quantization. When quantizing, the precision of the parameters and of the computation is decreased and the resultingmore coarse bit-width allows for more efficient use of computing resources and for runtime reductions. However, quantization has to be performed with caution, as naive approaches or a too low bit-width can have a negative impact on the accuracy of the network and its ability to converge during training, which is unacceptable for important use cases. For example, binary quantization as proposed in [10] reduces memory requirements and can effectively speed up computation, as multiplication can be performed as bit shifts, but the accuracy suffers slightly from this approach and convergence is shown to much slower compared to float32 training.

Existing quantization approaches do not fully leverage the potential of quantized training. They do not take the differences quantization can have on different layers during training into account, nor dynamically raise or lower the precision used. AdaPT extends these approaches and introduces a new precision switching mechanism based on the Kullback-Leibler-Divergence (KLD) [11], that calculates the average number of bits lost due to a precision switch. This is done not only for the entire network, but on a per-layer basis, therefore considering different quantization effects on the different layers, leading to **Ada**ptiv**E P**recision **T**raining of DNNs over time during training. Our approach does not need any refinement phase in full precision and produces an already fixed-point quantized network that can then also be deployed on high-performance application-specific integrated circuits (ASICs) or or field-programmable gate arrays (FPGAs) hardware.

1.1 Related Work

In general studies of the sensitivity of neural networks (NNs), perturbation analysis has historically been applied with a focus on the theoretical sensitivity of single neurons or multi-layer perceptrons [12, 13, 14], but recently also yielded interesting results for the sensitivity of simple modern architectures by provid-

ing efficient algorithms for evaluating networks comparable to the complexity of LeNet-5 [15, 16]. While these studies certainly improve our understanding of the effect perturbation has on NNs, they lack practical applicability for quantized DNN training due to beeing largely analytical in nature.

For exploring the accuracy degradation induced by quantizations of weights, activation functions and gradients, [17] and [18] introduced the frameworks *TensorQuant* and *QPyTorch*, capable of simulating the most common quantizations for training and inference tasks on a float32 basis. Both frameworks allow to freely choose exponent and mantissa for floating-point representation, word and fractional bit length for fixed-point representation and word length for block-floating-point representation as well as signed/unsigned representations. Since quantization is only simulated in these frameworks, no runtime speedup can be achieved on this basis.

Several approaches have tried to minimize inference accuracy degradation induced by quantizing weights and/or activation functions while leveraging associated performance increases. *Quantization Aware Training* (QAT) [19, 20] incorporates simulated quantization into model training and trains the model itself to compensate the introduced errors. A similar approach is take by *Uniform Noise Injection for Non-Uniform Quantization* (UNIQ) [21], which emulates a non-uniform quantizer to inject noise at training time to obtain a model suitable for quantized inference. *Learned Quantization Nets* (LQ-Nets) [22] learn optimal quantization schemes through jointly training DNNs and associated quantizers. The *Reinforcement-Learning Framework* (ReLeQ) introduced by [23] uses a reinforcement learning agent to learn the final classification accuracy w.r.t. the bit-width of each of the DNNs layers to find optimal layer to bit-width assignments for inference. Dedicated quantization friendly operations are used in [24]. A different approach is taken by *Variational Network Quantization* (VNQ) [25] which uses variational dropout training [26] with a structured sparsity inducing prior [27] to formulate post-training quantization as the variational inference problem searching the posterior optimizing the KLD. *High-order entropy minimization for neural network compression*

(HEMP) [28] introduces a entropy coding-based regularizer minimizing the quantized parameters entropy in gradient based learning and pairs it with a separate pruning scheme [29] to reach a high degree of model compression after training.

Machine learning frameworks such as PyTorch or Tensorflow already provide built-in quantization capabilities. These quantization methods focus on either QAT or on post-training quantization [30, 31]. Both of these methods only quantize the model after the training and consequently provide no efficiency gain during training. Additional processing steps during or after training are needed which add computational overhead.

From a theoretical perspective, quantized training has been investigated in [32] with a particular focus on rounding methods and on convergence guarantees while [33] provided an analysis of the behaviour of the straight-through estimator (STE) [34] during quantized training. For distributed training on multi-node environments, *Quantized Stochastic Gradient Descent* (QSGD) [35] incorporates a family of gradient compression schemes aimed at reducing inter-node communication occurring during SGD's gradient updates. This produces a significant reduction in runtime and network training can be guaranteed to converge under standard assumptions [36]. Very low bit width single-node training (binary, ternary or similarly quantized weights and/or activations), usually in combination with STE, has been shown to yield non-trivial reductions in runtime and model size as well but often either at comparably high costs to model accuracy, decreased convergence speed or not fully quantizing the network [10, 37, 38, 39, 40, 41] and its most capable representative, *Quantized tensor train neural networks* (QTTNet) [42], which combines tensor decomposition and full low bit-width quantization during training to reduce runtime and memory requirements significantly, still suffers up to 2.5% accuracy degradation compared to it's respective baseline. A low bit-width integer quantized (re-) training approach reporting accuracy drops of 1% or less compared to a float32 baseline was introduced by [43], but it requires networks pre-trained in full precision as starting points and it's explicitly stated goal is reducing computational cost and model size

during inference. For speeding up training on a single node via block-floating point quantization, [44] introduced a dynamic training quantization scheme (*Multi Precision Policy Enforced Training*, MuPPET). After quantized training it produces a model for float32 inference.

1.2 Contributions

Existing quantized training and inference solutions leave room for improvements in several areas. QAT, LQ-Nets, ReLeQ, UNIQ, HEMP and VNQ are capable of producing networks such that inference can be performed under quantization with small accuracy degradation (relative to baselines, which are not comparable and where it is unclear how well they are optimized), but require computationally expensive float32 training and the algorithms themselves incur a certain overhead as well. QSGD training only quantizes gradients and focuses on reducing communication overhead in multi-node environments. Binary, ternary or similarly quantized training has been shown to commonly come at the cost of reduced accuracy, reduced convergence speed or only quantizing weights and not activations and its most capable representative QTTNet still does not achieve iso-accuracy.

MuPPET requires at least N epochs for N quantization levels (pre-defined bit-widths applied globally to all layers) because precision switches are only triggered at the end of an epoch and the algorithm needs to go through all precision levels. Potential advantages from having different precision levels at different layers of the network are not exploited. The precision switching criterion is only based on the diversity of the gradients of the last k epochs, no metric is used to measure the amount of information lost by applying a certain quantization to the weights. Precision levels can only increase during training and never decrease. Furthermore, MuPPET outputs a float32 network s.t. inference hast to be done in expensive full precision.

We advance the SOTA by providing an easy-to-use solution for quantized training of DNNs by using an information-theoretical intra-epoch precision switching mechanism capable of dynamically increasing and

3

decreasing the precision of the network on a per-layer basis. Weights and activations are quantized to the lowest bit-width possible without information loss according to our heuristic and a certain degree of sparsity is induced while at the same time the bit-width is kept high enough for further learning steps to succeed. This results in a network which has advantages in terms of model size and time for training and inference. By training AlexNet and ResNet20 on the CIFAR10/100 datasets, we demonstrate on the basis of an analytical model for the computational cost that in comparison to a float32-baseline AdaPT is competitive in terms of accuracy and produces a non-trivial reduction of computational costs (speedup). Compared to MuPPET, AdaPT also has certain intrinsic methodological advantages. After AdaPT training, the model is fully quantized and sparsified to a certain degree s.t., unlike the case with MuPPET, which outputs a float32 model, AdaPT carries over it's advantages to the inference phase as well.

2   Background

2.1   Quantization

Numerical representation describes how numbers are stored in memory (illustrated by fig. 1) and how arithmetic operations on those numbers are conducted. Commonly available on consumer hardware are floating-point and integer representations while fixed-point or block-floating-point representations are used in high-performance ASICs or FPGAs. The numerical precision used by a given numerical representation refers to the amounts of bits allocated for the representation of a single number, e.g. a real number stored in float32 refers to floating-point representation in 32-bit precision. With these definitions of numerical representation and precision in mind, most generally speaking, quantization is the concept of running a computation or parts of a computation at reduced numerical precision or a different numerical representation with the intent of reducing computational costs and memory consumption. Quantized execution of a computation however can lead to the introduction of an error either through the quantized

representations the machine epsilon ϵ_{mach} being too large (underflow) to accurately depict resulting real values or the representable range being too small to store the result (overflow).

Floating-Point Quantization The value v of a floating point number is given by $v = \frac{s}{b^{p-1}} \times b^e$ where s is the significand (mantissa), p is the precision (number of digits in s), b is the base and e is the exponent [45]. Hence quantization using floating-point representation can be achieved by reducing the number of bits available for mantissa and exponent, e.g. switching from a float32 to a float16 representation, and is offered out of the box by common machine learning frameworks for post-training quantization[46, 47].

Integer Quantization Integer representation is available for post-training quantization and QAT (int8, int16 due to availability on consumer hardware) in common machine learning frameworks [46, 47]. Quantized training however is not supported due to integer quantized activations being not meaningfully differentiable, making standard backpropagation inapplicable [33]. Special cases of integer quantization are 1-bit and 2-bit quantization, which are often referred to as binary and ternary quantization in literature.

Block-Floating-Point   Quantization Block-floating-point represents each number as a pair of WL (word length) bit signed integer x and a scale factor s s.t. the value v is represented as $v = x \times b^{-s}$ with base $b = 2$ or $b = 10$. The scaling factor s is shared across multiple variables (blocks), hence the name block-floating point, and is typically determined s.t. the modulus of the larges element is $\in [\frac{1}{b}, 1]$ [48]. Block-floating-point arithmetic is used in cases where variables cannot be expressed with sufficient accuracy on native fixed-point hardware.

Fixed-Point Quantization Fixed-point numbers have a fixed number of decimal digits assigned and hence every computation must be framed s.t. the
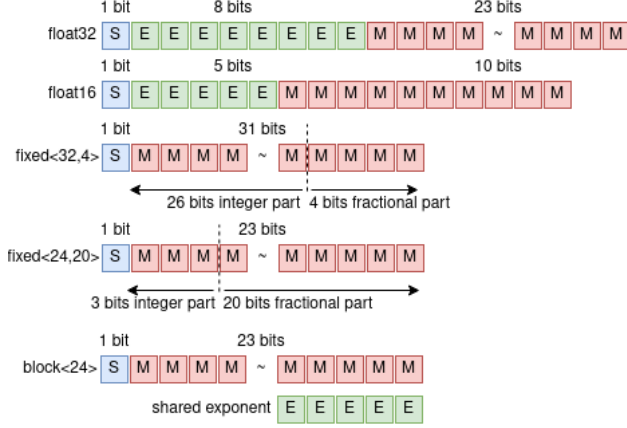
Figure 1: *Examples of floating-point, fixed-point and block-floating-point representations at different numerical precisions. Sign (S), Exponent (E), Mantissa (M) bits.*

results lies withing the given boundaries of the representation [49]. By definition of [50], a signed fixed-point numbers of world length $WL = i + s + 1$ can be represented by a 3-tuple $\langle s, i, p \rangle$ where s denotes whether the number is signed, i denotes the number of integer bits and p denotes the number of fractional bits.

2.2 MuPPET

MuPPET is a mixed-precision DNN training algorithm that combines the use of block-floating and floating-point representations. The algorithm stores two copies of the networks weights: a float32 master copy of the weights that is updated during backwards passes and a quantized copy used for forward passes. MuPPETs uses block-floating point representation as outlined in sec. 2.1 with base $b = 2$ for quantization. The precision level i of a layer l of all layers $\mathbb{L}$ is defined as

$$q_l^i = \left\langle WL^{net}, s_l^{weights}, s_l^{act} \right\rangle^i$$

with WL^{net} being global across the network, and scaling factor s (for weights and activation functions) varying per layer, determined each time precision

switch is triggered. The scaling factor for a matrix X is given by

$$s^{weights,act}$$
$$= \left\lfloor log_2 min \left(\left(\frac{UB + 0.5}{\mathbf{X}_{max}^{\{weights,act\}}}, \frac{LB - 0.5}{\mathbf{X}_{min}^{\{weights,act\}}} \right) \right) \right\rfloor$$

with $\mathbf{X}_{max,min}^{\{weights,act\}}$ describing the maximum or minimum value in the weights or feature maps matrix of the layer. UB and LB describe the upper bounds and lower bounds of the of the word length WL^{net}. Its individual elements x are quantized:

$$x_{quant}^{\{weights,act\}} = \lfloor x^{\{weights,act\}} * 2^{s^{\{weights,act\}}}$$
$$+ \text{Unif}(-0.5, 0.5) \rceil$$

$Unif(a, b)$ is the sampling from a uniform distribution in the interval $[a, b]$. The parameters 0.5 and -0.5 to add to the bounds is chosen for maximum utilisation of WL^{net}. During training the precision of the quantized weights is increased using a precision-switching heuristic based on gradient diversity [51], where the gradient of the last mini-batch $\nabla f_l^j(\boldsymbol{w})$ of each layer $l \in \mathbb{L}$ at epoch j is stored and after a certain number of epochs (r), the inter-epoch gradient diversity Δs at epoch j is computed by:

$$\Delta s(\boldsymbol{w})^j = \frac{\sum_{\forall l \in \mathbb{L}} \frac{\sum_{k=j-r}^{j} \|\nabla f_l^k(\boldsymbol{w})\|_2^2}{\|\sum_{k=j-r}^{j} \nabla f_l^k(\boldsymbol{w})\|_2^2}}{\mathbb{L}}$$

At epoch j there exists a set of gradient diversities $S(j) = \{\Delta s(\boldsymbol{w})^i \forall e \leq i < j\}$ (e denotes the epoch in which it was switched into the quantization scheme) of which the ratio $p = \frac{\max S(j)}{\Delta s(\boldsymbol{w})^i}$ is calculated. If p violates a threshold for a certain number of times, a precision switch is triggered. Speedups claimed by MuPPET are based on an estimated performance model simulating fixed-point arithmetic using NVIDIA CUTLASS [52] for compatibility with GPUs, only supporting floating and integer arithmetic.

We chose MuPPET as baseline to compare AdaPT against because of the examined related work for the quantized training task, MuPPET comes closest to the goal of iso-accuracy.

3 AdaPT

3.1 Quantization Friendly Initialization

Despite exhaustive literature research and to our best knowledge, it has not yet been explored how weights initialization as counter-strategy for vanishing/exploding gradients impacts quantized training. However the preliminary experimental results of our first investigation of the impact of different weights initialization strategies showed the resilience of DNNs trained under a fixed forwards pass integer quantization scheme (int2, int4, int8, int16) with float32 master copies for gradient computations correlates strongly with initializer choice (fig. 2). Using Adam [53] as optimizer, we trained LeNet-5 on MNIST/FMNIST [54] and AlexNet on CIFAR10/100 [55] and examined the degree to which the quantized networks are inferior in accuracy compared to baseline networks trained in float32 dependent on initializer choice (Random Normal, Truncated Normal, Random Uniform, Glorot Normal/Uniform [56], He Normal/Uniform [57], Variance Scaling [58], Lecun Normal/Uniform [59, 60]) and initializer parameters. We found that DNNs initialized by fan-in truncated normal variance scaling (TNVS) degrade least under quantized training with a fixed integer quantization scheme as described above. This correlates with results published by [61] who introduced a learnable scale parameters in their asymmetric quantization scheme to achieve more stable training. We thus initialize networks with TNVS and an empirically chosen scaling factor s where n^l is the number of input units of a weights tensor $\boldsymbol{W}^l$ before quantized training with AdaPT.

$$\boldsymbol{W}^l \sim N\left(\sigma = \sqrt{\frac{s}{n^l}}, \mu = 0, \alpha = \pm\sqrt{\frac{3 \cdot s}{n^l}}\right)$$

The examination of how other counter strategies (eg. gradient amplification, [62], gradient normalization [63], weights normalization [64]) affect quantized training remains an open question.
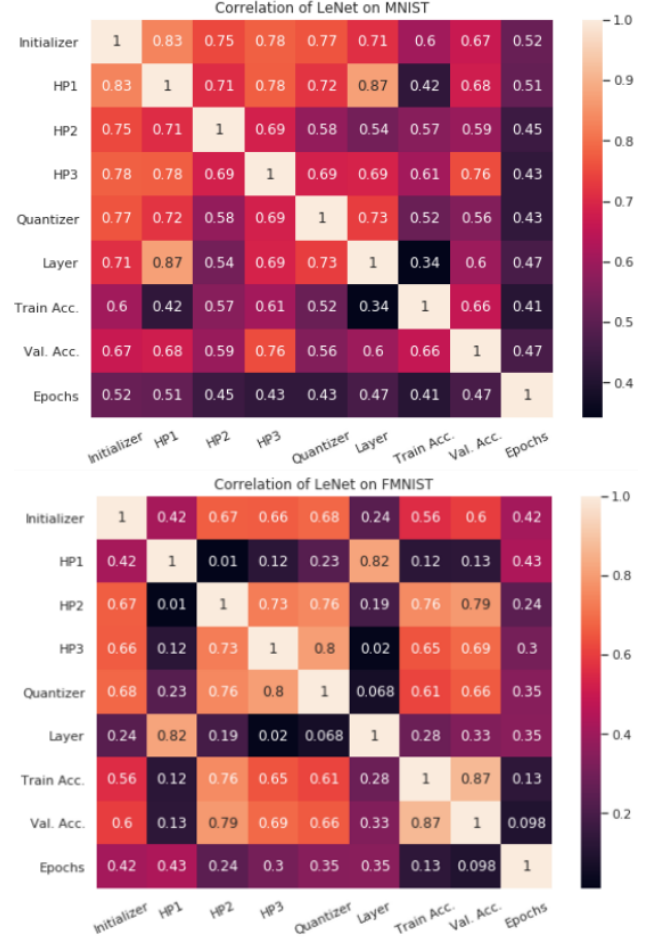

Figure 2: *Correlation of initializer choice, hyperparameters (HP), quantizer, layer, training and validation accuracy, duration of training (epochs) for LeNet5 on MNIST and FMNIST datasets*

3.2 Precision Levels

AdaPT uses fixed-point quantization as defined in sec. 2.1 because unlike block floating point as used by MuPPET with a global word length across the whole network and per-layer shared exponent, we conjecture that optimal word and fractional lengths are local properties of each layer under the hypothesis that different layers contain different amounts of information during different points of time in training. We

decided against using floating-point quantization because fixed-point gives us better control over numerical precision and is available in high performance ASICs which are our target platform and against integer quantization, particularly its extreme forms binarization and ternarization, because our goal is iso-accuracy which literature shows is difficult to reach with such coarse quantization. In principle however, the AdaPT concept could be extended to representations other than fixed-point. Furthermore, AdaPT employs stochastic rounding, which in combination with a fixed-point representation has been shown to consistently outperform nearest-rounding by [50], for quantizing float32 numbers. Given that AdaPT is agnostic towards whether a number is signed or not, we represent the precision level of each $l \in \mathbb{L}$ simply as $\langle WL^l, FL^l \rangle$ whereby fractional length FL^l denotes the number of fractional bits. For a random number $P \in [0, 1]$, x is stochastically rounded by

$$SR(x) = \begin{cases} \lfloor x \rfloor, \text{if } P \geq \frac{x - \lfloor x \rfloor}{\epsilon_{mach}} \\ \lfloor x \rfloor + 1, \text{if } P < \frac{x - \lfloor x \rfloor}{\epsilon_{mach}} \end{cases}$$

3.3 Precision Switching Mechanism

Precision switching in quantized DNN training is the task of carefully balancing the need to keep precision as low as possible in order to improve runtime and model size, yet still keep enough precision for the network to keep learning. In AdaPT, we have encoded these opposing interests in two operations, the PushDown Operation and the PushUp Operation.

The PushDown Operation Determining the amount of information lost if the precision of the fixed-point representation of a layer's weight tensor is lowered, can be heuristically accomplished by interpreting the precision switch as a change of encoding. Assume a weights tensor $\boldsymbol{W}^l \sim Q^l$ of layer $l \in \mathbb{L}$ with its quantized counterpart $\widehat{\boldsymbol{W}}^l \sim P^l$, where P^l, Q^l are the respective distributions. Then the continuous Kullback-Leibler-Divergence [11] (1) represents the average number of bits lost through changing the encoding of l from Q^l to P^l, with p and q denoting probabilities, and w the elements of the weights tensor:

$$D(P^l \| Q^l) = \int_{-\infty}^{\infty} p^l(w) \cdot log \frac{p^l(w)}{q^l(w)} dw$$

Using discretization via binning, we obtain P^l and Q^l at resolution r^l through the empirical distribution function:

$$\widehat{F_{r^l}}(w) = \frac{1}{r^l + 1} \sum_{i=1}^{r^l} 1_{W_i^l \leq w} \qquad (1)$$

that can then be used in the discrete Kullback-Leibler-Divergence (2).

$$KL(P^l \| Q^l) = \sum_{w \in \boldsymbol{W}^l} P^l(w) \cdot log \frac{P^l(w)}{Q^l(w)} \qquad (2)$$

Using a bisection approach, AdaPT efficiently finds the smallest quantization $\langle WL_{min}^l, FL_{min}^l \rangle$ of $\boldsymbol{W}^l$ s.t. $KL(P^l \| Q^l) = 0 \; \forall l \in \mathbb{L}$.

The PushUp Operation However, determining the precision of the fixed-point representation of $\boldsymbol{W}^l$ at batch j, s.t. the information lost through quantization is minimal but there is still sufficient precision for subsequent batches $j + 1$ to keep learning, is a non-trivial task. Solely quantizing $\boldsymbol{W}^l$ at the beginning of the training to a low precision fixed-point representation (e.g. $\langle WL^l, FL^l \rangle = \langle 8, 4 \rangle$) would result in the network failing to learn, because at such low precision levels, gradients would vanish very early on in the backwards pass. Hence, AdaPT tracks for each layer a second heuristic over the last j batches to determine how much precision is required for the network to keep learning. If gradients are quantized, a gradient diversity based heuristic is employed (3), (4).

$$\Delta s(\boldsymbol{w})_j^l = \frac{\sum_{k=j-r}^{j} \|\nabla f_k^l(\boldsymbol{w})\|_2}{\|\sum_{k=j-r}^{j} \nabla f_k^l(\boldsymbol{w})\|_2} \qquad (3)$$

$$\Delta \tilde{s}(\boldsymbol{w})_j^l = \begin{cases} log \Delta s(\boldsymbol{w})_j^l & \text{if } 0 < \Delta s(\boldsymbol{w})_j^l < \infty \\ 1 & \text{otherwise} \end{cases}$$

If $\Delta \tilde{s}(\boldsymbol{w})_j^l > 0$, two suggestions for an increase in precision are computed, $s_1^l = max(\lceil \frac{1}{log \Delta s(\boldsymbol{w})_j^l - 1} \rceil, 1)$ and

$s_2^l = max(min(32 \cdot log^2 \Delta s(\boldsymbol{w})_j^l - 1, 32) - FL_{min}^l, 1)$ and the final suggestion is computed dependent on a global strategy st via

$$s^l = \begin{cases} min(s_1^l, s_2^l) & \text{if st = min} \\ \lceil 0.5 \cdot (s_1^l + s_2^l) \rceil & \text{if st = mean} \\ max(s_1^l, s_2^l) & \text{if st = max} \end{cases} \quad (4)$$

Otherwise, i.e. $\Delta \tilde{s}(\boldsymbol{w})_j^l > 0$, $s^l = 1$. The new fixed-point quantization of layer l is then obtained by $FL^l = (min(FL_{min}^l + s^l, 32), WL^l = min(max(WL_{min}^l, FL_{min}^l) + 1, 32))$.

Dealing with Fixed-Points Limited Range As outlined in sec. 2.1, fixed-point computations must be framed s.t. results fit within the given boundaries. We approach this by adding a number of buffer bits $buff$ to every layers word-length, i.e. at the end of PushUp, $FL^l = (min(FL_{min}^l, 32 - buff), WL^l = max(min(FL_{min}^l + buff, 32), WL_{min}^l)$. Additionally, we normalize gradients to limit weight growth and reduce chances of weights weights becoming unrepresentable in the given precision after an update step.

$$\nabla f^l(\boldsymbol{w}) = \frac{\nabla f^l(\boldsymbol{w})}{\|\nabla f^l(\boldsymbol{w})\|_2}$$

Strategy, Resolution and Lookback For adapting the strategy st mentioned in (4), we employ a simple loss-based heuristic. First we compute the average lookback over all layers $lb_{avg} = |\mathbb{L}|^{-1}\sum_{i=0}^{|\mathbb{L}|} lb^l$ and average loss $\mathcal{L}_{avg} = |\mathcal{L}|^{-1}\sum_{i=0}^{lb_{avg}} \mathcal{L}_i$ over the last lb_{avg} batches. Then via (5), the strategy is adapted.

$$st = \begin{cases} max & \text{if } |\mathcal{L}_{avg}| \leq |\mathcal{L}_i| \text{ and st = 'mean'} \\ mean & \text{if } |\mathcal{L}_{avg}| \leq |\mathcal{L}_i| \text{ and st = 'min'} \\ min & \text{if } |\mathcal{L}_{avg}| > |\mathcal{L}_j| \end{cases}$$

Because the number of gradients collected for each layer affects the result of the gradient diversity based heuristic (3), (4), we introduce a parameter lookback lb^l bounded by hyperparameters $lb_{lwr} \leq lb^l \leq lb_{upr}$

which is estimated at runtime. First, lb_{new} is computed:

lb_{new}^l
$$= \begin{cases} min(max(\lceil \frac{lb_{upr}}{\Delta s(\boldsymbol{w})_j^l} \rceil, lb_{lwr}), lb_{upr}) & \text{if } 0 < \Delta s(\boldsymbol{w})_j^l \\ lb_{upr} & \text{otherwise} \end{cases}$$

Then, to prevent jitter, a simple momentum is applied to obtain the updated $lb^l = \lceil lb_{new}^l \cdot \gamma + (1-\gamma) \cdot lb^l \rceil$ with $\gamma \in [0,1]$.
Similarly, the number of bins used in the discretization step (1) affects the result of the discrete Kullback-Leibler-Divergence (2). We control the number of bins via a parameter referred to as resolution r^l, which is derived at runtime and bounded by hyperparameters $r_{lwr} \leq r^l \leq r_{upr}$.

$$r^l = \begin{cases} min(max(r^l + 1, r_{lwr}), r_{upr}) & \text{if } lb^l = lb_{upr} \\ min(max(r^l - 1, r_{lwr}), r_{upr}) & \text{if } lb^l = lb_{lwr} \end{cases}$$
$$(5)$$

3.4 AdaPT-SGD (ASGD)

Although AdaPT can in principle be combined with any iterative gradient based optimizer (e.g. Adam), we chose to implement AdaPT with Stochastic Gradient Descent (SGD), because it generalizes better than adaptive gradient algorithms [65]. The implementation of AdaPT employs the precision switching mechanism and numeric representation described in sec. 3, by splitting it in two operations: the PushDown Operation (alg. 3), which, for a given layer, finds the smallest fixed-point representation that causes no information loss, and the PushUp Operation (alg. 4), which, for a given layer, seeks the precision that is required for the network to keep learning. The integration of these two operations into the precision switching mechanism is depicted in alg. 2, and the integration into SGD training process is depicted in alg. 1.

Inducing Sparsity In addition to the AdaPT precision switching mechanism, we used an L1 sparsifying regularizer [66, 67, 68] to obtain sparse and particularly quantization-friendly weight tensors, combined

linearly with L2 regularization for better accuracy in a similar way as proposed by [69]. Additionally using a technique similar to [5], we penalize learning steps that lead to increased word-length or decreased sparsity by:

$$\mathcal{P} = \frac{WL^l}{32} \cdot sp^l$$

where sp^l is the percentage of non-zero elements of layer l. Thus loss in ASGD is computed by:

$$\widehat{\mathcal{L}}(\boldsymbol{W}^l) = \mathcal{L} + \alpha \left\| \boldsymbol{W}^l \right\|_1 + \frac{\beta}{2} \left\| \boldsymbol{W}^l \right\|_2^2 + \mathcal{P}$$

Data: Untrained Float32 DNN
Result: Trained and Quantized DNN

1 $\widehat{\mathbb{L}}$ = InitFixedPointTNVS()
2 $\mathbb{Q}$ = InitQuantizationMapping($\widehat{\mathbb{L}}$)
3 $\mathbb{L}$ = Float32Copy($\widehat{\mathbb{L}}$)
4 **for** *Epoch in Epochs* **do**
5   **for** *Batch in Batches* **do**
6          $\widehat{\mathbb{G}}, \mathcal{L}$ = ForwardPass($\widehat{\mathbb{L}}$, *Batch*)
7          $\mathbb{Q}$ = PrecisionSwitch($\widehat{\mathbb{G}}, \mathcal{L}, \mathbb{Q}, \mathbb{L}$)
8   SGDBackwardsPass($\mathbb{L}, \widehat{\mathbb{G}}$)
9   **for** $l \in \mathbb{L}$ **do**
10             $\widehat{\mathbb{L}}[l]$ = Quantize($\mathbb{L}[l]$, $\mathbb{Q}[l][quant]$)
11   **end**
12   **end**
13 **end**

Algorithm 1: AdaPT-SGD

Algorithm When AdaPT-SGD (alg. 1) is started on an untrained float32 DNN, it first initializes the DNNs layers (denoted as $\widehat{\mathbb{L}}$) weights with TNVS (alg. 1, ln. 1). Next the quantization mapping $\mathbb{Q}$ is initialized, which assigns each $l \in \mathbb{L}$ a tuple $\langle WL^l, FL^l \rangle$, a lookback lb^l and a resolution r^l (alg. 1, ln. 2). Then a float32 master copy $\mathbb{L}$ of $\widehat{\mathbb{L}}$ is created (alg. 1, ln. 3). During training for each forward pass on a batch, quantized gradients $\widehat{\mathbb{G}}$ and loss $\mathcal{L}$ are computed with a forward pass using quantized layers $\widehat{\mathbb{L}}$ (alg. 1, ln. 4-6). The precision switching mechanism described in sec. 3 is then called (alg. 1, ln. 7) and after adapting the push up strategy as described in sec.

Data: $\widehat{\mathbb{G}}, \mathcal{L}, \mathbb{Q}, \mathbb{L}$
Result: $\mathbb{Q}$

1 AdaptStrategy($\mathcal{L}$)
2 **for** $l \in \mathbb{L}$ **do**
3      $\mathbb{Q}[l][grads].append(\widehat{\mathbb{G}}[l])$
4   AdaptLookback($\mathbb{Q}[l][grads], \mathbb{Q}[l][lb]$)
5   AdaptResolution($\mathbb{Q}[l][lb], \mathbb{Q}[l][res]$)
6   **if** $|\mathbb{Q}[l][grads]| \geq \mathbb{Q}[l][lb]$ **then**
7          $WL^l, FL^l = \mathbb{Q}[l][quant]$
8   WL^l_{min}, FL^l_{min} = PushDown($\mathbb{L}[l]$, WL^l, FL^l)
9   WL^l_{new}, FL^l_{new} = PushUp(l, WL^l, $FL^l, WL^l_{min}, FL^l_{min}$)
10         $\mathbb{Q}[l][quant] = WL^l_{new}, FL^l_{new}$
11   **end**
12 **end**

Algorithm 2: PrecisionSwitch

3, it iterates over $l \in \mathbb{L}$ (alg. 2 ln. 2), first adapting resolution r^l and lookback lb^l (alg. 2 ln. 4-5) and then executing PushDown and PushUp on layer l (alg. 2 ln. 6 -10) to update $\langle WL^l, FL^l \rangle \in \mathbb{Q}$. When

---

Data: L, WL^l, FL^l
Result: WL^l_{min}, FL^l_{min}

1 $WL^l_{min}, FL^l_{min} = L, WL^l, FL^l$
2 **repeat**
3   WL^l_{min}, FL^l_{min} = Decrease(WL^l_{min}, FL^l_{min})
4      $\widehat{L}$ = Quantize($L, WL^l_{min}, FL^l_{min}$)
5 **until** $KL(EDF(L_i), EDF(\widehat{L})) < \epsilon$

Algorithm 3: PushDown Operation

PushDown is called on l, it decreases the quantization mapping in a bisectional fashion until KL indicates a more coarse quantization would cause information loss at resolution r^l (3, ln. 1-5). After computing the most coarse quantization $\langle WL^l_m in, FL^l_m in \rangle \in \mathbb{Q}$ not causing information loss in l, PushUp is called on l to increase the quantization to the point where the network is expected to keep learning, based on gradient diversity of the last lb^l batches (alg. 4, ln. 1-6). After PushUp the PrecisionSwitch returns an

9

Data: L, WL^l, FL^l, WL^l_{min}, FL^l_{min}
Result: WL^l_{new}, FL^l_{new}

1 Compute $\widehat{\Delta s(L)}$ using WL^l_{min}, WL^l_{min}

2 Compute $\Delta s(L)$ using WL^l, FL^l

3 while $\Delta s(L) \geq \widehat{\Delta s(L)}$ do

4 WL^l_{min}, FL^l_{min} = Increase(WL^l_{min}, FL^l_{min})

5   Compute $\widehat{\Delta s(L)}$ using WL^l_{min}, FL^l_{min}

6 end

Algorithm 4: PushUp Operation

updated $\mathbb{Q}$ to the training loop, and a regular SGD backward pass updates the float32 master copy $\mathbb{L}$, using quantized gradients $\widehat{\mathbb{G}}$ (alg. 1, ln. 7,8). Finally, the now updated weights $\mathbb{L}$ are quantized using the updated $\mathbb{Q}$ and written back to $\widehat{\mathbb{L}}$ to be using in the next forward pass (alg. 1, ln. 9-11).

4 Experimental Evaluation

4.1 Setup

For experimental evaluation of AdaPT, we trained AlexNet and ResNet20 on the CIFAR-10/100 datasets with reduce on plateau learning rate (ROP) scheduling which will reduce learning rate by a given factor if loss has not decreased for a given number of epochs [46]. Due to unavailability of fixed-point hardware, we used QPyTorch to simulate fixed point arithmetic and our own performance model (sec. 4.1.2) to simulate speedups, model size reductions and memory consumption. Experiments were conducted on an Nvidia DGX-1. It has 8 x Tesla V100 GPUs, which is capable of integer and floating-point arithmetic [70].

4.1.1 Hyper Parameters

All layers $l \in \mathbb{L}$ were quantized with $\langle WL^l, FL^l \rangle = \langle 8, 4 \rangle$ at the beginning of AdaPT training. Other hyperparameters specific to AdaPT were set to $r_{lwr} = 50$, $r_{upr} = 150$, $lb_{lwr} = 25$, $lb_{upr} = 100$, lookback momentum $\gamma = 0.33$ for all experiments, buffer bits

was set to $buff = 4$ for training AlexNet on CIFAR10 and $buff = 8$ for training ResNet and AlexNet on CIFAR100. Hyperparameters unspecific to AdaPT (lr, $L1_{decay}$, $L2_{decay}$, $ROP_{patience}$, $ROP_{threshold}$ *batch size, accumulation steps*) were selected using grid search and 10-fold cross-validation and we refer to our code repository [1] for the exact configuration files of each experiment.

4.1.2 Performance Model

Speedups and model size reductions were computed using a performance model taking forward and backward passes, as well as batch sizes, gradient accumulations numerical precision and sparsity, into account. Our performance model computes per layer operations (MAdds, subsequently referred to as ops), weights them with the layer's world length and a tensors percentage of non-zero elements at a specific stage in training to simulate quantization and a sparse tensor format, and aggregates them to obtain the overall incurred computational costs of all forwards and backwards passes. Additionally, the performance model estimates AdaPTs overhead for each layers l push up operation pu^l and push down operation pd^l by

$$ops^l_{pd,i} \leq 2 \cdot log_2(32 - 8) \cdot r^l_i \cdot 3 \cdot \prod_{dim \in l} dim \qquad (6)$$

$$ops^l_{pu,i} \leq (lb_i + 1) \cdot \prod_{dim \in l} dim + 1 \qquad (7)$$

Using (6), (7) and the simplifying assumption that a backwards pass incurs as many operations as a quantized forwards pass but is conducted in full precision i.e. 32 bits word length with non-sparse gradients, AdaPTs training costs are then bounded by

$$costs_{train} \leq \sum_{i=1}^{n} \sum_{l=1}^{|\mathbb{L}|} ops^l \cdot \left(sp^l_i \cdot WL^l_i + \frac{32}{accs} \right) \quad (8)$$

and AdaPTs overhead is bounded by

$$costs_{AdaPT} \leq \sum_{i=1}^{n} \sum_{l=1}^{|\mathbb{L}|} 32 \cdot \frac{sp^l_i \cdot ops^l_{pd,i} + ops^l_{pu,i}}{accs \cdot lb^l_i} \quad (9)$$

---

[1] https://gitlab.cs.univie.ac.at/sidakk95cs/marvin2

where n is the number of training steps, $\mathbb{L}$ are the networks layers and WL_i^l is the $l-ths$ layers word length at training step i and sp_i^l is the percentage of non-zero elements of layer l at step i. Using (8), (9), we obtain total costs via $costs = cost_{train} + costs_{AdaPT}$ and further the speedup of our training approach compared to MuPPET or a float32 baseline incorporating batch size bs, gradient accumulation steps $accs$ and computational costs $costs$ is thus obtained via

$$SU = \frac{bs_{other} \cdot costs_{other}}{bs_{ours} \cdot costs_{ours}}$$

whereby naturally we exclude AdaPTs overhead when computing $costs_{other}$. Models size reductions SZ were calculated by first computing individual model sizes sz

$$sz = \sum_{l=1}^{|\mathbb{L}|} sp_i^l \cdot WL_i^l$$

with $i = n$ and then forming the quotient $SZ = sz_{other}/sz_{ours}$. Similarly, average memory consumption during training $MEM = mem_{other}/mem_{ours}$ is formed by the quotient of each models memory consumption mem

$$mem = \left(\sum_{i=1}^{n} \sum_{l=1}^{|\mathbb{L}|} \left(sp_i^l \cdot WL_i^l + 32 \right) \right) \cdot \frac{1}{n}$$

Because sz and mem ignore tensor dimensions, they can not be interpreted as absolute values. However given that in the quotients MEM and SZ the effects of tensor dimensions would cancel out when comparing identical architectures (i.e. float32 AlexNet and quantized AlexNet), these values provide valid relative measures for memory consumption and model size under this constraint.

4.2 Results

4.2.1 Training

Tab. 1 shows the top-1 validation accuracies and accuracy differences achieved by AdaPT and MuPPET for both quantized training on CIFAR100, and a float32 basis. Tab. 2 shows results for training on CIFAR10. As can be seen in the tables, AdaPT is capable of quantized training not only to an accuracy comparable to float32 training but even reaches or surpasses iso-accuracy in all examined cases. The chosen experiments furthermore illustrate that AdaPT delivers this performance independent of the underlying DNN architecture or the dataset used. Fig. 3

CIFAR100	Float32	Quantized	Δ
AlexNet$_\text{AdaPT}$	$41.3\ {}_{512}^{100}$	$42.4\ {}_{512}^{100}$	1.1
AlexNet$_\text{MuPPET}$	$39.2\ {}_{128}^{150}$	$38.2\ {}_{128}^{?}$	-1.0
ResNet$_\text{AdaPT}$	$64.3\ {}_{512}^{100}$	$65.2\ {}_{512}^{100}$	0.9
ResNet$_\text{MuPPET}$	$64.6\ {}_{128}^{150}$	$65.8\ {}_{128}^{?}$	1.2

Table 1: Top-1 accuracies, AdaPT (ours) vs MuPPET, CIFAR10, 100 to 150 epochs. Float 32 indicates 32-bit floating-point training (baseline), Quantized indicates variable bit fix-point (AdaPT) or block-floating-point (MuPPET) quantized training, subscript indicates batch size used, superscript indicates epochs used

CIFAR10	Float32	Quantized	Δ
AlexNet$_\text{AdaPT}$	$73.1\ {}_{512}^{100}$	$74.5\ {}_{512}^{100}$	1.4
AlexNet$_\text{MuPPET}$	$75.5\ {}_{128}^{150}$	$74.5\ {}_{128}^{99}$	-1.0
ResNet$_\text{AdaPT}$	$89.5\ {}_{512}^{100}$	$90.0\ {}_{512}^{100}$	0.5
ResNet$_\text{MuPPET}$	$90.1\ {}_{128}^{150}$	$90.9\ {}_{128}^{114}$	0.8

Table 2: Top-1 accuracies, AdaPT (ours) vs MuPPET, CIFAR100

and 4 display word length usage over time for individual layers for ResNet20 and AlexNet on CIFAR100. In both cases, we observe that individual layers have different precision preferences dependent on progression of the training, an effect that is particularly pronounced in AlexNet. For ResNet20, the word length interestingly decreased notably in the middle of training and later decreased, which we conjecture is attributable to sparsifying L1 regularization as well as the wordlength/sparsity penalty introduced in sec. 3 leading to a reduction in irrelevant weights, that

would otherwise offset the KL heuristic used in the PushDown Operation. Tabs. 4 and 3 show AdaPTs
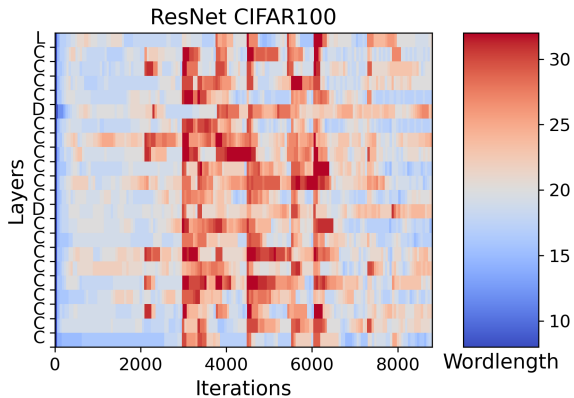

Figure 3: *Wordlengths (bit) of ASGD optimized ResNet20 on CIFAR100, 100 epochs, Linear (L), Convolutional (C), Downsampling (D) layers*

CIFAR10				
	MEM	**SU**^{1}	**SU**^{2}	**SU**^{3}
AlexNet$_{\text{AdaPT}}$	1.47	1.42	2.37	6.42
AlexNet$_{\text{MuPPET}}$	1.66	?	?	1.2
ResNet$_{\text{AdaPT}}$	1.63	1.20	1.49	4.01
ResNet$_{\text{MuPPET}}$	1.61	?	?	1.25

Table 3: Memory Footprint, Final Model Size, Memory Footprint, Speedup, AdaPT (ours) vs MuPPET on respective baseline float32 training on CIFAR10. SU^1: our baseline, our performance model, SU^2: our baseline, our performance model adjusted for iso-accuracy, SU^3: MuPPET baseline, our performance model

CIFAR100				
	MEM	**SU**^{1}	**SU**^{2}	**SU**^{3}
AlexNet$_{\text{AdaPT}}$	1.41	1.32	1.57	5.23
ResNet$_{\text{AdaPT}}$	1.66	1.13	1.62	3.36

Table 4: Memory Footprint, Final Model Size, Memory Footprint, Speedup, AdaPT (ours) vs MuPPET, CIFAR100, training
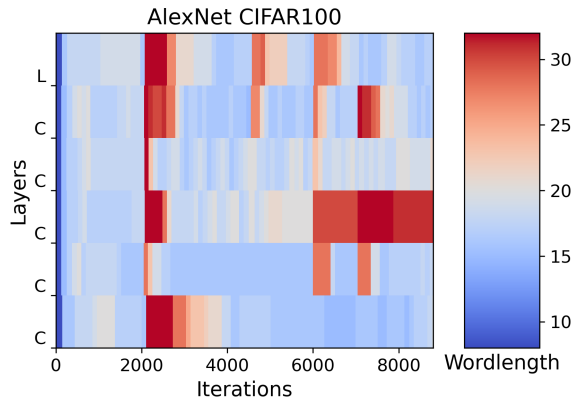

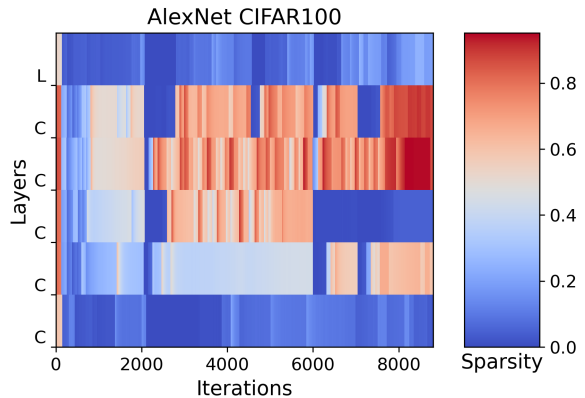Figure 4: *Wordlengths (bit) of ASGD optimized AlexNet on CIFAR100, 100 epochs*

Figure 5: *Sparsity of ASGD optimized AlexNet on CIFAR100, 100 epochs*

estimated speedups vs. our float32 baseline (100 epochs, same number of gradient accumulation steps and batch size as AdaPT trained models) and MuP-PETs float32 baseline. AdaPT achieves speedups comparable with SOTA solutions on our own baseline and outperforms MuPPET on MuPPETs baseline in every scenario. Unfortunately, MuPPETs code base could not be executed so we were unable to apply MuPPET to the more efficient AdaPT baseline and were limited to comparing against results provided by the original authors. We used our performance

model for simulating MuPPETs performance based on the precision switches stated in the MuPPET paper because MuPPETs authors did not publish their performance model. An overview over the reduction of computational costs through ASGD relative to a float32 baseline is provided by fig .8 as well Fig. 5
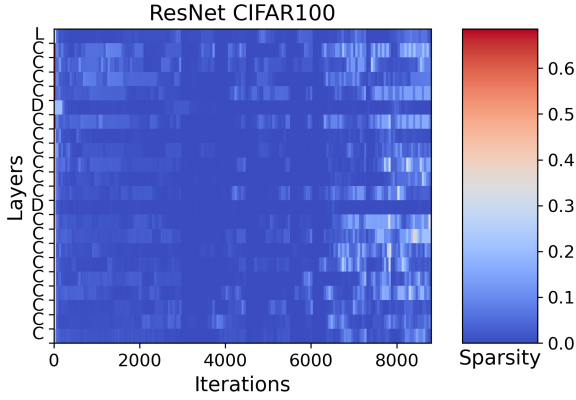
Sparsity		
	Final Model	**Average**
AlexNet$_{CIFAR10}$	0.26	0.27
ResNet$_{CIFAR10}$	0.07	0.04
AlexNet$_{CIFAR100}$	0.44	0.35
ResNet$_{CIFAR100}$	0.07	0.03

Table 5: Final model sparsity and average intra-training sparsity for AdaPT Training

Figure 6: *Sparsity of ASGD optimized ResNet on CIFAR100, 100 epochs*

and fig. 6 illustrate the induction of sparsity during AdaPT training. Interestingly in some cases, we observe an increasing degree of sparsity as AdaPT training progresses, with some layers reaching 80% sparsity or more at the end of training as is the case with AlexNet trained on CIFAR100, while less sparsity could be induced in fewer layers training ResNet on CIFAR10/100. As can be seen in tab. 5, AdaPT induces most sparsity in AlexNet, with a 45% final model sparsity and an average intra-training sparsity of 34% for training on CIFAR100. In the residual architecture ResNet, we observe that AdaPT introduces 7% sparsity in the final model and 3% intra-training sparsity. As fig. 6 illustrates increasing sparsity towards the end training, we conjecture that for ResNet-like architectures, the sparsity inducing effect could be more pronounced if training is conducted for a higher number of epochs.

A side effect illustrated by fig. 7 and tab. 4 and tab. 2 of AdaPT for reducing the computational cost of network training (fig. 8) is it's increased intra-training memory consumption that is caused by
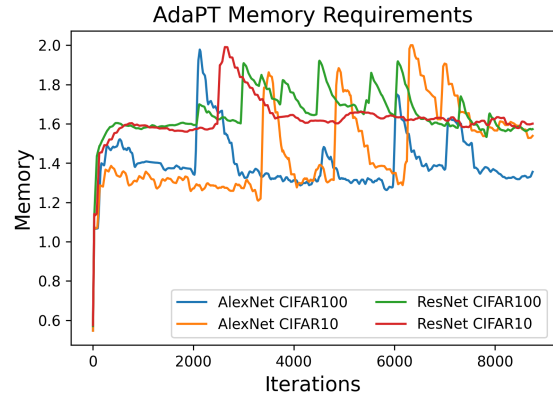

Figure 7: *ASGD Memory Consumption relative to float32 SGD*

Figure 8: *ASGD computational cost relative to float32 SGD*

Inference		
	SZ	**SU**
AlexNet$_{\text{CIFAR10}}$	0.54	3.56
ResNet$_{\text{CIFAR10}}$	0.60	1.63
AlexNet$_{\text{CIFAR100}}$	0.36	2.60
ResNet$_{\text{CIFAR100}}$	0.57	1.52

Table 6: Inference with AdaPT trained models

AdaPT maintaining a complete float32 mastercopy of the models weights tensors used for backwards pass updates. However this effect is only present during training as the master-copies are discarded once the model is deployed and used for inference.

4.2.2 Inference

Given AdaPT trained networks are fully quantized and sparsified, they have advantage that extends beyond the training phase into the inference phase. Tab. 6 shows that inference speed ups for AdaPT trained networks range from 1.63 to 3.56. The speed ups achieved during the inference phase are even higher than those achieved during training because during inference, no expensive backwards pass has to be conducted and no overhead induced by AdaPT is occurring. This is a non-trivial advantage over MuP-PET which does not provide any post-training advantages in terms of speed up or memory consumption at all due to the resulting network being float32.

5 Conclusion

With AdaPT we introduce a novel quantized DNN sparsifying training algorithm that can be combined with iterative gradient based training schemes. AdaPT exploits fast fixed-point operations for forward passes, while executing backward passes in highly precise float32. By carefully balancing the need to minimize the fixed-point precision for maximum speedup and minimal memory requirements, while at the same time keeping precision high enough for the network to keep learning, our approach produces top-1 accuracies comparable or better than SOTA float32 techniques or other quantized train-

ing algorithms. Furthermore AdaPT has the intrinsic methodical advantage of not only training the network in a quantized fashion, but also the trained network itself is quantized and sparsified s.t. it can be deployed to fast ASIC hardware at reduced memory cost and executed with reduced computational compared to a full precision model. Additionally we contribute a performance model for fixed point quantized training.

6 Future Work

AdaPT as presented employs a fixed-point representation to enable fine granular precision switches. However, fixed-point hardware is not as common as floating-point hardware, so we plan to extend the concept to floating point quantization s.t. AdaPT becomes compatible with float16/float32 consumer hardware. Further, we intend to explore whether AdaPT can be used to generate very low bit-width (binarization/ternarization) networks through gradually reducing the quantization search space during training. Additionally, we conjecture the heuristics used by AdaPT can be used for intra-training DNN pruning as well, which will be subject to future research. We plan ablation testing to reduce the complexity of AdaPT. Another interesting application of AdaPT which we intend to explore is it's usefulness in other fields of research such as Drug Discovery.

7 Acknowledgements

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional

neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[3] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via overparameterization," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 242–252, PMLR, 09–15 Jun 2019.

[4] M. Li, M. Soltanolkotabi, and S. Oymak, "Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics* (S. Chiappa and R. Calandra, eds.), vol. 108 of *Proceedings of Machine Learning Research*, pp. 4313–4324, PMLR, 26–28 Aug 2020.

[5] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1586–1595, 2018.

[6] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[8] M. Lis, M. Golub, and G. Lemieux, "Full deep neural network training on a pruned weight budget," in *Proceedings of Machine Learning and Systems* (A. Talwalkar, V. Smith, and M. Zaharia, eds.), vol. 1, pp. 252–263, 2019.

[9] R. Stewart, A. Nowlan, P. Bacchus, Q. Ducasse, and E. Komendantskaya, "Optimising hardware accelerated neural networks with quantisation and a knowledge distillation evolutionary algorithm," *Electronics*, vol. 10, no. 4, 2021.

[10] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 02 2016.

[11] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[12] K. Wang and A. N. Michel, "Robustness and perturbation analysis of a class of artificial neural networks," *Neural Networks*, vol. 7, no. 2, pp. 251–259, 1994.

[13] A. Meyer-Base, "Perturbation analysis of a class of neural networks," in *Proceedings of International Conference on Neural Networks (ICNN'97)*, vol. 2, pp. 825–828 vol.2, 1997.

[14] Xiaoqin Zeng and D. S. Yeung, "Sensitivity analysis of multilayer perceptron to input and weight perturbations," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1358–1366, 2001.

[15] L. Xiang, X. Zeng, Y. Niu, and Y. Liu, "Study of sensitivity to weight perturbation for convolution neural network," *IEEE Access*, vol. 7, pp. 93898–93908, 2019.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] D. M. Loroch, F.-J. Pfreundt, N. Wehn, and J. Keuper, "Tensorquant: A simulation toolbox

for deep neural network quantization," in *Proceedings of the Machine Learning on HPC Environments*, pp. 1–8, 2017.

[18] T. Zhang, Z. Lin, G. Yang, and C. De Sa, "Qpytorch: A low-precision arithmetic simulation framework," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pp. 10–13, 2019.

[19] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[20] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, "Quantization networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7308–7316, 2019.

[21] C. Baskin, N. Liss, E. Schwartz, E. Zheltonozhskii, R. Giryes, A. M. Bronstein, and A. Mendelson, "Uniq: Uniform noise injection for non-uniform quantization of neural networks," *ACM Trans. Comput. Syst.*, vol. 37, Mar. 2021.

[22] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[23] A. T. Elthakeb, P. Pilligundla, F. Mireshghallah, A. Yazdanbakhsh, and H. Esmaeilzadeh, "Releq : A reinforcement learning approach for automatic deep quantization of neural networks," *IEEE Micro*, vol. 40, no. 5, pp. 37–45, 2020.

[24] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, and M. Aleksic, "A quantization-friendly separable convolution for mobilenets," in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pp. 14–18, IEEE, 2018.

[25] J. Achterhold, J. M. Koehler, A. Schmeink, and T. Genewein, "Variational network quantization," in *International Conference on Learning Representations*, 2018.

[26] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, Curran Associates, Inc., 2015.

[27] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, "Structured bayesian pruning via log-normal multiplicative noise," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), pp. 6775–6784, 2017.

[28] E. Tartaglione, S. Lathuilière, A. Fiandrotti, M. Cagnazzo, and M. Grangetto, "Hemp: High-order entropy minimization for neural network compression," *Neurocomputing*, vol. 461, pp. 244–253, 2021.

[29] E. Tartaglione, A. Bragagnolo, A. Fiandrotti, and M. Grangetto, "Loss-based sensitivity regularization: towards deep sparse neural networks," *CoRR*, vol. abs/2011.09905, 2020.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Model optimization," 2015. Software available from tensorflow.org.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: Quantization," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[32] H. Li, S. De, Z. Xu, C. Studer, H. Samet, and T. Goldstein, "Training quantized nets: A deeper understanding," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 5813–5823, 2017.

[33] P. Yin, J. Lyu, S. Zhang, S. J. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," 2019.

[34] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *CoRR*, vol. abs/1308.3432, 2013.

[35] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in Neural Information Processing Systems*, vol. 30, pp. 1709–1720, 2017.

[36] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *arXiv preprint arXiv:1610.02132*, 2016. https://arxiv.org/pdf/1610.02132.pdf.

[37] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[38] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[39] P. Yin, S. Zhang, Y. Qi, and J. Xin, "Quantization and training of low bit-width convolutional neural networks for object detection," *Journal of Computational Mathematics*, vol. 37, no. 3, pp. 349–359, 2018.

[40] L. Hou and J. T. Kwok, "Loss-aware weight quantization of deep networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018.

[41] T. Chu, Q. Luo, J. Yang, and X. Huang, "Mixed-precision quantized neural networks with progressively decreasing bitwidth," *Pattern Recognition*, vol. 111, p. 107647, 2021.

[42] D. Lee, D. Wang, Y. Yang, L. Deng, G. Zhao, and G. Li, "Qttnet: Quantized tensor train neural networks for 3d object and video recognition," *Neural Networks*, vol. 141, pp. 420–432, 2021.

[43] P. Peng, M. You, W. Xu, and J. Li, "Fully integer-based quantization for mobile convolutional neural network inference," *Neurocomputing*, vol. 432, pp. 194–205, 2021.

[44] A. Rajagopal, D. A. Vink, S. I. Venieris, and C.-S. Bouganis, "Multi-precision policy enforced training (muppet): A precision-switching strategy for quantised fixed-point training of cnns," *arXiv preprint arXiv:2006.09049*, 2020.

[45] J. H. Wilkinson, "Rounding errors in algebraic processes," p. 2, 1994.

[46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang,

J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[47] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[48] J. H. Wilkinson, "Rounding errors in algebraic processes," pp. 26–27, 1994.

[49] J. H. Wilkinson, "Rounding errors in algebraic processes," p. 1, 1994.

[50] M. Hopkins, M. Mikaitis, D. R. Lester, and S. Furber, "Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ordinary differential equations," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20190052, 2020.

[51] D. Yin, A. Pananjady, M. Lam, D. Papailiopoulos, K. Ramchandran, and P. Bartlett, "Gradient diversity: a key ingredient for scalable distributed learning," in *International Conference on Artificial Intelligence and Statistics*, pp. 1998–2007, 2018.

[52] A. Kerr, H. Wu, M. Gupta, D. Blasig, P. Ramani, N. Farooqui, P. Majcher, P. Springer, J. Wang, S. Yokim, M. Hohnerbach, A. Atluri, D. Tanner, T. Costa, J. Demouth, B. Fahs, M. Goldfarb, M. Hagog, F. Hu, A. Kaatz,

T. Li, T. Liu, D. Merrill, K. Siu, M. Tavenrath, J. Tran, V. Wang, J. Wu, F. Xie, A. Xu, J. Yang, X. Zhang, N. Zhao, G. Bharambe, C. Cecka, L. Durant, O. Giroux, S. Jones, R. Kulkarni, B. Lelbach, J. McCormack, and K. P. and, "Nvidia cutlass," 2020. Online, accessed 15.07.2020.

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[54] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[55] G. H. Alex Krizhevsky, Vinod Nair, "The cifar-10 and cifar-100 dataset," 2019. Online, accessed 15.07.2020.

[56] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[58] B. Hanin and D. Rolnick, "How to start training: The effect of initialization and architecture," in *Advances in Neural Information Processing Systems*, pp. 571–581, 2018.

[59] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Advances in neural information processing systems*, pp. 971–980, 2017.

[60] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, pp. 9–48, Springer, 2012.

[61] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, and N. Kwak, "Lsq+: Improving low-bit quantization through learnable offsets and better initialization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 696–697, 2020.

[62] S. Basodi, C. Ji, H. Zhang, and Y. Pan, "Gradient amplification: An efficient way to train deep neural networks," *CoRR*, vol. abs/2006.10560, 2020.

[63] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, "Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks," in *International Conference on Machine Learning*, pp. 794–803, 2018.

[64] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in neural information processing systems*, pp. 901–909, 2016.

[65] P. Zhou, J. Feng, C. Ma, C. Xiong, S. C. H. Hoi, and W. E, "Towards theoretically understanding why sgd generalizes better than adam in deep learning," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 21285–21296, Curran Associates, Inc., 2020.

[66] P. M. Williams, "Bayesian regularization and pruning using a laplace prior," *Neural computation*, vol. 7, no. 1, pp. 117–143, 1995.

[67] A. Y. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," in *Proceedings of the twenty-first international conference on Machine learning*, p. 78, 2004.

[68] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[69] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[70] N. Corporation, "NVIDIA DGX-1essential instrument for ai research," 2017. Online, accessed 07.07.2021.