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

Lorenz Kummer*†‡ Kevin Sidak*§ Tabea Reichmann*¶ Wilfried Gansterer *‖

Abstract

Quantizing deep neural networks (DNNs) is an important strategy for training or inference in time critical applications. State-of-the-art quantization approaches focus on *post-training* quantization. While some work on quantization during training exists, most approaches require refinement in full precision (usually single precision) in the final training phase, use a rather coarse quantization, that leads to a loss in accuracy, or enforce a global bit-width across the entire DNN. This leads to suboptimal assignments of bit-widths to layers and, consequently, suboptimal resource usage. To overcome such limitations, we introduce AdaPT, a new *fixed-point* quantized sparsifying training strategy for deep neural networks. AdaPT decides about precision switches between training epochs based on an information theory motivated heuristic. On a *per-layer basis*, AdaPT chooses 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 this quantization are evaluated based on an analytical performance model. We illustrate an average 1.31× (or 4.76× adjusted for iso-accuracy) speedup compared to standard training in float32 at *iso-accuracy*, even achieving an average accuracy increase of 0.74 percentage points for AlexNet/ResNet-20 on CIFAR10/CIFAR100/EMNIST and LeNet-5/MNIST. We demonstrate that these trained models reach an average inference 2.28× speedup with a model size reduction up to 51% of the corresponding unquantized model.

1 Introduction

As machine learning models become increasingly complex, inference in time-critical applications or training under resource constraints becomes challenging. Applications such as robotics, augmented reality, self-driving vehicles, mobile applications and scientific research often require high number of trained models for hyperparameter optimization. Accompanied by this, already some of the most common DNN architectures, such as 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]), or quantization. Network pruning reduces a DNN's model size (e.g. by sparsification of weights tensors and using a sparse tensor format), and therefore also improves the runtime. However, pruning methods do not typically exploit the advantages of low-bitwidth arithmetic to reduce computational resource consumption. We propose an approach that combines pruning in the form of sparsification with quantization to improve resource efficiency while maintaining accuracy. Quantization must be used with caution to avoid negative impact on accuracy and convergence during training.

AdaPT, a new approach to quantization during training, has been introduced to improve upon existing methods. The approach utilizes a per-layer precision switching mechanism based on the Kullback-Leibler Divergence (KL) [7], and a novel memory efficient layer-wise formulation of Gradient Diversity (GD) [8] to dynamically adjust precision as needed. This addresses the limitations of current quantization methods that do not consider the impact on different layers and lack dynamic precision control. After quantized training, our approach does not need a refinement phase in full precision. That is, AdaPT does not dequantize the model after quantized training for additional full-precision refinement epochs in contrast to, e.g., [9]. Instead, it produces an already fixed-point quantized network to be directly deployed on high-performance application-specific integrated circuits (ASICs) or field-programmable gate array (FPGA) hardware.

1.1 Related Work For exploring the accuracy degradation induced by quantizations of weights, activation functions and gradients, [10] introduced *QPyTorch*, capable of simulating the most common quantizations for training and inference tasks on a float32 basis. Since quantization is only simulated in QPyTorch, no runtime speedup can be achieved on this basis.

Quantized Inference Several approaches have tried to minimize inference accuracy degradation induced by quantizing weights or activation functions while leveraging associated performance increases. [11] and [12] incorporate simulated quantization into model training and train the model itself to compensate the introduced errors. A similar approach is taken by [13], which emulates a non-uniform quantizer to inject noise at training time

*University of Vienna, Faculty of Computer Science, Vienna, Austria
†University of Vienna, Doctoral School Computer Science, Vienna, Austria
‡lorenz.kummer@univie.ac.at
§kevin.sidak@univie.ac.at
¶tabea.reichmann@univie.ac.at
‖wilfried.gansterer@univie.ac.at

to obtain a model suitable for quantized inference. [14]'s approach learns optimal quantization schemes through jointly training DNNs and associated quantizers. [15] use 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 by [16]. A different approach is taken by [17] which uses variational dropout training-with a structured sparsity inducing prior to formulate post-training quantization as the variational inference problem looking for the posterior optimizing the KL. [18] introduce an entropy coding-based regularizer minimizing the quantized parameters entropy in gradient-based learning and pairs it with a separate pruning scheme [19] to reach a high degree of model compression after training. Using reinforcement learning and the hardware accelerator's feedback, [20] automatically determines the quantization policy optimal for inference on a specific target hardware.

Quantization in machine learning frameworks like PyTorch and Tensorflow is typically performed via post-training quantization or quantization-aware training (QAT, which only simulates quantization). However, these methods offer no efficiency gain during training and require additional computational overhead.

Quantized Training (Distributed) For distributed training on multi-node environments, [21] incorporates a family of gradient compression schemes aimed at reducing inter-node communication occurring during stochastic gradient descent's (SGD) gradient updates. Conceptionally similar, [22] also seeks to reduce communications overhead in distributed training through quantization.

Quantized Training (Local) Very low bit width single-node training (binary, ternary or similarly quantized weights and/or activations), usually in combination with straight-through estimator (STE) [23], 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 [24, 25, 26, 27, 28, 29]. Its most capable representative [30], 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 percentage points of accuracy degradation compared to its respective baseline. A low bit-width integer quantized (re-)training approach reporting accuracy drops of 1 percentage point or less compared to a float32 baseline was introduced by [31], but it requires networks pre-trained in full precision as starting points and its explicitly stated goal is reducing computational cost and model size during inference. Documenting at energy savings but not speedups, a simple but effective integer quantized training approach assigning each layer an individual bit-width was presented by [32]. Similarly, [33, 34] quantize training for the purpose of reducing energy consumption during training. For speeding up training on FPGAs or ASICs via block-floating point quantization, [9] introduced a dynamic training quantization scheme (*Multi Precision Policy Enforced Training*, MuPPET). The algorithm stores two copies of the network's weights: a float32 master copy of the weights that is updated during the weight update step and a block-floating point quantized copy used for forward passes and gradient computation. MuPPET uses a global bit-width that starts at 8 bit and increases at the end of an epoch by a certain step size if an empirically determined threshold is violated more than a certain number of times by a GD- based metric whereby both the step size and the available bit-widths are empirically determined before the actual training of the DNN. Speedups claimed by MuPPET are estimated based on a performance model simulating fixed-point arithmetic on GPUs which support floating and integer arithmetic. We chose MuPPET as baseline to compare AdaPT's training performance against because of the examined related work for the quantized training task, MuPPET comes closest to the goal of iso-accuracy. By iso-accuracy, we refer to the concept that at least the same accuracy can be reached with the quantized network compared to the unquantized network (i.e. no accuracy is lost).

Open Questions Existing quantized training and inference solutions leave room for improvements in several aspects. Approaches proposed in [11, 12, 14, 15, 13, 18, 20] and [17] produce networks where quantization during inference leads to only small accuracy degradation relative to some baselines. Unfortunately, the baselines are not comparable and it is unclear how well they are optimized. Moreover, these approaches require computationally expensive float32 training and the algorithms themselves incur a certain overhead as well. The training method in [21] only quantizes gradients and focuses on reducing communication overhead in multi-node environments, a goal shared with [22]. Binary, ternary or similarly quantized training has been shown to commonly lead to reduced accuracy, reduced convergence speed or only quantizing weights and not activations [24, 25, 26, 27, 28, 29]. Even the most capable representative introduced in [30] does not achieve iso-accuracy. Assigning different bit-widths to different layers at training time as discussed in [32] can lead to limited loss in accuracy. While the authors do not claim any speedup during training, they show that their approach reduces energy consumption, like in [33, 34].

MuPPET [9] requires at least N epochs for N quantization levels because 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. 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. MuPPET outputs a float32 network such that inference has to be done in expensive full precision and no measures are taken to mitigate GD's memory overhead.

1.2 Problem Definition Based on this assessment of the state-of-the-art, we address the problems of efficient heuristics for accelerating (in terms of computational cost) single node *training* through layer-wise quantization without sacrificing accuracy in the sense of the fraction of predictions the model indicated correctly. That is, the same or better accuracy than the float32 baseline model shall be achieved by the quantized model (i.e iso-accuracy). This is to be achieved through assigning an individual quantization dynamically to each layer during training and without relying on expensive full precision refinement after quantized training. Per-layer bit-width assignment is motivated by the common observation that different layers of a network extract different levels of features, e.g., [35, 36], which suggests that different layers may have different precision requirements, which can be addressed by per-layer bit-width assignment.

1.3 Contributions AdaPT is a state-of-the-art solution for quantized DNN training using an information-theoretical intra-epoch precision switching mechanism for dynamic precision adjustment on a per-layer basis, and introducing a novel memory efficient layer-wise formulation of GD. It quantizes weights and activations to the lowest bit-width without information loss while inducing sparsity, and reaches or surpasses iso-accuracy while reducing computational cost and model size. AdaPT demonstrates advantages over float32-baseline and MuPPET by training representative model-dataset combinations and developing an analytical model for computational cost, and also carries over its advantages to the inference phase unlike MuPPET which outputs a dense float32 model.

To the best of our knowledge, currently no work exists on accelerating *training* through dynamically determined fixed-point quantization on a per-layer level while achieving iso-accuracy. Existing work either focusses on other types of quantization or on different scenarios (e.g. inference, distributed computing, energy savings) or does not achieve iso-accuracy in quantized training. This underlines our work's relevancy as it shows that information theory-based heuristics can select the optimal fixed-point precision for each layer dynamically during training while achieving iso-accuracy.

2 Background

Quantization reduces computational costs and memory consumption by performing computation at reduced numerical precision or representation. Fixed-point or block-floating-point representations are commonly used in high-performance ASICs or FPGAs, while floating-point and integer representations are available on consumer hardware. However, quantized execution may introduce errors through large machine epsilon ϵ_{mach} or small representable range of the quantized representation.

Fixed-Point Quantization Fixed-point numbers have a fixed number of decimal digits assigned, and hence every computation must be framed s.t. the results lies within the given boundaries of the representation.A signed fixed-point numbers of bit-width $BW = i + s + 1$ 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.

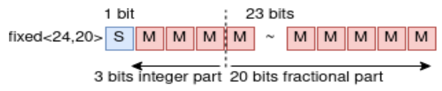

Figure 1: of *Example of a possible fixed-point representation. Sign (S), Exponent (E), Mantissa (M) bits.*

3 Adaptive Precision Training

3.1 Precision Levels AdaPT uses fixed-point quantization (defined in Section 2), as it provides superior control over numerical precision and is supported by high-performance ASICs, our target platform. This method allows for optimal bit-width and fractional length which are local properties of each layer, as different layers contain different amounts of information during different points of time in training. Floating-point and extreme forms of integer quantization such as binarization and ternarization were rejected as they are not appropriate for our goal of iso-accuracy. However, the AdaPT concept can be extended to other representations. AdaPT employs stochastic rounding, which in combination with a fixed-point representation has been shown to consistently outperform nearest-rounding by [37], 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 $\langle BW^l, FL^l\rangle$ whereby the fractional length FL^l denotes the number of fractional bits and BW^l denotes the total number of bits. For a random number $P \in [0, 1]$, x is stochastically rounded by

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

3.2 AdaPT-SGD (ASGD) Although AdaPT can algorithmically 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 in computer vision tasks as shown by [38] for float32 training. Our preliminary observations suggest this holds for quantized SGD as well, however an in-depth investigation of the ability of different optimizers to generalize well in a quanzized setting is considered beyond the scope of this work. The integration of AdaPT into SGDs training loop is accomplished by executing AdaPTs precision seeking heuristic described in Section 3 after gradient computation and then, after the weights update step, applying the quantization found by AdaPT to each layer, see Figure 2. Similar to MuPPET, ASGD

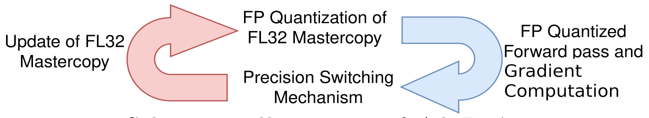

Figure 2: *Schematic illustration of AdaPT's integration into SGD training, float32 (FL32), fixed point (FP)*

stores two copies of the network's weights: a float32 master copy of the weights that is updated in the weights update step and a fixed-point point quantized copy used for forward passes and gradient computation. The activations of a layer are quantized to the same level the layers' weights, gradients are dequantized before the update of the float32 mastercopy.

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 to improve runtime and model size, yet still maintain enough precision for the network to keep learning. In AdaPT's precision switching mechanism, we have encoded these opposing goals in two operations, the Push-Down Operation (PD) and the Push-Up Operation (PU), which separatly track heuristics associated with each of these two goals. The PD and PU operations are executed for each layer $l \in \mathbb{L}$ indiviudally every lb^l batches, which allows fine-granular control of layer-wise precision levels as well as the frequency of the precision switches, reducing the overhead of the associated heuristics compared to execution at every batch. The variable lb^l iteself, which subsequently will also be referred to as lookback, is dynamically inferred at runtime based on each layer l's GD metric (details see below).

PD 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 introduced by [7] 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:

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

Although KL is not a metric, KL is very common in machine learning because it is easily optimizable, related to maximum likelihood estimation, and is able to sample gradients in an unbiased way, unlike e.g. the Wasserstein metric. Using discretization via binning, we obtain P^l and Q^l at resolution r^l through the empirical distribution function

$$\widehat{F_{r^l}}(w) = \frac{1}{r^l + 1} \sum_{i=1}^{r^l} 1_{W_i^l \le w} \tag{3.1}$$

where $1_{W_i^l \le w}$ denotes the indicator function. P^l and Q^l can then be used in the discrete Kullback-Leibler-Divergence (3.2).

$$KL(P^l \| Q^l) = \sum_{w \in \boldsymbol{W}^l} P^l(w) \cdot log \frac{P^l(w)}{Q^l(w)} \tag{3.2}$$

Using a bisection approach, AdaPT efficiently finds the smallest quantization $\langle BW^l_{min}, FL^l_{min} \rangle$ of $\boldsymbol{W}^l$ s.t. $KL(P^l \| Q^l) = 0 \ \forall l \in \mathbb{L}$. By first applying the empirical distribution function and discretizing via binning before computing the KL, the value 0 can be achieved.

PU 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 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. $\langle BW^l, FL^l \rangle = \langle 8, 4 \rangle$) would result in the network failing to learn because at such low precision levels, gradients would vanish very early in the backwards pass. Hence, AdaPT tracks for each layer a novel layer-wise GD heuristic over the last lb^l batches to determine how much precision is required for the network to keep learning:

$$\Delta s(\boldsymbol{W}^l)_j = \frac{\sum_{k=j-lb^l}^{j} \| \nabla f_k(\boldsymbol{W}^l) \|_2}{\| \sum_{k=j-lb^l}^{j} \nabla f_k(\boldsymbol{W}^l) \|_2} \tag{3.3}$$

If gradients are normalized as in (3.4), which is a common technique for coping with the problem of vanishing and exploding gradients [39, p. 142], GD as defined in (3.3) becomes (3.5) and can be computed without having to allocate memory for collecting the last lb^l gradients of l. Only l's accumulated gradient has to be stored.

$$\nabla f(\boldsymbol{W}^l) = \frac{\nabla f(\boldsymbol{W}^l)}{\|\nabla f(\boldsymbol{W}^l)\|_2} \tag{3.4}$$

$$\Delta s(\boldsymbol{W}^l)_j = \frac{lb^l}{\|\sum_{k=j-lb^l}^{j} \nabla f_k(\boldsymbol{W}^l)\|_2} \tag{3.5}$$

This reduces the memory complexity of computing l's GD from $\mathcal{O}(lb^l \cdot m^l)$ to $\mathcal{O}(m^l)$ where m^l is the memory required to store l's accumulated gradients. This is equal to the storage requirements of l's trainable weights. We apply this technique to compute GD without the memory overhead incurred by originally proposed GD used in MuPPET.

Next, two suggestions for an increase in precision are computed, $s_1^l = max(\lceil \frac{1}{log^2 \Delta s(\boldsymbol{W}^l)_{j-1}} \rceil, 1)$ and $s_2^l = max(min(32 \cdot log^2 \Delta s(\boldsymbol{W}^l)_j - 1, 32) - FL_{min}^l, 1)$. We chose the two s_n^l as functions which we empirically evaluated to be suitable bounds, other functions of $\Delta s(\boldsymbol{W}^l)_j$ providing suitable bounds might exist as well. The final suggestion is computed dependent on a global strategy st via

$$s^l = \begin{cases} min(s_1^l, s_2^l) & \text{if st = 'min'} \\ \lceil 0.5 \cdot (s_1^l + s_2^l) \rceil & \text{if st = 'mean'} \\ max(s_1^l, s_2^l) & \text{if st = 'max'} \end{cases} \tag{3.6}$$

The new fixed-point quantization of layer l is then obtained by $FL^l = (min(FL_{min}^l + s^l, 32)$, $BW^l = min(max(BW_{min}^l, FL_{min}^l) + 1, 32))$.

Strategy, Resolution and Lookback To let AdaPT adjust to the important feedback provided by the loss function regarding the progress of learning, we introduce the concept of strategy, which controls how our quantization heuristic finds the optimal bit widths and fractional lengths for each layer. For adapting the strategy st mentioned in (3.6), we employ a simple loss-based heuristic. First we compute the average lookback over all layers $lb_{avg} = |\mathbb{L}|^{-1} \sum_{l=0}^{|\mathbb{L}|} lb^l$ and average loss $\mathcal{L}_{avg} = |\mathcal{L}|^{-1} \sum_{i=0}^{lb_{avg}} \mathcal{L}_i$ over the last lb_{avg} batches. Then the strategy is adapted:

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

Because the number of gradients used by GD for each layer affects the result of the GD heuristic (3.5), we use the above-mentioned lookback lb^l bounded by hyperparameters $lb_{lwr} \leq lb^l \leq lb_{upr}$ which is estimated at runtime. First, lb_{new} is computed:

$$lb_{new}^l = min(max(\lceil \frac{lb_{upr}}{\Delta s(\boldsymbol{W}^l)_j} \rceil, lb_{lwr}), lb_{upr})$$

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 (3.1) affects the result of the discrete KL (3.2). We control the number of bins via a parameter referred to as resolution r^l, which is derived at runtime and bounded by hyperparameters $r_{lwr} \leq r^l \leq r_{upr}$.

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

Dealing with Fixed-Points Limited Range As outlined in Section 2, 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 layer's bit-width, i.e. at the end of PU, $FL^l = (min(FL_{min}^l, 32 - buff)$, $BW^l = max(min(FL_{min}^l + buff, 32), BW_{min}^l)$. Additionally, we normalize gradients ((3.4)) to limit weight growth and reduce chances of weights becoming unrepresentable in the given precision after an update step.

Inducing Sparsity In addition to the AdaPT precision switching mechanism, which as a by-product of quantization already induces a certain degree of sparsity through the rounding of near-zero weights [40] (see Figure 3), we further leverage sparsity by using an L1 sparsifying regularizer (see e.g. [41]) to obtain sparse and particularly quantization-friendly weight tensors, combined linearly with L2 regularization for better accuracy similarly as proposed by [42].

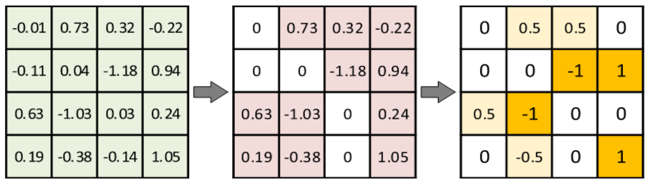

Figure 3: *Schematic illustration how sparsity can be increased through regularization and quantization [40]*

3.4 ASGD Execution Details When ASGD is started on an untrained float32 DNN, it initializes the DNNs layers (denoted as $\widehat{\mathbb{L}}$) weights with truncated normal variance scaling [43]. Next the quantization mapping $\mathbb{Q}$ is initialized, which assigns each $l \in \widehat{\mathbb{L}}$ a tuple $\langle BW^l, FL^l \rangle$, a lookback lb^l and a resolution r^l. Then a float32 master copy $\mathbb{L}$ of $\widehat{\mathbb{L}}$ is created and $\widehat{\mathbb{L}}$ is quantized using $\mathbb{Q}$. During training, for each quantized forward pass using $\widehat{\mathbb{L}}$ on batch i, quantized gradients $\widehat{\mathbb{G}}$ and loss $\mathcal{L}$ are computed. The precision switching mechanism is then called and after adapting the strategy, it iterates over $l \in \mathbb{L}$: if i mod. $lb^l = 0$, first r^l and lb^l are adapted and then PD and PU are executed on layer l to update $\langle BW^l, FL^l \rangle \in \mathbb{Q}$. When PD is called on l, it decreases $\langle BW^l, FL^l \rangle$ using bisection until KL indicates a more coarse quantization would cause information loss at resolution r^l. After computing the most coarse quantization not causing information loss in l, PU is called on l to increase $\langle BW^l, FL^l \rangle$ to the point where the network is expected to keep learning, based on the GD of the last lb^l batches. After PU, the precision switching mechanism returns an updated $\mathbb{Q}$ to the training loop, and a regular SGD weights update is applied to the float32 master copy $\mathbb{L}$, using de-quantized copies of gradients $\widehat{\mathbb{G}}$. The now updated weights $\mathbb{L}$ are quantized using the updated $\mathbb{Q}$ and written back to $\widehat{\mathbb{L}}$ to be used in the next forward pass.

4 Experimental Evaluation

Our approach is not limited to a specific architectural type or task, however, due to the numerous contemporary architecture-dataset combinations [44, 45], an exhaustive evaluation is unfeasible within a reasonable timeframe. Therefore, we selected a subset of experiments that align with our main competitor MuPPET's focus on image recognition tasks with convolutional neural networks, particularly ResNet and AlexNet, and popularity in the domain. [46, p. 62]

4.1 Setup For experimental evaluation of ASGD, we trained AlexNet and ResNet20 on the CIFAR-10/100 datasets, with reduce on plateau learning rate (ROP) scheduling, which will reduce the learning rate by a given factor if loss has not decreased for a given number of epochs for 100 epochs using 512 batch size. We further trained AlexNet and ResNet20 on EMNIST as well as LeNet-5 on MNIST for 15 epochs using a fixed learning rate schedule and 256 batch size. For all experiments, we used cross-entropy as loss function. Due to unavailability of fixed-point hardware, experiments were conducted on a DGX-1 GPU cluster, and we used QPyTorch to simulate fixed point arithmetic and our own performance model to estimate speed-ups, model size reductions and memory consumption. Our performance model, while assuming native hardware utilisation of the used precision levels, describes the entire training process (including e.g. precision switching heuristic overheads). The performance model computes the number of operations (MAdds) per layer, weighs them with the layer's word length and percentage of non-zero elements at a specific iteration during training to simulate quantization and a sparse tensor format, and aggregates them over all iterations to obtain the total computational cost of all forward passes and gradient computations and weight updates in the backward passes. Additionally, the performance model estimates ASGD's overhead for each layers PU and PD operations. This model is used to estimate both AdaPT's as well as MuPPET's performance, which ensures fair comparison. It also substitutes for lack of information about MuPPET's original performance model, which was not published by its authors. For reproducability, our performance model is included in our code repository[1].

Hyperparameters All layers $l \in \mathbb{L}$ were quantized with $\langle BW^l, FL^l \rangle = \langle 8, 4 \rangle$ at the beginning of ASGD training. Other hyperparameters specific to ASGD 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 ASGD (lr, $L1_{decay}$, $L2_{decay}$, $ROP_{patience}$, $ROP_{threshold}$, *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.

4.2 Results

4.2.1 Training As can be seen in Table 1, ASGD 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, which is not true for MuPPET. Our experiments furthermore illustrate ASGD delivers this performance independently of the underlying DNN architecture or the dataset used. We observe individual layers have different precision preferences depending on progression of the training (Figures 4 and 5), an effect that is particularly pronounced in AlexNet. For ResNet20 bit-width interestingly increased notably in the middle of training and later decreased, which we conjecture is attributable to regularization, see Section 3.2, leading to a reduction of irrelevant weights as training progresses. As illustrated in Table 2, ASGD achieves speedups comparable with state-of-the-art solutions on our own baseline (SU^1) and outperforms MuPPET on MuPPETs baseline (SU^3) in every scenario and on average by a factor of 4.26. Speedup adjusted for iso-accuracy SU^2 assumes training terminates once iso-accuracy is reached, which displays a particularly pronounced effect with (E)MNIST. Unfortunately, MuPPETs code base could not be executed, so we were unable to apply MuPPET to the more efficient

[1]`https://github.com/lorenz0890/AdaPT`

Table 1: Top-1 accuracies, ASGD (A) vs MuPPET (M), CIFAR10 (C10), CIFAR100 (C100) and (E)MNIST.Δ indicates difference to the respective 32-bit floating-point training (FL32 baseline)

Data	Network	Top-1	Δ	Iso-accuracy
C10	AlexNet$_A$	75.0	1.9	✓
	AlexNet$_M$	74.5	-1.0	×
	ResNet$_A$	90.0	0.5	✓
	ResNet$_M$	90.9	0.8	✓
C100	AlexNet$_A$	42.4	1.1	✓
	AlexNet$_M$	38.2	-1.0	×
	ResNet$_A$	65.2	0.9	✓
	ResNet$_M$	65.8	1.2	✓
EMNIST	AlexNet$_A$	83.9	1.2	✓
	ResNet$_A$	89.9	0.1	✓
MNIST	LeNet-5$_A$	98.3	0.0	✓

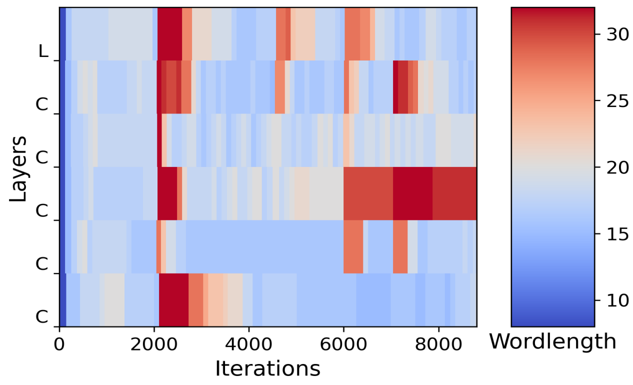

Figure 4: *Bit-widths (Wordlength) of ASGD optimized AlexNet on CIFAR100, 100 epochs, Convolutional (C) and Linear (L) layers*

ASGD baseline and were limited to comparing against results provided by the original authors. Hence, we used our performance model for simulating MuPPET's performance based on the precision switches stated in the MuPPET paper. An overview of the reduction of computational costs through ASGD relative to a float32 baseline is provided by Figure 6. The curves' oscilations in Figure 6 are caused by the dependency of computational costs on changes of the networks bit-widths, cf. Figures 5 and 4. As can be seen in Table 3, ASGD induces most sparsity in AlexNet, with a 44% final model sparsity and an average intra-training sparsity of 35% for training on CIFAR100. In ResNet, we observe that ASGD introduces up to 7% sparsity in the final model and 4% intra-training sparsity. A side effect of ASGD is the increased intra-training memory consumption (Table 2)caused by ASGD maintaining a float32 master-copy of the weights, as wells as the tracking of the accumulated gradients GD. However, this effect is only present during training as master-copies are discarded for inference and ASGD's memory footprint is lower than MuPPET's.

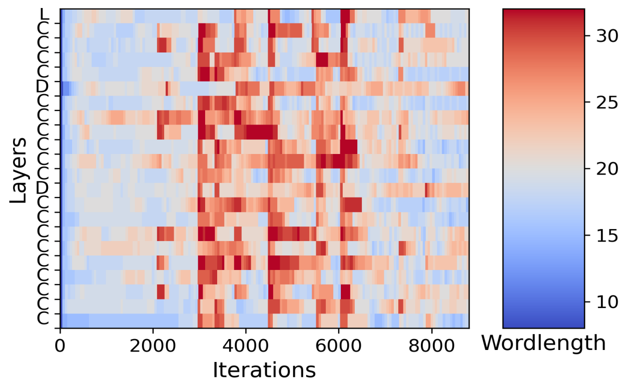

Figure 5: *Bit-widths (Wordlength) of ASGD optimized ResNet on CIFAR100, 100 epochs, Convolutional (C) and Linear (L) layers*

Table 2: Memory Footprint (MEM), Speedup (SU, including overhead), ASGD (A) vs MuPPET (M) on respective float32 CIFAR10 (C10), CIFAR100 (C100), (E)MNIST training baseline. SU[1]: FL32 baseline, our performance model, SU[2]: FL32 baseline, our performance model adjusted for iso-accuracy, SU[3]: MuPPETs FL32 baseline, our performance model (note a "?" indicates information missing in literature)

Data	Network	MEM	SU[1]	SU[2]	SU[3]
C10	AlexNet$_A$	**1.94**	1.42	2.37	**6.42**
	AlexNet$_M$	3.64	?	?	1.20
	ResNet$_A$	**2.26**	1.20	1.49	**4.01**
	ResNet$_M$	3.44	?	?	1.25
C100	AlexNet$_A$	1.82	1.32	1.57	5.23
	ResNet$_A$	1.66	1.26	1.62	3.36
EMNIST	AlexNet$_A$	1.68	1.49	**8.24**	?
	ResNet$_A$	2.20	1.22	**9.79**	?
MNIST	LeNet-5$_A$	1.94	1.42	**8.22**	?

Table 3: Fraction of zero weights (sparsity) induced by ASGD training, final model and intra-training average, CIFAR10 (C10) and CIFAR100 (C100), (E)MNIST

Data	Network	Final Model	Average
C10	AlexNet	0.26	0.27
	ResNet	0.07	0.04
C100	AlexNet	**0.44**	**0.35**
	ResNet	0.07	0.03
EMNIST	AlexNet	0.21	0.22
	ResNet	0.06	0.05
MNIST	LeNet-5	0.08	0.08

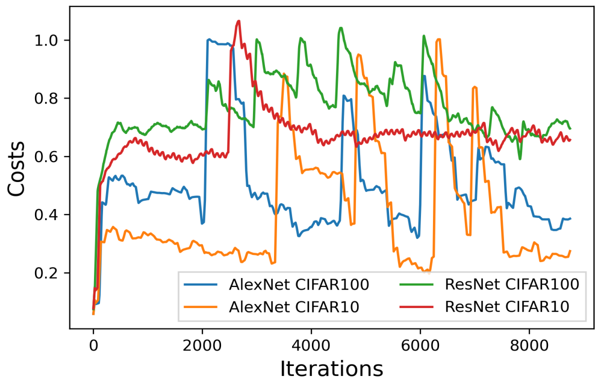

Figure 6: *ASGD computational cost relative to float32 SGD according to our performance model on CIFAR datsets*

Table 4: Inference with ASGD trained models, Final Model Size (SZ), Speedup (SU), CIFAR10 (C10) and CIFAR100 (C100), (E)MNIST

Data	Network	SZ	SU
C10	AlexNet	0.54	**3.56**
	ResNet	0.60	1.63
C100	AlexNet	0.36	2.60
	ResNet	0.57	1.52
EMNIST	AlexNet	0.35	2.98
	ResNet	0.62	1.46
MNIST	LeNet-5	0.52	2.18

4.2.2 Inference ASGD trained networks are fully quantized and sparsified. This advantage carries over to the inference phase. Table 4 shows that inference speed ups for ASGD trained networks range from 1.46 to 3.56. Inference speedup is higher than training, as it eliminates the costly backwards pass and ASGD overhead. While ASGD's inference acceleration may not be as competitive as low-precision approaches, it provides a significant advantage over MuPPET which does not offer post-training benefits in terms of speedup or memory consumption as the resulting network is float32.

5 Conclusion

We introduce AdaPT, a novel quantization method for DNN training acceleration. It employs a fixed-point representation for precision switching, and can be integrated with iterative gradient-based training. AdaPT balances efficient fixed-point operations for forward passes and gradient computation with high-precision float32 weight updates, to achieve optimal training speed while maintaining accuracy.

AdaPT outperforms MuPPET in mixed-precision training, but its inference acceleration is not competitive with state-of-the-art low-precision approaches. AdaPT still offers an advantage over MuPPET in inference acceleration and the inference speed ups are reported for completeness. Trade-offs between training and inference acceleration, memory consumption, and model accuracy are of interest, but considered as future work.

References

[1] A. Krizhevsky, I. Sutskever, and G.. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[3] Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization. *ICML*, 2019.

[4] M. Li, M. Soltanolkotabi, and S. Oymak. Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. *AISTATS*, 2020.

[5] M. Lis, M. Golub, and G. Lemieux. Full deep neural network training on a pruned weight budget. *MLSys*, 2019.

[6] 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*, 2021.

[7] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 1951.

[8] D. Yin, A. Pananjady, M. Lam, D. Papailiopoulos, K. Ramchandran, and P. Bartlett. Gradient diversity: a key ingredient for scalable distributed learning. *AISTATS*, 2018.

[9] A. Rajagopal, D. Vink, S. Venieris, and C. S. Bouganis. Multi-precision policy enforced training (MuPPET) : A precision-switching strategy for quantised fixed-point training of CNNs. *ICML*, 2020.

[10] T. Zhang, Z. Lin, G. Yang, and C. De Sa. QPyTorch: A low-precision arithmetic simulation framework. *NIPS*, 2019.

[11] 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. *CVPR*, 2018.

[12] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X. S. Hua. Quantization networks. *CVPR*, 2019.

[13] 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.*, 2021.

[14] D. Zhang, J. Yang, D. Ye, and G. Hua. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. *ECCV*, 2018.

[15] 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*, 2020.

[16] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, and M. Aleksic. A quantization-friendly separable convolution for mobilenets. *EMC2*, 2018.

[17] J. Achterhold, J .M. Köhler, A. Schmeink, and T. Genewein. Variational network quantization. *ICLR*, 2018.

[18] E. Tartaglione, S. Lathuilière, A. Fiandrotti, M. Cagnazzo, and M. Grangetto. HEMP: High-order entropy minimization for neural network compression. *Neurocomputing*, 2021.

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

[20] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. HAQ: Hardware-aware automated quantization with mixed precision. *CVPR*, 2019.

[21] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. QSGD: Communication-efficient sgd via gradient quantization and encoding. *NIPS*, 2017.

[22] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *NIPS*, 2017.

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

[24] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. *NIPS*, 2016.

[25] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 2017.

[26] F. Li, B. Zhang, and B. Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016.

[27] 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*, 2018.

[28] L. Hou and J. T. Kwok. Loss-aware weight quantization of deep networks. *ICLR*, 2018.

[29] Tianshu C., Qin L., Jie Y., and Xiaolin H. Mixed-precision quantized neural networks with progressively decreasing bitwidth. *Pattern Recognition*, 2021.

[30] Donghyun L., Dingheng W., Yukuan Y., Lei D., Guangshe Z., and Guoqi L. QTTNet: Quantized tensor train neural networks for 3d object and video recognition. *Neural Networks*, 2021.

[31] P. Peng, Y. Mingyu, X. Weisheng, and L. Jiaxin. Fully integer-based quantization for mobile convolutional neural network inference. *Neurocomputing*, 2021.

[32] T. Huang, L. Tao, and J. T. Zhou. Adaptive precision training for resource constrained devices. *ICDCS*, 2020.

[33] S. Wu, G. Li, F. Chen, and L. Shi. Training and inference with integers in deep neural networks. *CoRR*, abs/1802.04680, 2018.

[34] Y. Wang, Z. Jiang, X. Chen, P. Xu, Y. Zhao, Y. Lin, and Z. Wang. E2-train: Training state-of-the-art CNNs with over 80% energy savings. *NIPS*, 2019.

[35] J. Liu, H. Shao, Y. Jiang, and X. Deng. Cnn-based hidden-layer topological structure design and optimization methods for image classification. *Neural Processing Letters*, 2022.

[36] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. 2014.

[37] 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*, 2020.

[38] P. Zhou, J. Feng, C. Ma, C. Xiong, S. C. H. Hoi, and E. Weinan. Towards theoretically understanding why SGD generalizes better than adam in deep learning. *NIPS*, 2020.

[39] C. C. Aggarwal. *Neural Networks and Deep Learning*. Springer, 2018.

[40] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 2020.

[41] A. Y. Ng. Feature selection, L1 vs. L2 regularization, and rotational invariance. *ICML*, 2004.

[42] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 2005.

[43] B. Hanin and D. Rolnick. How to start training: The effect of initialization and architecture. *NIPS*, 2018.

[44] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 2019.

[45] L. Kummer. Dynamic neural network architectural and topological adaptation and related methods–a survey. *CoRR*, abs/2108.10066, 2021.

[46] L. Kummer. A multi-directional approach for accelerating single-node image classification neural network training via pruning. *u:theses*, 63874, 2022.